EasyBuilder X

Software User Manual

V1.01



Copyright © 2026 WEINTEK IIOT LTD. All rights reserved.

Table of Contents

1. Overview 1-1
   1.1. Core Concepts 1-2
   1.2. Object-Oriented Concept 1-2
   1.3. Modular Design 1-2
   1.4. Object-Oriented Project Design 1-3
2. Menu Introduction 2-1
   2.1. File Menu 2-2
      2.1.1. Preference 2-3
   2.2. Home Menu 2-4
   2.3. View Menu 2-5
      2.3.1. Panel 2-5
      2.3.2. Positioning 2-7
      2.3.3. Opacity 2-8
      2.3.4. Zoom / Alignment 2-8
3. Quick Start 3-1
   3.1. Creating a Simple EasyBuilder X Project 3-2
4. Device List & Server List 4-1
   4.1. Device List 4-2
      4.1.1. Steps to Create a Device 4-2
      4.1.2. Device Properties 4-2
      4.1.3. Device Events 4-3
      4.1.4. Device Methods 4-4
   4.2. Server List 4-5
      4.2.1. Steps to Create a Server 4-5
5. Data Type Library and Variable Library 5-1
   5.1. Namespace 5-2
      5.1.1. Why Namespaces Matter 5-2
      5.1.2. Definition of a Namespace 5-2
      5.1.3. Application Examples 5-2
      5.1.4. Rules and Limitations 5-3
   5.2. Data Type Library 5-3
      5.2.1. Data Type Classification 5-3
      5.2.2. Steps to Create a User-defined Data Type 5-4
   5.3. Variable Library 5-5
      5.3.1. Value Variable 5-6
      5.3.2. Arithmetic Variable 5-7
      5.3.3. Interaction between Variables and PLC Devices 5-9

5.4. Application Case: Temperature Monitoring System 5-9
6. Widgets 6-1
6.1. Overview of Widgets 6-2
6.2. Properties and Property Binding 6-2
6.2.1. Runtime Characteristics of Properties 6-2
6.2.2. Property Classification by Data Flow 6-2
6.2.3. Property Binding and Variable Access Rights 6-3
6.2.4. Data Synchronization Timing in Property Binding 6-3
6.3. Widget List 6-4
6.4. Property 6-5
6.4.1. Modify / Common Properties / Batch Modify 6-6
6.4.2. Introduction to Widget Properties 6-8
6.5. Method 6-19
6.5.1. Introduction to Widget Methods 6-19
6.6. Event 6-21
6.6.1. Introduction to Widget Events 6-21
7. Custom Model 7-1
7.1. Functions and Uses of Custom Models 7-2
7.2. Enabling the Interface and Panel Functions 7-2
7.3. Object-Oriented Structure (Property, Method, Event) 7-2
7.4. Creating a Custom Model 7-2
7.5. Custom Model Editing Interface 7-3
7.6. Function Settings and Examples 7-4
7.6.1. Custom Properties 7-4
7.6.2. Custom Methods 7-8
7.6.3. Custom Events 7-10
7.7. Deployment, Use, and Maintenance 7-13
7.7.1. Deploying a Custom Widget 7-13
7.7.2. Instance Concept 7-13
7.7.3. Centralized Maintenance and Synchronized Updates 7-13
8. String Library 8-1
8.1. String Library 8-2
8.1.1. Steps to Create a String 8-2
8.1.2. String Library Properties 8-2
8.2. Usage Instructions 8-3
8.2.1. Static Text Display 8-3
8.2.2. Dynamic Text Display 8-4
8.2.3. Export/Import String Table 8-5

8.2.4. Display Strings with Inserted Variables 8-6
8.2.5. Display Read-Only System Parameters 8-8
8.3. Multi-language Settings 8-9
8.4. Font Application Logic 8-11
9. Condition 9-1
9.1. Condition 9-2
9.1.1. Steps to Create a Condition 9-2
9.1.2. Condition Properties 9-2
9.1.3. Steps to Set a Condition Expression 9-2
9.1.4. Condition Event Description 9-4
9.2. Use Case: Temperature Monitoring System 9-4
9.2.1. System Overview 9-4
9.2.2. Create Variables 9-4
9.2.3. Create a Condition 9-4
9.2.4. Set the Condition Expression 9-5
9.2.5. Set Actions for Condition Satisfied and Unsatisfied 9-5
9.2.6. Operation Instructions 9-6
10. UAC 10-1
10.1. Overview of UAC 10-2
10.2. UAC Management Interface 10-2
10.3. UAC Property 10-2
10.4. UAC Method 10-3
10.5. Usage 10-5
10.5.1. Implementing Login/Logout 10-5
11. Data Logging 11-1
11.1. Data Sampling 11-2
11.1.1. Steps to Create Data Sampling 11-2
11.1.2. Data Sampling Properties 11-2
11.1.3. Data Sampling Events 11-3
11.1.4. Data Sampling Methods 11-4
11.2. Trend Display 11-4
11.2.1. Trend Display Properties 11-5
11.2.2. Trend Display Events 11-7
11.3. Data Display 11-7
11.3.1. Data Display Properties 11-7
12. Project Setting 12-1
12.1. Overview 12-2
12.2. General 12-2

12.3. Remote 12-2
12.4. Security 12-2
12.5. HMI Model 12-3
13. Alarm 13-1
13.1. Alarm 13-2
13.1.1. Steps to Create Alarm 13-2
13.2. Alarm Settings 13-3
13.2.1. Alarm Manager 13-3
13.2.2. Alarm Properties 13-5
13.2.3. Alarm Events 13-6
13.2.4. Alarm Methods 13-6
13.2.5. Alarm Class 13-6
13.2.6. Alarm Tag 13-8
13.3. Alarm Display 13-8
13.3.1. Alarm Display Properties 13-8
13.3.2. Alarm Display Methods 13-10
13.4. Alarm Bar 13-10
13.4.1. Alarm Bar Properties 13-10
14. Recipe 14-1
14.1. Recipe Database 14-2
14.1.1. Steps to Create Recipe Database 14-2
14.1.2. Recipe Database Properties 14-2
14.1.3. Element Properties 14-3
14.1.4. Recipe Database Methods 14-3
14.2. Recipe View 14-5
14.2.1. Recipe View Properties 14-5
14.2.2. Recipe View Events 14-6
14.3. Use Case: Managing Production Batch Parameters 14-6
15. Weincloud 15-1
15.1. Weincloud 15-2
15.1.1. Server Settings 15-2
15.1.2. Weincloud Properties 15-2
15.2. EasyAccess 2.0 15-2
15.2.1. EasyAccess 2.0 Properties 15-3
15.2.2. EasyAccess 2.0 Method 15-3
15.3. Dashboard 15-4
15.3.1. Configuration Steps 15-4
15.3.2. Dashboard Properties 15-5

15.3.3. Dashboard Event 15-5
15.3.4. Dashboard Method 15-6
15.3.5. Dashboard Tag Properties 15-7
16. Macro 16-1
16.1. Overview 16-2
16.2. Macro Editor 16-2
16.3. Configuration 16-3
16.4. Syntax 16-5
16.4.1. Constants and Variables 16-5
16.4.2. Operators 16-8
16.5. Statements 16-10
16.5.1. Definition Statements 16-10
16.5.2. Assignment Statements 16-10
16.5.3. Logical Statements 16-11
16.5.4. Select-Case Statements 16-12
16.5.5. Loop Statements 16-14
16.6. Subfunctions 16-16
16.7. Built-In Functions 16-18
16.7.1. Function List 16-18
16.8. How to Create and Execute Macros 16-22
16.8.1. Creating a Macro 16-22
16.8.2. Executing Macros 16-22
16.9. User-Defined Function Library 16-23
16.9.1. Function Library 16-23
16.9.2. Using the Function Library in Macros 16-24
16.10. Compilation Error Messages 16-24
17. Picture Library 17-1
17.1. Overview 17-2
17.2. Picture Selection Menu 17-2
17.2.1. Category Tree 17-2
17.2.2. Search and Filter 17-3
17.2.3. View Modes 17-3
17.3. Managing Search Paths for External Picture Libraries 17-3
17.3.1. Picture Library Search Paths 17-3
17.3.2. Path Recording and Auto-Repair Mechanism 17-3
17.3.3. Picture Library Categories 17-4
17.4. Managing the Project Picture Libraries 17-4
17.4.1. This Project 17-4

17.4.2. In Use 17-5
17.5. Picture Panel 17-5
17.5.1. Interface Functions 17-5
17.5.2. Drag and Drop Behavior 17-5
18. cMT Viewer 18-1
18.1. cMT Remote 18-2
18.1.1. Opening cMT Remote 18-2
18.2. cMT Remote Interface 18-2
18.2.1. cMT Remote Settings 18-2
19. Modbus TCP/IP Server 19-1
19.1. Modbus Server Settings 19-2
19.2. Mapping Table 19-2
19.2.1. Address Types and Variable Limits 19-2
19.2.2. Notes and Conflict Rules 19-2
19.2.3. Mapping Examples (Address Browser View) 19-3
19.3. Address Base 19-4
19.4. Enable In-Memory Data 19-4
20. Audit Trail 20-1
20.1. Overview 20-2
20.2. Settings 20-2
20.2.1. Audit Trail Properties 20-2
20.2.2. Audit Trail Events 20-2
20.2.3. Audit Trail Methods 20-4
20.3. Usage 20-5
20.3.1. Variable Audit Trail 20-5
20.3.2. Recipe Database Audit Trail 20-6
20.3.3. Audit Trail Usage Limitations 20-7
20.4. Audit Trail Viewer 20-7
20.4.1. Audit Trail Viewer Properties 20-7
20.5. Backup and Verification 20-9
20.5.1. Backing Up Audit Trail Files 20-9
20.5.2. Verifying Audit Trail Files 20-9
21. System Object 21-1
21.1. Overview 21-2
21.2. General 21-2
21.2.1. General Properties 21-2
21.2.2. General Events 21-3
21.2.3. General Methods 21-3

21.3. Network 21-3
21.3.1. Network Properties 21-3
21.3.2. Network Methods 21-4
21.4. Backlight 21-4
21.4.1. Backlight Properties 21-4
21.4.2. Backlight Event 21-5
21.5. Remote 21-5
21.5.1. Remote Properties 21-5
22. Variant Scope 22-1
22.1. Overview of Variant and Scope 22-2
22.2. Differences Between Value variables and Arithmetic Variables 22-3
22.3. Compatibility Rules for Variant Binding and Configuration 22-3
22.4. Dynamic Resolution of Arithmetic Variables and Examples 22-4
22.5. Common Errors and Debugging Suggestions 22-5
23. cMT Inspector 23-1
23.1. Overview 23-2
23.1.1. Opening cMT Inspector 23-2
23.2. cMT Inspector Interface 23-3
23.2.1. Object 23-3
23.2.2. Variable 23-3
23.2.3. Device 23-3
23.2.4. JS 23-4

Other company names, product names, or trademarks in this document are the trademarks or registered trademarks of their respective companies.
This document is subject to change without prior notice.

1. Overview

1.1. Core Concepts 1-2
1.2. Object-Oriented Concept 1-2
1.3. Modular Design 1-2
1.4. Object-Oriented Project Design 1-3

1.1. Core Concepts

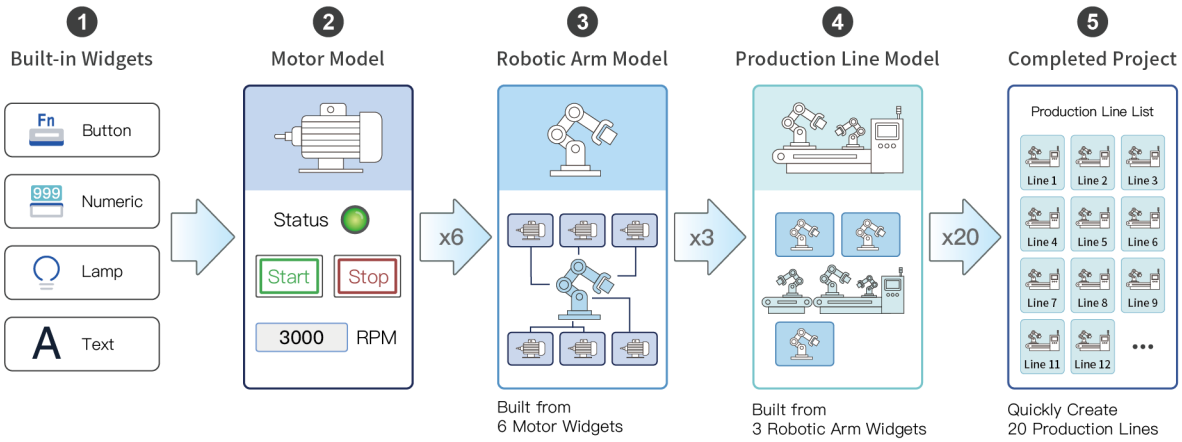
EasyBuilder X is a graphic design software that incorporates the "object-oriented" concept from programming, allowing for "modular design" of projects. With modular design, projects become easier to understand, test, and maintain, and multiple developers can work on different parts of a project simultaneously, thus reducing overall development time. EasyBuilder X's unique "binding" mechanism allows for flexible and intuitive defining of relationships and cooperation between Widgets. This efficient workflow meets the modern user's demand for rapid development of graphic design software.

1.2. Object-Oriented Concept

In EasyBuilder X, an object can be anything, such as a Device, Screen, or Widget. Take a smartphone as an example: it consists of multiple components, including the case, screen, camera, flash, motherboard, etc. These components can be further subdivided into smaller parts, each considered a new component, forming a nested structure. If the camera on the phone has an issue, only the camera or a specific lens within it needs to be replaced, rather than the entire phone. This modular design concept also applies to EasyBuilder X. When a project is designed modularly, any required modifications can be made to the specific module without affecting the entire project, significantly accelerating the development speed of large projects.

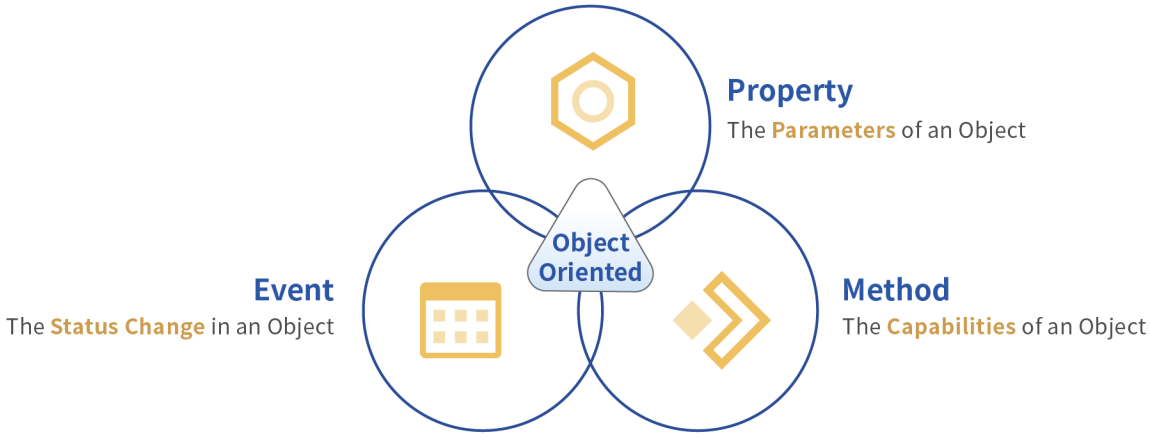
1.3. Modular Design

Modular design is a key trend in modern software development. In EasyBuilder X, users can create Custom Models by combining several to dozens of Widgets based on their needs and reuse these Custom Widgets throughout the project. For instance, to display the status of 20 production lines in an HMI project, start by designing a Custom Model called "Motor" using the built-in Widgets of EasyBuilder X. Use this Custom Model to create six "Motor" Custom Widgets, then combine them with other Widgets to form a new Custom Model called "Mechanical Arm". Next, use ten "Mechanical Arm" Custom Widgets to create another Custom Model named "Production Line". This method enables the rapid establishment of 20 production lines in the project. It not only enhances project flexibility and maintainability but also significantly reduces repetitive work during development and maintenance.



1.4. Object-Oriented Project Design

Object-oriented project design is a development method that breaks down an HMI application into small, independent, and reusable Objects. These independent Objects are then reorganized or related to form a complete project. In this design method, Property, Method, and Event are the basic elements for describing and operating Objects:



- **Property**

Properties represent the "state" or "characteristics" of an Object. For example, a Button Widget's properties might include length, width, number of states, color, and so on.

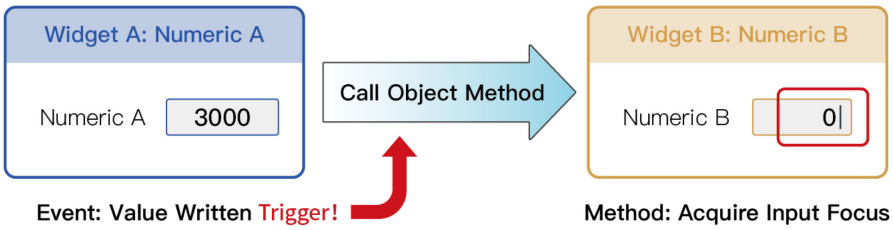
- **Method**

Methods represent the "capabilities" of an Object. By calling an Object's Method, it can perform specific actions. For example, calling the "Acquire input focus" method of a Numeric widget causes it to enter the input state.

- **Event**

Events serve as a "notification mechanism" to trigger pre-defined actions (set by the project

developer) when specific conditions are met. For example, if there are two Numeric widgets, Numeric A and Numeric B, and Numeric B needs to automatically enter the input state (pre-defined action) once the user finishes inputting in Numeric A (specific condition)—without requiring a click on Numeric B on the screen—add a "Call object method" action to the "Value written" event of Numeric A and select the "Acquire input focus" method for Numeric B.



The design philosophy of EasyBuilder X is based on these modern project design requirements, providing users with a more efficient, flexible, and convenient graphic editing experience. EasyBuilder X is expected to be a powerful assistant in the field of industrial control, aiding in the completion of various projects more quickly and easily.

2. Menu Introduction

2.1. File Menu 2-2
2.2. Home Menu 2-4
2.3. View Menu 2-5

2.1. File Menu

This section provides descriptions for each item in the File menu.

Icon	Name	Description
	**New**	Create a new project file with options to select an HMI model for screen editing.
	**Open**	Open an existing project file.
	**Save**	Save the currently open project file using the same path and name.
	**Save As**	Save the currently open project file with a different path and name.
	**Close**	Close the current project file.
	**Decompile**	Decompile an .ebxobj file into an editable .ebxprj project file and open it in EasyBuilder X.
	**Languages**	Switch the interface language of EasyBuilder X.
	**Preference**	Set various attributes for editing project files. See the table below for further details.
	**About**	Display EasyBuilder X version information and copyright details.

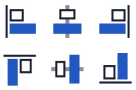
2.1.1. Preference

Click [File] » [Preference] to open the settings window.

Setting	Description
General	
Automatically save and compile the project when download and simulate	When enabled, clicking Download/Simulate automatically saves and compiles the project. This merges the original three-step process—Save → Compile → Download/Simulate—into a single Download/Simulate action.
Display group titles at the bottom	Set whether to display group names of functions in the graphical user interface.
Ruler	When enabled, rulers appear at the top and left of the editing window, allowing addition of alignment guide lines. Widgets will snap to these guide lines when moved or resized.
Grid	When enabled, display a grid in the window.
Smart Guide	When enabled, alignment lines appear as the user drags a Widget, helping it align with other Widgets vertically or horizontally.
Pin all widgets	When enabled, Widgets cannot be resized or moved with the mouse after being placed in the window.
Layer Opacity	Adjust the visibility of Widgets in different window layers. The higher the value, the clearer the visibility.
CSV delimiter	Specify the separator used when importing or exporting the String Table.
Picture Library	
Picture library search path	When the project is opened, the system searches the specified path for related picture libraries.
Picture library category	Enable Picture Library categories and sort pictures by category.

2.2. Home Menu

This section provides descriptions for each item in the Home menu.

Icon	Name	Description
	**Paste**	Paste the content from the clipboard into the project file.
	**Cut**	Cut the selected Widget from the project file and place it in the clipboard.
	**Copy**	Copy the selected Widget from the project file and temporarily place it on the clipboard. Alternatively, select the widget, hold Ctrl, and drag it with the left mouse button to create a copy.
	**Format Painter**	Copy the settings of a label to other Widgets.
	**Select**	Select or drag Widgets in the window.
	**Hand Tool**	Click and drag the window to view different parts of the screen.
	**Undo**	Revert the last action.
	**Redo**	Re-apply the last reverted action.
	**Alignment**	Align multiple selected Widgets to the last selected Widget.
	**Space Evenly**	Distribute multiple selected Widgets evenly either horizontally or vertically.
	**Create Custom Model / Decompose Custom Widget**	Convert the selected Widgets into a custom model, or decompose a custom model back into individual Widgets.
	**Pin**	Lock the position of a Widget, preventing it from being resized or moved. Locked Widgets display an icon  in the upper-right corner.

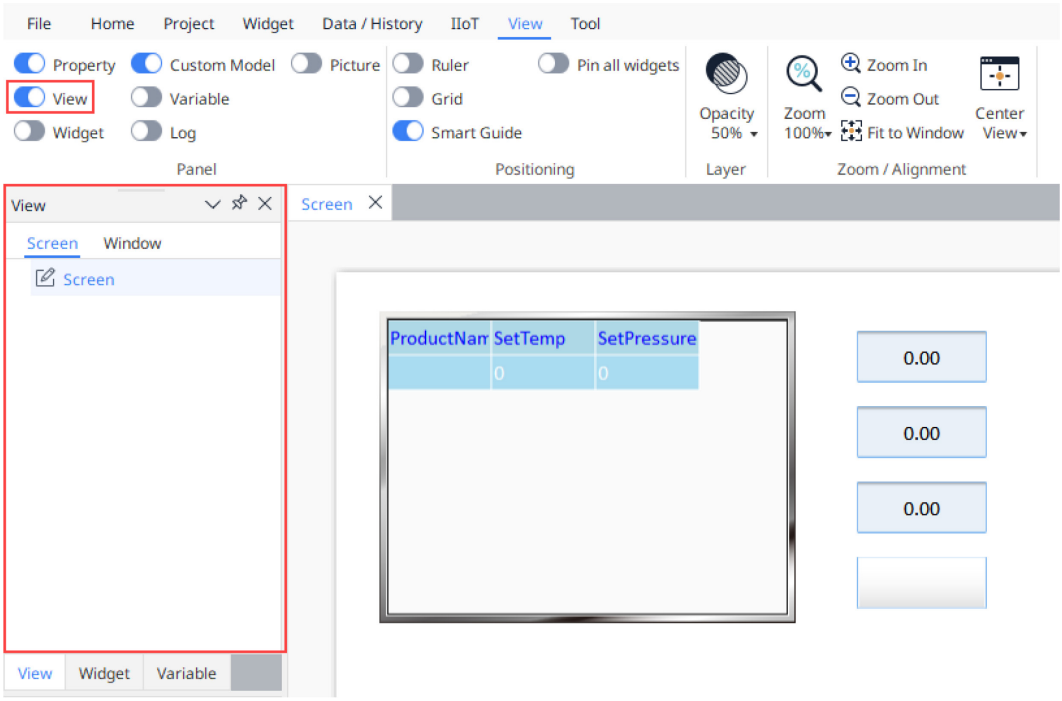
	**Make Same Size**	Set the width and/or height of multiple selected Widgets to match the last selected Widget.
	**Center in Window**	Move the selected Widget to the center of the window.
	**Resize**	Adjust the width or height of the selected Widget by one pixel each time the Resize button is clicked.
	**Nudge**	Move the selected Widget one pixel in the specified direction each time the Nudge button is clicked.
	**Layer**	Change the stacking order of Widgets.
	**Rotate**	Rotate the selected Widget by 90 degrees.
S0 S1 S2 S3	**State**	Preview the state of the Widget during editing.
L1 L2 L3 L4	**Language**	Preview the language displayed during editing.

2.3. View Menu

In EasyBuilder X, the [View] menu is an essential tool for designing interfaces. Through the [View] menu, various features and design elements can be easily managed and adjusted. By effectively using the functions in the [View] menu, it is possible to more flexibly handle and optimize the design process, enhancing the overall project quality and user experience. Below is an introduction to the functions in each settings group.

2.3.1. Panel

The toggle switches in the Panel settings group show or hide the corresponding panels. These panels can also be dragged to suitable positions for easier operation in EasyBuilder X.
For example, when the [View] panel is enabled, it appears as shown below:



2.3.1.1. Property

Enabling the [Property] option opens the [Property] panel, displaying the selected Widget's [Property], [Method], and [Event].

2.3.1.2. View

Enabling the [View] option opens the [View] panel, displaying all Screens and Windows created in the current project.

[Screen]: Used as a standard base screen. It is always displayed in full screen and can be switched using the [Change screen] action.

[Window]: Used as a pop-up window. It can be opened using the [Popup window] action.

2.3.1.3. Widget

Enabling the [Widget] option opens the [Widget] panel, displaying all the Widgets created in the current editing screen / window.

2.3.1.4. Custom Model

Enabling the [Custom Model] option opens the [Custom Model] panel, displaying all the Custom Models created in the current project.

2.3.1.5. Variable

Enabling the [Variable] option opens the [Variable] panel, displaying the variables created in the current project.

2.3.1.6. Log

Enable the [Log] option to open the Log panel, where different log sources, such as design log and compilation log, can be selected for display.
Design logs accurately record the deletion, warnings, and errors of each Widget, resetting each time the project is opened, with a maximum of 1000 entries. Exceeding this number will delete the oldest logs.
Compilation logs record messages, warnings, and errors during project compilation, also with a maximum of 1000 entries. Exceeding this number will delete the oldest logs. If an error occurs after compilation, double-click the error message to jump directly to the related setting.

2.3.1.7. Picture

Enabling the [Picture] option opens the [Picture] panel, which displays the contents of the Picture Library.

2.3.2. Positioning

The toggle switches in the [Positioning] settings group help users quickly and accurately locate the positions of various Widgets, providing convenient features to assist in project design. The following details the functions in this settings group.

2.3.2.1. Ruler

Enabling the [Ruler] function displays rulers on the top and left of the editing window. Users can right-click on the ruler to add alignment guides, which greatly assist in aligning the Widgets. When dragging or resizing Widgets, their positions will automatically snap to the alignment guides.
The color and style of the guides can be changed in the main menu under [File] » [Preferences] » [Positioning].

2.3.2.2. Grid

Enabling the [Grid] function displays grid points in the editing window, which assist in aligning the Widgets. When dragging or resizing Widgets, their positions will automatically snap to the grid points.
The color and distance of the grid points can be changed in the main menu under [File] » [Preferences] » [Positioning].

2.3.2.3. Smart Guide

Enabling the [Smart Guide] function displays alignment guides when a widget is dragged and aligns horizontally or vertically with other widgets.

2.3.2.4. Pin all widgets

Enabling the [Pin all widgets] function fixes Widgets in their current positions until the function is disabled. When fixed, an icon will appear in the upper right corner of the Widgets, indicating they are fixed and cannot be resized by dragging.

2.3.3. Opacity

Adjust the visibility of Widgets in different window layers. The higher the value, the clearer the visibility.

2.3.4. Zoom / Alignment

The [Zoom / Alignment] settings group provides a series of tools to help users precisely adjust the screen during project design, enhancing detail adjustments. The following details the functions in this settings group.

2.3.4.1. Zoom

The [Zoom] function allows selecting the zoom ratio of the screen from 50% to 200%. Click [Zoom In] or [Zoom Out] to adjust the screen zoom. Alternatively, use [Ctrl + Mouse Wheel] to zoom in and out. To return to the original 100% ratio, use the shortcut key [Ctrl + 0]. Clicking [Fit to Window] adjusts the zoom ratio to the system's recommended value.

2.3.4.2. Center View

The [Center View] function moves the editing window to the center point and the four corners of the display area.

3. Quick Start

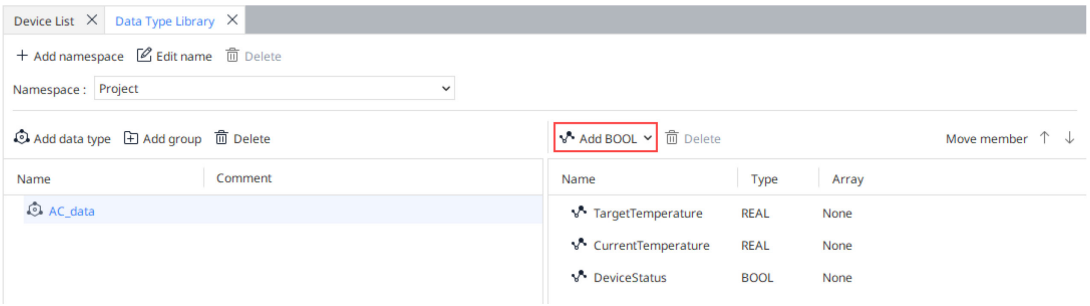
3.1. Creating a Simple EasyBuilder X Project 3-2

3.1. Creating a Simple EasyBuilder X Project

This section will guide you step-by-step on how to create a simple EasyBuilder X project. To simplify the explanation, this project will only display the status of an air conditioning unit (Modbus TCP/IP device), including a power indicator and two values representing the target temperature and current temperature.

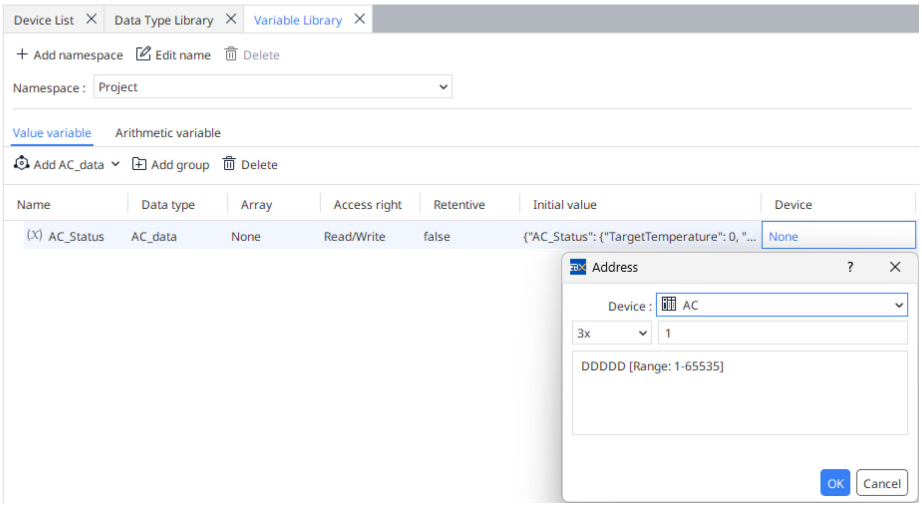
Steps:

1. Create a new project.
   a. Launch EasyBuilder X.
   b. Select the "Create" tab to start a new project.
   c. Choose the HMI model; in this example, cMT2078X is used. Click [Create].
2. Set up Modbus communication.
   a. Click [Project] » [Device List] to open the Device List settings page.
   b. In the Device List, click [Add device], select a new MODBUS TCP/IP device, and then double-click to rename it to AC.
   c. Set the correct IP address and port number on the Property page.
3. Add User-defined Data Type.
   a. Click [Project] » [Data Type] to open the Data Type Library settings page. Select the Namespace where you want to create the Data Type. In this example, choose the “Project” Namespace.
   b. In the Data Type Library, click [Add data type] to create a new Data Type, and then double-click to rename it to AC_data.
   c. The Members and their Types for AC_data are displayed on the right side.
   d. Click [Add] data type button or use the dropdown arrow to select other types and sequentially add three Members.
      Add Member TargetTemperature, type REAL.
      Add Member CurrentTemperature, type REAL.
      Add Member DeviceStatus, type BOOL.

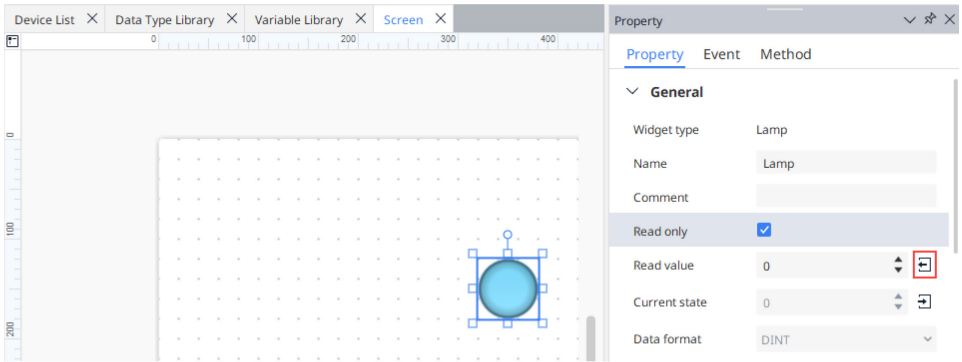


4. Add Variable.

a. Click [Project] » [Variable] to open the Variable Library settings page.
b. Select the Namespace where you want to create the Variable.
c. In the Variable Library, click the dropdown next to [Add] and add a Variable of Structed Data Type AC_data.
d. The interface will display the Variables in the Namespace.
e. Double-click to rename the Variable to AC_status.
f. Configure the air conditioning unit data: double-click the [Device] field to bind this Variable to the address of the device added in step 2, in this example, 3x-1.



5. Create power indicator.
a. On the screen, add a "Lamp" Widget by clicking [Widget] » [Lamp].
b. In the Lamp's Property, click the bind button to bind the read value to DeviceStatus under AC_status created in step 3. This Variable corresponds to the power status of the air conditioning unit.



c. Select [Widget] » [Draw\Text] to add a Widget and set its text to DeviceStatus to help identify the Widget on the screen.

6. Display target temperature and current temperature.
a. On the screen, add two "Numeric" Widgets by clicking [Widget] » [Numeric].
b. In the Numeric's Property, bind the Value to TargetTemperature and CurrentTemperature under AC_status created in step 3.

c. Select [Widget] » [Draw\Text] to add two Widgets and set their texts to TargetTemperature and CurrentTemperature to help identify the Widgets on the screen.

7. Save and simulate.

a. Save the project: click [File] » [Save], then [Project] » [Compile].

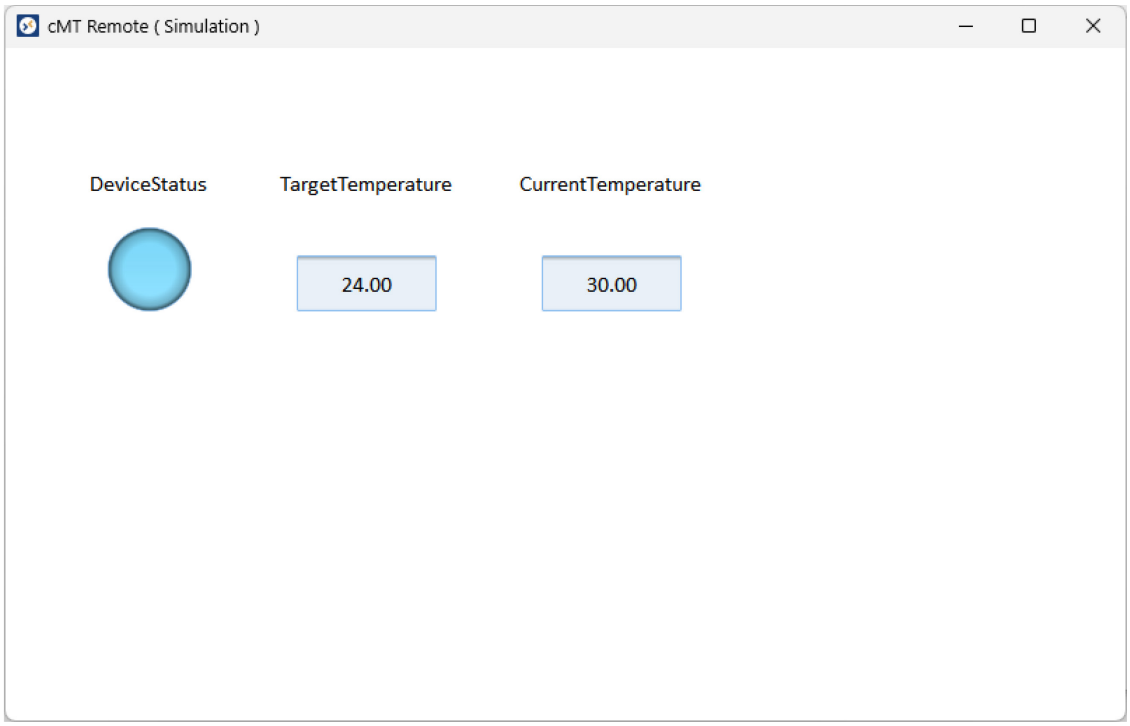
b. Offline/Online Simulation: Select [Project] » [Offline Simulation] to start the simulation, establish a connection with the air conditioning unit, and view the status and set the target temperature.

8. Download to the HMI.

a. Save the project: click [File] » [Save], then [Project] » [Compile].

b. Download the project: click [Project] » [Download].

c. Before downloading, enter the HMI user name (admin) and password.



4. Device List & Server List

4.1. Device List 4-2
4.2. Server List 4-2

4.1. Device List

The Device List is used to define external communication devices that the HMI actively connects to, such as PLCs, controllers, and sensors.
Click [Project] » [Device List] to open the Device List settings page. Devices can be added on this page to enable communication between the HMI and connected external devices.

4.1.1. Steps to Create a Device

1. Click [Project] » [Device List] to open the Device List settings page.
2. In the Device List, click [Add device] and choose the device type.
3. The added device will appear in the list.

Setting	Description
Add device	Add a device.
Add group	Add a group for easier device classification and management.
Delete	Delete the selected device or group.
Import tags	Available if the device supports tag file import.
Import tags from device	Available if the device supports direct connection for retrieving tag information.

4.1.2. Device Properties

Setting	Description
General	
Name	The device name. A custom name can be assigned for easier identification.
Comment	Add or modify the comment for the device.
Enabled	Set whether communication with the device is enabled.
Interface	
Interface	Set the communication interface and parameters for the device.
Device Settings	
Station number	Set the station number of the device.
Timing	
Connect timeout (sec)	Set the timeout for establishing a connection. If the connection is not established within this time, it is treated as a failure.
Request timeout (ms)	Set the wait time for a response to a single request. If the timeout is reached, the request is treated as failed.

Inter request delay (ms)	Set the interval between two requests.
Retry attempts	Set the number of retry attempts after a request fails.
Block Size Settings	
Interval of block pack (elements)	Set the address interval that can be merged into a single read command.
Outputs	
Connection status	Display the connection status with the device.
Last error code	Display the last communication error code.
Unsolved error	Display true if an unresolved error exists.

4.1.3. Device Events

The system monitors device communication and reports connection status changes and communication errors. Detailed event descriptions are provided below.

Connection Status Changed

Triggered when the connection status changes.

This event can be used to trigger specific actions. For example:

When the status is Disconnecting (1), display a warning or notify the user to check the device.

Parameter	Status (UINT): Indicates the device connection status.
Status Values and Descriptions	0: Disabled: Communication with the device is disabled.
	1: Disconnecting: The HMI is disconnecting from the device.
	2: Disconnected: The HMI is disconnected from the device.
	3: Connecting: The HMI is connecting to the device.
	4: Connected: The HMI is connected to the device.
	5: Faulted: A communication error has occurred.

Connected

Triggered when the connection status changes to Connected.

Disconnected

Triggered when the connection status changes from Connected to Disconnected.

Fault Occurred

Triggered when the connection status changes to Faulted (latched).

Error Codes (UDINT):

Code	Description
0	No error.
10000	An unknown device error occurred.
10400	Network unreachable.
10401	Host unreachable.
10402	Connection refused.
10403	Connection timed out.
10500	Serial port not found.
10501	Serial port permission denied.
10502	Serial mode setup failed.
10600	Socket address already in use.
10601	Socket permission denied.
10700	Connection failed.
10701	Connection lost.
10702	The device stopped responding.
10800	Initialization command failed.
10801	Initialization timed out.
10802	No response to the connection probe.
15000	Invalid connection setting.
15001	Internal driver error.
15002	Invalid certificate time.
15003	Certificate verification failed.
15004	Authentication failed.

4.1.4. Device Methods

The Device List provides the following methods for updating the interface configuration online or clearing errors. Detailed method descriptions are provided below.

Method	Description
Update interface config	Read the interface parameters and reconnect.
Clear fault	Clear the latched Faulted state.

4.2. Server List

The Server List is used to configure how the HMI provides data to external clients, allowing SCADA systems, other HMIs, or computers to read internal HMI variables through the specified protocol.

4.2.1. Steps to Create a Server

1. Click [Project] » [Server List] to open the Server List settings page.
2. In the Server List, click [Add server] and choose the server type.
3. After the server is created, it appears in the list, and the interface and mapping table can then be configured.

Note: To ensure communication performance and connection stability, server creation must follow the limitations below:

Interface usage limit: To ensure stable communication, only one server can be enabled for the same protocol on a single physical network interface, such as Ethernet.

Example: If a Modbus TCP Server has already been created on the Ethernet interface, a second Modbus TCP Server cannot be created on the same interface.

Setting	Description
Add server	Add a server.
Add group	Add a group for easier server classification and management.
Delete	Delete the selected server or group.
Name	The server name. A custom name can be assigned for easier identification.
Server type	Display the server type.
Comment	Add or edit the comments for the server.
Interface	Set the communication interface and parameters for the server.

5. Data Type Library and Variable Library

5.1. Namespace 5-2
5.2. Data Type Library 5-3
5.3. Variable Library 5-5
5.4. Application Case: Temperature Monitoring System 5-9

5.1. Namespace

Before using the Data Type Library and Variable Library, it is important to understand what a namespace is and why EasyBuilder X introduces this concept.
By using namespaces effectively, EasyBuilder X helps organize project structures more logically, significantly improving modular reuse and maintenance efficiency.

5.1.1. Why Namespaces Matter

In cross-industry, multi-site, or modular development scenarios, developers often use intuitive and common names for data types or variables. However, the same name may represent completely different meanings or data structures in different contexts. Without proper isolation, duplicate names can cause conflicts or errors, making the system difficult to manage. Namespaces solve this problem by grouping and isolating names, allowing the system to distinguish between identical names used in different contexts.

5.1.2. Definition of a Namespace

A namespace can be thought of as an independent container or folder.

- Inside a namespace:
  All data type and variable names must be unique. Duplicate names are not allowed.
- Across namespaces:
  The same name can be reused in different namespaces. Each namespace is independent and does not interfere with others.

This means there is no need to create long names such as Tank_EnergyStorage or Tank_FoodProcess to avoid naming conflicts; instead, simpler naming can be used.

5.1.3. Application Examples

The following examples illustrate how namespaces solve real-world problems.
Scenario 1: Data Types Across Industries
Assume a system involves both an **Energy Storage System** and a **Food Processing System**.
Both use a concept called "*Tank",* but their definitions are very different.

- **Energy Storage Tank:** May include electrolyte level, temperature, and pH value.
- **Food Processing Tank:** May include mixing speed, heating temperature, and ingredient weight.

Solution using namespaces:

- Namespace: EnergyStorage → Data Type: Tank
- Namespace: FoodIndustry → Data Type: Tank

Although both are named Tank, the system treats them as EnergyStorage.Tank and

FoodIndustry.Tank, completely independent with no conflict.

Scenario 2: Variable Management Across Workshops

Assume there are multiple workshops, each monitoring utilization rate.

- Workshop A has a variable named UtilizationRate.
- Workshop B also has a variable named UtilizationRate.

Solution using namespaces:

- Namespace: Workshop_A → Variable: UtilizationRate
- Namespace: Workshop_B → Variable: UtilizationRate

This approach allows screen objects to be copied and pasted easily. Only the namespace needs to be switched, without renaming large numbers of variables.

5.1.4. Rules and Limitations

When using namespaces in EasyBuilder X, note the following:

- Uniqueness:
  Names within the same namespace must be unique.
- Reusability:
  The same name can exist in different namespaces.
- Structural limitation:
  Nested namespaces are not supported. All namespaces exist at the same level.
- Creation method:
  A namespace can be created by clicking [Add namespace] in the Data Type Library or Variable Library settings, and then data types or variables can be assigned to the desired namespace.

5.2. Data Type Library

Click [Project] » [Data Type] to open the Data Type Library settings page.
The Data Type Library in EasyBuilder X is a collection of data types used to define data structures. Ranging from basic data types to complex data structures, this library provides developers with powerful flexibility and extensibility.

5.2.1. Data Type Classification

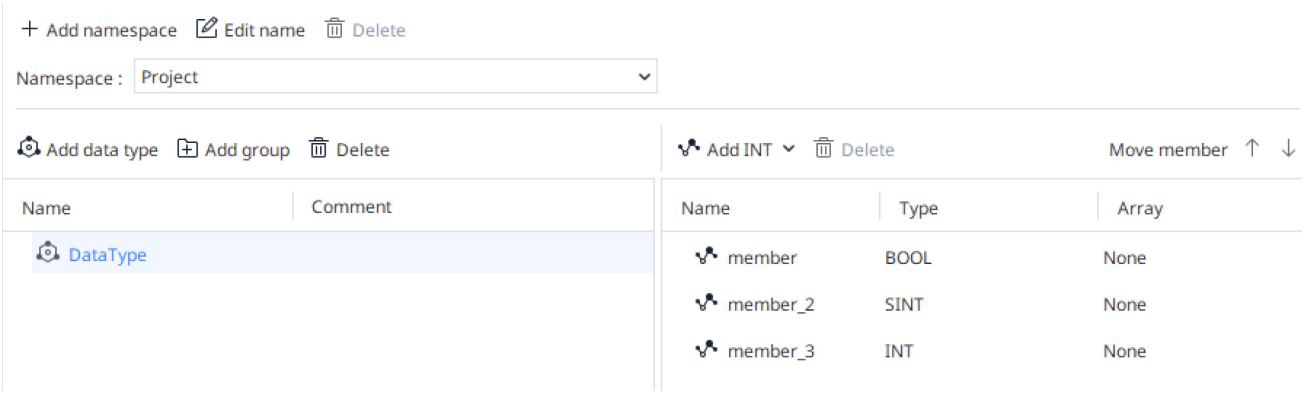
In EasyBuilder X, data types are classified into three categories: Primitive Data Type, Built-in Data Type, and User-defined Data Type.

Data Type	Description
Primitive Data Type	Supports basic data types, including BOOL,

	BYTE, WORD, DWORD, LWORD, SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, STRING, WSTRING, TIME, LTIME, DATE_AND_TIME, LDATE_AND_TIME, etc.
Built-in Data Type	Includes structured data types commonly used in the system such as Animation, Color, Brush, and Range.
User-defined Data Type	Used to define complex variable data structures to meet specific application requirements.

5.2.2. Steps to Create a User-defined Data Type

1. Click [Project] » [Data Type] to open the Data Type Library settings page.
2. Select the Namespace for the new Data Type.
3. In the Data Type Library, click [Add data type] to create a new Data Type.
4. The right-side interface will display the members and their types for the new Data Type.
5. Click [Add] or use the dropdown arrow to select and add a member of another type.
6. Double-click to edit the Name, Type, and other properties.



Namespace Properties	Description
Add namespace	Add a new Namespace. Namespaces allow Data Types with the same name to be reused in different contexts to meet various requirements.
Edit name	Edit the name of the Namespace.
Delete	Delete the selected Namespace.
Namespace	Select the Namespace. This allows Data Types to be organized and managed under different Namespaces for better structuring and modularity.

User-defined Group Properties	Description
Add data type	Add a new User-defined Data Type.
Add group	Add a Group to categorize Data Types for easier management.
Delete	Delete the selected Data Type or Group.
Name	The name of the Data Type.
Comment	Add or modify the comment for the Data Type.
User-defined Data Type Properties	**Description**
Add (data type)	Add a Member with a specified Data Type. The default Data Type for the Member can be selected and adjusted from a dropdown menu.
Delete	Delete the selected Member.
Move member	Adjust the order of Members within the Data Type.
Name	The name of the Member.
Type	The Data Type of the Member. This can be a Primitive Data Type, Built-in Data Type, or User-defined Data Type. Members cannot be created in a way that forms circular references with other Data Type members.
Array	Specify whether the Member is defined as an array.
Comment	Add or modify the comment for the Member.

5.3. Variable Library

Click [Project] » [Variable] to open the Variable Library settings page.
The Variable Library in EasyBuilder X is a feature used for storing and managing variables. This library allows users to create, edit, and delete variables and associate them with data types in EasyBuilder X. Within the Variable Library, there are two types of variables: Value Variables and Arithmetic Variables. Value Variables are used to store basic data, while Arithmetic Variables allow for simple arithmetic operations.

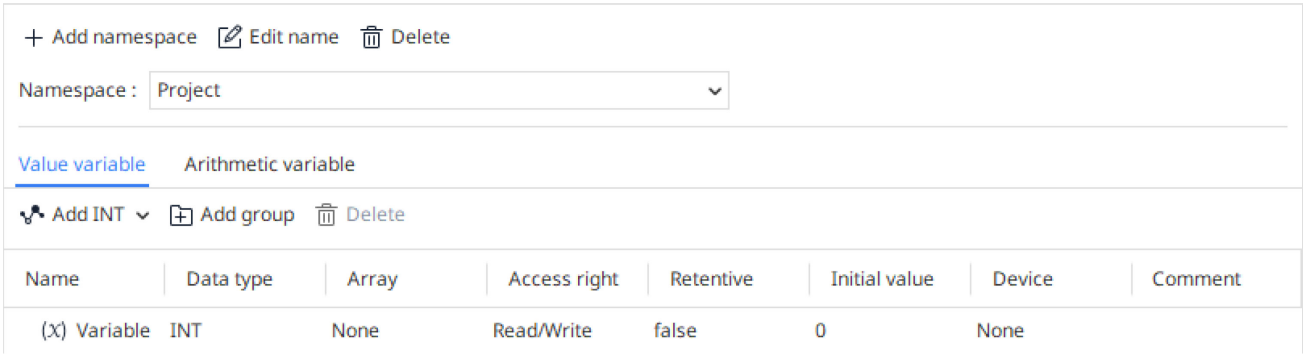
5.3.1. Value Variable

5.3.1.1. Overview

A Value Variable is used to store and manage various basic data types, such as integers (INT), floating-point numbers (REAL), and boolean values (BOOL). These variables can be used to represent and access different types of data within the system and can be bound to device addresses for data communication.

5.3.1.2. Steps to Create a Value Variable

1. Click [Project] » [Variable] to open the Variable Library settings page.
2. Select the Namespace for the new Variable.
3. In the [Value variable] tab, click [Add] to create a new Variable.
4. The interface will display the Value variables within the selected Namespace.
5. Double-click to edit the Name, Data type, and other properties.



5.3.1.3. Binding to Device Addresses

1. Prior to binding, a device must be created in the Device List.
2. Double-click the Device property field and configure the device address to which the Variable will be bound in the dialog.



Namespace Properties	Description
Add namespace	Add a new Namespace. Namespaces allow Variables with the same name to be reused in different namespaces to meet various requirements.
Edit name	Edit the name of the Namespace.
Delete	Delete the selected Namespace.
Namespace	Select the Namespace. This allows Variables to

	be organized and managed under different Namespaces for better structuring and modularity.
Variable Group Properties	**Description**
Add	Add a new Variable. The default Data Type for the Variable can be selected and adjusted from a dropdown menu.
Add group	Add a Group to categorize Variables for easier management.
Delete	Delete the selected Variable or Group.
Variable Properties	**Description**
Name	The name of the Variable.
Data Type	Select the Data Type for the Variable.
Array	Specify whether the Variable is defined as an array.
Access right	Specify the access rights for the Variable, which can be Read/Write, Write, or Read.
Retentive	Specify whether the Variable retains its value after a power loss. If retentive, the Variable cannot be bound to the device.
Initial value	Set the initial value of the Variable at the first startup. For retentive Variables, this value will be the last value before power loss.
Device	Set the device address the Variable is bound to. A PLC device must be created before binding. Only root Variables can be bound, not their individual members. For BOOL and BOOL Array Variables, they can only be bound to BOOL type addresses (e.g., 0x and 1x addresses on MODBUS devices); other Variable types should be bound to 3x and 4x addresses.
Comment	Add or modify the comment for the variable.

5.3.2. Arithmetic Variable

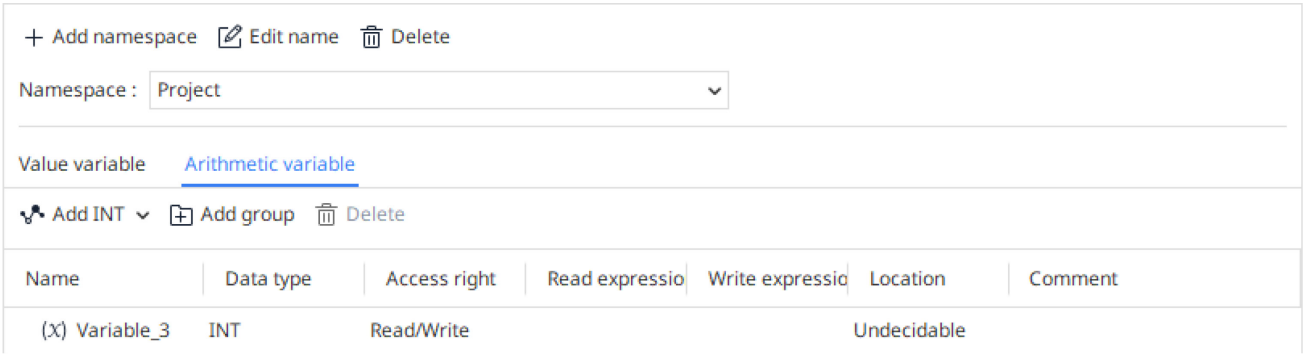
5.3.2.1. Overview

An Arithmetic Variable is a type of variable used to define and perform arithmetic operations. These variables allow users to process and calculate data by defining arithmetic expressions,

making them suitable for applications that require data processing.

5.3.2.2. Steps to Create an Arithmetic Variable

1. Click [Project] » [Variable] to open the Variable Library settings page.
2. Select the Namespace for the new Variable.
3. In the [Arithmetic variable] tab, click [Add] to create a new Variable.
4. The interface will display the variables within the selected Namespace.
5. Double-click to edit the Name, Data type, and other properties.



5.3.2.3. Setting Arithmetic Expressions

1. Double-click the [Read expression] field to define an expression for reading operations.
2. In the editor, choose variables to include in the calculation from "Variant operand". These variables can be other Value Variables or Arithmetic Variables from the system.
3. Use "Arithmetic operator" (e.g., +, -, *, /) to define the logic of the operation. For example, to add two variables, select the first variable, choose the addition operator, and then select the second variable.
4. Once completed, click [OK] to save the arithmetic expression.
5. Double-click the [Write expression] field to define the variable to be written to, as well as the expression for writing operations.
6. In the editor, use "Arithmetic operators" to define the write operation logic, and use “${value}” to represent the value input on the object. For example, to divide the input value by 2, enter “${value} / 2” in the right field.
7. Once finished, click [OK] to save the arithmetic expression.

Variable Properties	Description
Name	The name of the Variable.
Data Type	Select the Data Type for the Variable.
Access right	Specify the access rights for the Variable, which can be Read/Write, Write, or Read.
Read expression	Define the arithmetic expression for reading operations.

Write expression	Define the arithmetic expression for writing operations.
Comment	Add or modify the comment for the variable.
Write value operand	The value input on an object.
Variant operand	Variables that participate in the calculation.
Arithmetic operator	Operators used in expressions: +: Addition, used to sum two values. -: Subtraction, used to subtract the second value from the first. *: Multiplication, used to multiply two values. /: Division, used to divide the first value by the second.

5.3.3. Interaction between Variables and PLC Devices

In EasyBuilder X, screen Widgets do not directly poll PLC addresses. Instead, they subscribe to the relevant Variables. These Variables handle communication and data polling with the PLC addresses.

5.4. Application Case: Temperature Monitoring System

In industrial automation, a temperature monitoring system is a common application scenario. This system needs to monitor temperature data from multiple sensors in real-time and perform corresponding control actions based on temperature changes. To achieve this, various data types need to be defined and used to handle temperature data.

Data Type Classification

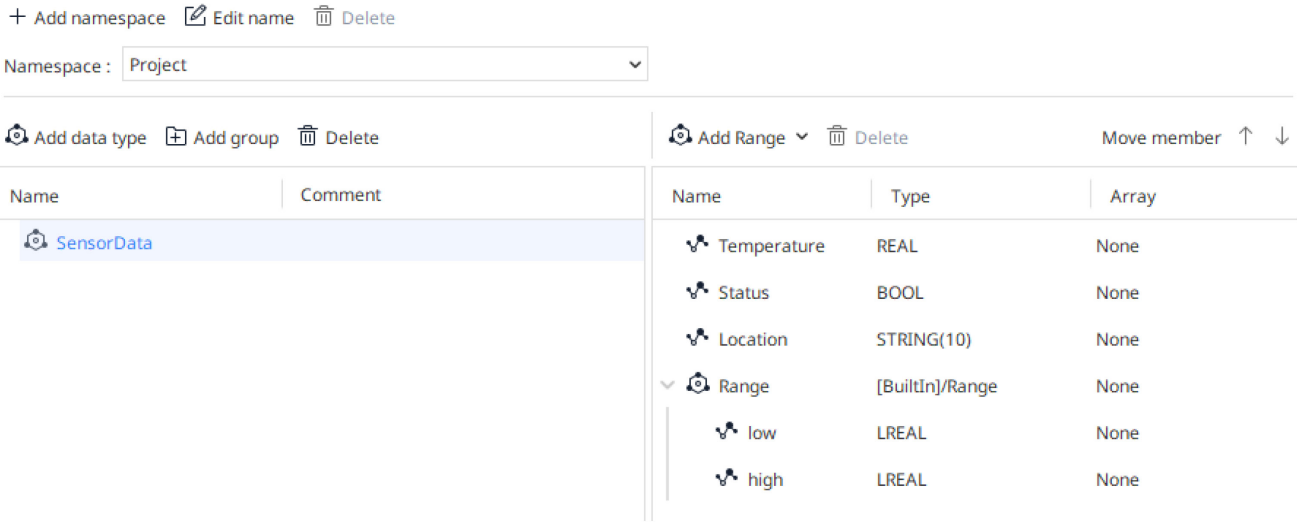
In this case, the following data types are used:

1. Primitive Data Type
   - REAL: Used to store temperature.
   - BOOL: Used to indicate sensor status (normal/fault).
2. Built-in Data Type
   - Range: Used to define the normal and alarm ranges of temperature.
3. User-defined Data Type
   - SensorData: Used to store detailed information for each sensor, including temperature value, status, and location.

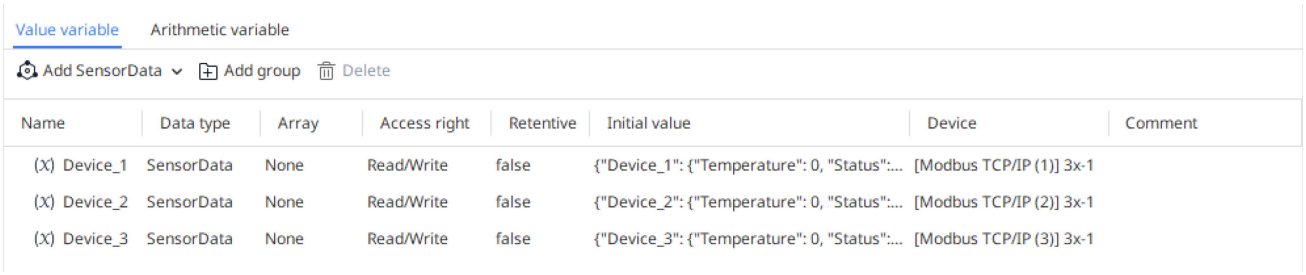
Implementation Steps

1. Add User-defined Data Types.

a. In the Data Type Library, click [Add data type] and name it “SensorData”.
b. Add a member Temperature, of type REAL.
c. Add a member Status, of type BOOL.
d. Add a member Location, of type STRING.
e. Add a member Range, of type built-in data type Range.



2. Configure Sensor Data.
   a. In the Variable Library, create multiple variables of Data Type SensorData, each corresponding to a different sensor.
   b. Set the initial temperature value, status, and location for each sensor.



3. Read Sensor Temperature Data in Real-time.
   a. Perform alarm and control actions based on the temperature range.

6. Widgets

6.1. Overview of Widgets 6-2
6.2. Properties and Property Binding 6-2
6.3. Widget List 6-4
6.4. Property 6-5
6.5. Method 6-17
6.6. Event 6-20

6.1. Overview of Widgets

Widgets provide the primary means for designing HMIs , enabling interface design for a wide range of fields and applications according to specific needs. EasyBuilder X also offers multiple convenient features that significantly enhance the design experience, such as batch widget modification and property binding. These features are explained in the following sections.

6.2. Properties and Property Binding

This section explains the design concepts and classifications of Properties in EasyBuilder X, as well as the data flow and access rules when binding Properties to Variables or to other Properties. A correct understanding of these mechanisms is essential for building stable and maintainable projects. EasyBuilder X adopts an object-oriented design architecture. Each Widget consists of Properties, Methods, and Events. Properties describe a Widget's state and characteristics and can be linked to external data sources through the Property Binding mechanism. Methods define the actions a Widget can perform, while Events define the notifications triggered under specific conditions.

6.2.1. Runtime Characteristics of Properties

Based on whether a property can be modified at runtime, properties are classified into the following two types:

Runtime Unchangeable

These properties must be defined at design time. Their values are read only once when the Widget is created and are used as initialization parameters. They cannot be modified at runtime. They are typically used to define a Widget's basic structure or fixed characteristics, such as widget name, button task, or the number of states in a switch.

Runtime Changeable

These properties can change dynamically at runtime. Default values can be set during design, and values can be updated at runtime through Data Binding or Set-value Action. Based on data flow direction, these properties are further classified as Input, Output, or Input/Output properties.

6.2.2. Property Classification by Data Flow

For runtime-changeable properties, EasyBuilder X defines the following data flow types:

Input Property

The Widget only reads this property and does not write to it. The Widget continuously monitors value changes throughout its lifecycle, and any update is immediately reflected in the Widget's state. For example, when the X coordinate in the Widget's Profile settings is modified, the Widget moves immediately. External sources may write to this property, but the Widget itself acts only as a receiver.

Output Property

The Widget only writes to this property to expose its internal state, such as a switch's current state or a condition's satisfied status. External writing is not allowed. From the user's perspective, this property should be treated as read-only.

Input / Output Property

This type combines both input and output behavior. The Widget monitors the value to update its display and writes back new values when user interaction occurs. A typical example is the Value property of a Numeric Widget.

6.2.3. Property Binding and Variable Access Rights

In EasyBuilder X, variables are categorized by access rights as Read-only, Write-only, or Read-Write. When performing Property Binding, the system automatically applies the following rules based on property type:

- Input Property: the bound variable must have read permission
- Output Property: the bound variable must have write permission
- Input / Output Property: the bound variable must have both read and write permissions

These rules also apply to Property-to-Property binding.

EasyBuilder X also defines access rights for properties based on their data flow:

- Input and Input/Output properties are Read-Write
- Output properties are Read-only

6.2.4. Data Synchronization Timing in Property Binding

Different property types follow different synchronization behaviors:

Input Property

Synchronization direction: Property ← Binding Target

Synchronization timing:

- During widget initialization
- When the binding target value changes

Output Property

Synchronization direction: Property → Binding Target

Synchronization timing:

- When the Widget writes to the property for the first time
- When the property value changes

The timing of writes depends on Widget behavior, for example:

- Numeric: after user input is confirmed
- Condition: when the Widget is created or when the expression is re-evaluated
- Switch: when evaluated based on Value or Read Value

Input / Output Property

Supports bidirectional synchronization:

- Read from binding target: during initialization and when external values change
- Write to binding target: when the Widget writes the value for the first time and when the property value changes

6.3. Widget List

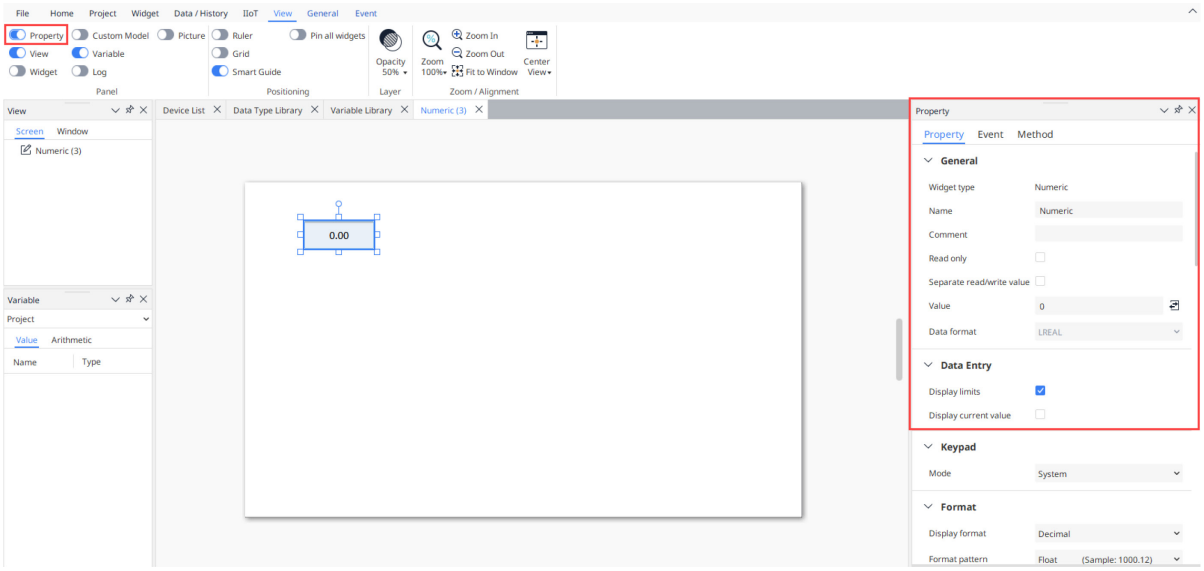
Icon	Name	Description
	**Line**	Draw a straight line.
	**Arbitrary Line**	Draw an arbitrary line.
	**Link Line**	Draw a series of connected straight lines.
	**Arc**	Draw an arc.
	**Ellipse**	Draw an ellipse or circle.
	**Rectangle**	Draw a rectangle.
	**Polygon**	Draw an irregular polygon.
	**Scale**	Draw tick marks.
	**Text**	Add text to the window, with options to set font, color, size, and various related attributes.

	**Picture**	Add an image to the window.
	**Lamp**	Display image or text content based on the state of the bound variable.
	**Switch**	Combine the functions of an indicator lamp and a switch. Pressing the Widget changes the state of the bound variable.
	**Button**	The Button not only provides its own Click method but also allows the actions of other Widgets to be triggered.
	**Option List**	Display multiple items in a list format, allowing selection of specific items.
	**Slider**	Display a value on a slider track and allow changing it by dragging the slider handle.
	**Numeric**	Input or display numeric values within the bound variable.
	**Text**	Input or display text and symbols within the bound variable.
	**Date/Time**	Display the date and time on the HMI. The display format and font can be configured.
	**Duration**	Use with the Relative mode of the Backup widget to dynamically adjust time settings.
	**Bar Graph**	Display variable data using a percentage and bar graph format.
	**Embedded Window**	Pop up or embed another window on the screen.
	**2D Barcode**	Display a 2D code, such as a QR Code, by converting text, URLs, or variable data into a scannable image.
	**PDF Reader**	Display PDF files from external devices on the HMI.
	**Backup**	Back up multiple types of HMI history records.

6.4. Property

Each Widget has its own properties, and each property parameter can be freely set. In simple

terms, properties determine the look and data settings of a Widget. Before delving into the properties and parameters of Widgets, it's essential to understand the location of the property fields and the common icons used for property settings. In the [Panel] menu under [View] tab, check the Property box to bring up the Property panel, which is positioned by default on the right side of the window.



Icon	Description
	Read: Bind data from other parameters or variables to this parameter.
	Write: Bind data from this parameter to other parameters or variables.
	Read/Write: Bind this parameter to other parameters or variables.
	Cancel the binding.
	Set parameters for multiple states simultaneously.
	Set parameters for a single state individually.
	Click to open the settings window.

6.4.1. Modify / Common Properties / Batch Modify

When needing to modify properties, directly select a Widget and modify the parameters of each property in the Property field without repetitive confirmation.

Common properties are the attributes shared by multiple Widgets. When multiple Widgets are selected, the Property field displays only the properties shared by those Widgets.

If the selected Widgets are of the same type, editing the common properties applies the changes to all selected Widgets. If the selected Widgets are of different types, fewer common properties will be available, but batch modification is still supported.

The following introduces common properties present in every Widget:

Item	Description
General	
Widget type	Display the Widget type.
Comment	Set the Widget description.
Outline	
Picture	Set the display picture for the Widget.
Picture Style	Set the color of the picture.
Background	
Fill	Set the background fill color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Label	
Font	Set the font.
Font Size	Set the font size.
Color	Set the font color.
Alignment	Set the font alignment.
Padding	Set the distance between the font and Widget boundaries when alignment is not centered.
Scrolling	Enable text to scroll when displayed. Direction: Set the scrolling direction of the text. Speed: Adjust the scrolling speed of the text (1~15 mm per second). Repeated: If this option is not selected, the text will restart scrolling only after it has finished. If this option is selected, the text will scroll continuously.
Availability	
Visible	Set whether the Widget is visible.
Enabled	Set whether the Widget allows control.
Safety Control	
Control mode	**Press and hold** Execute when the Widget is held for the set minimum time until the following icon appears.

	 **Confirmation** Before execution, display the confirmation window. The window's wait time can be set.
Profile	
Pin	Set whether to fix the Widget's position.
X	Set the X-axis coordinate of the Widget's top-left corner.
Y	Set the Y-axis coordinate of the Widget's top-left corner.
Width	Set the Widget's width.
Height	Set the Widget's height.
Lock aspect ratio	When selected, adjusting the width of the object automatically adjusts its height proportionally, and vice versa.
Rotation	Set the Widget's rotation angle.

6.4.2. Introduction to Widget Properties

6.4.2.1. Line / Arbitrary Line / Link Line / Arc / Ellipse / Rectangle / Polygon / Scale

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Type	Select the type—Linear or Circular; the relevant properties for the chosen type appear. Supported only on Scale.
Pattern / Frame	
Color	Set the line color.
Width	Set the line width.
Style	Set the line style.
Radius	Set the rounded-corner radius. Supported only on rectangles.

Arrow	
Type	Set the arrow type. Supported only on Line and Link Line.
Size	Set the arrow size. Supported only on Line and Link Line.
Interior	
Fill	Set the fill style. Supported only on Ellipse, Rectangle, and Polygon.
Tick Mark	
Tick width	Set the tick mark width. Supported only on Scale.
Style	Set the tick mark style. Supported only on Scale.
Tick color	Set the tick mark color. Supported only on Scale.
Tick radius	Set the radius for a circular scale. Supported only on Scale.
Division	Sets the number of scale divisions (main and sub scale ticks) and their lengths. Supported only on Scale.

6.4.2.2. Text / Text Input

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Text	Text: Set the displayed text content. Format: Set the format by choosing between Single-line or Multi-line display. Password mode: Set whether to hide the text content. When this option is selected, the text will be replaced by * symbols.
Keypad	
Mode	None: No keypad. System: Use system keypad.

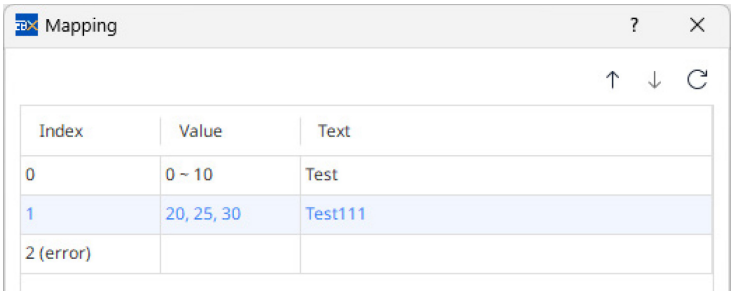
6.4.2.3. Picture

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Outline	
Picture	Select the image to be displayed.
Horizontal flip	Set whether to flip the image horizontally.
Vertical flip	Set whether to flip the image vertically.

6.4.2.4. Lamp / Switch / Button

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Read only	Sets whether the Widget is read-only.
Value	**When Read Only is enabled:** Read value: Set the value to read; can bind to other parameters or variables. **When Read Only is disabled:** Separate read/write value: Set whether read and write values are specified separately.
Current state	Display the current status of the widget.
Data format	Display the Widget's data type.
State mapping policy	**Simple** The range for **No. of states** is 1-256. **Custom** Allow mapping custom read values to different states.
Task	Specify an extra task to execute when the Widget is activated.

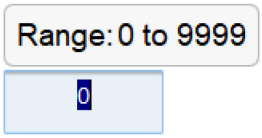
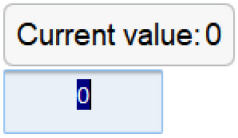
6.4.2.5. Option List

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Mode	Choose between Drop-down List and List Box. When using the Drop-down List, the following settings can be configured: Style: Set the style of the Drop-down List, with options for Classic and Standard.  Font Size Mode: When set to Fixed Size, menu items are displayed at a constant size. Otherwise, items are displayed according to the Font Size setting in Label.
Row height	Set the height of the rows in the Option List.
Value	Set the data or corresponding variables for the Widget.
Source Data	
Source of item data	**Predefine** Predefined item no.: Set the number of items. Mapping: Define the correspondence between the item's Index and Value, as well as the content to be displayed. Values can be set either as a range (e.g. 0~10) or individually (e.g. 20, 25, 30).  **Dynamic** Dynamic data sources can be configured using the Option List method (Update content).
Outline	
Background color	Set the background color of the Option List.
Selection color	Set the color for items in the Option List when they are selected.

6.4.2.6. Slider

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Step	Set the minimum increment for moving the slider.
Value	Set the read value; can be bound to other parameters or variables.
Value limits	Set the slider's minimum and maximum limits.
Quick move policy	Set how the slider moves when the value changes: **Move to target** Moves directly to the target value. **Move toward target coarsely** Moves stepwise toward the target based on the coarse step setting.
Data Format	Display the slider's data type.
Outline	
Style	Select the slider's style.
Direction	Set the direction the handle moves.
Block style	Select the handle's style.
Block width	Set the handle's width.
Block color	Set the handle's color.
Frame color	Set the slider's frame color.
Background color	Set the slider's s background color.
Slot color	Sets the slot's color.

6.4.2.7. Numeric

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Read only	Set whether the Widget is read-only.
Value	**When Read Only is enabled:** Read value: Set the value to read; can bind to other parameters or variables. **When Read Only is disabled:** Separate read/write value: Set whether read and write values are specified separately.
Data format	Display the Widget's data type.
Data Entry	
Display limits	Set whether to display upper and lower limits. 
Display current value	Set whether to display the current value. 
Keypad	
Mode	None: No keypad. System: Use system keypad.
Format	
Display format	Set the display format. Only decimal is supported.
Format Pattern	Set the display pattern, with default options including Integer, Float, and Scientific. Alternatively, use Customize to define a custom display pattern. Supported tokens: ';', '0', '#', '?', '.', ',', 'E+', 'E-', 'e+', 'e-', "text". For details on custom number formats, refer to the number format patterns guide.

Limits	
Device limits	Set the upper and lower limits allowed by the device.
Alarm color	When the value is below the lower limit or above the upper limit, different warning colors and blinking effects are used.

6.4.2.8. Date/Time

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Display	
Mode	**Clock time** Display the date/time content in clock time format. **Value** Use with the Interval mode of the Backup widget to dynamically adjust the start time and end time of the backup interval.
Popup editor on click	If enabled, tapping this widget on the HMI opens an editor for modifying the value.
Editor title	Set the title of the editor.
Format	
Format pattern	Set the display format of the date/time.

6.4.2.9. Duration

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Display	
Read only	Set whether the Widget is read-only.
Value	**When Read Only is enabled:** Read value: Set the value to read; can bind to other parameters or variables. **When Read Only is disabled:**

	Minimum / Maximum value: Set the minimum value and maximum value separately.
Popup editor on click	If enabled, tapping this Widget on the HMI opens an editor for modifying the value.
Editor title	Set the title of the editor.
Visible fields	Set the content displayed on the Widget.
Editable fields	Set the content that can be edited on the Widget.

6.4.2.10. Bar Graph

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Read value	Set the data or variable that the Bar Graph Widget reads.
Value limits	Set the minimum and maximum values displayed by the Bar Graph Widget.
Data Format	Display the widget's data type.
Outline	
Type	**Straight** Style: Set the style of the Bar Graph, with options for Default, Crystal, and Flat. Origin position: Set the starting point's position, with options for Min. Value and Custom, allowing the origin to be set at any position. Direction: Set the display direction of the Bar Graph. Bar width ratio (%): Set the ratio of bar to widget width. **Circular** Angle setting: Select "Predefined" to choose from predefined templates. For custom angles, select "Custom" and set the starting angle, ending angle, and rotation direction as needed. Hole ratio(%): Set the hollow ratio of the Circular Bar.
Bar fill	Set the base color of the Bar.
Target value enabled	Target value: Set the target value.

	Target tolerance: The target value is considered achieved if within a specified tolerance range. Target color: Set the color of the Bar when the target value is reached.
High limit enabled	High limit: Set the upper limit value. High limit color: Set the color of the Bar when the value exceeds the upper limit.
Low limit enabled	Low limit: Set the lower limit value. Low limit color: Set the color of the Bar when the value falls below the lower limit.
Bar background color	Set the background color of the Circular Bar.
Bar frame color	Set the frame color of the Bar.
Picture	Set the image for the Bar.
Horizontal flip	Set whether to flip the image horizontally.
Vertical flip	Set whether to flip the image vertically.

6.4.2.11. Embedded Window

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Read value	Set the data or variable the Embedded Window widget reads.
Data format	Display the widget's data type.
Window mapping policy	**Direct** Select a window which will be hidden when the value is 0 and visible otherwise. **Custom** Define rules to display different windows according to the read value.
Outline	
Mode	**Embedded in screen / window** Set the window to embedded mode. **Popup window** Set the window to pop-up mode. Enable title: Toggle the title bar on or off.

	Title: Set the text displayed in the title bar. Title font: Set the font of the title text.
Display anchor	Set the display anchor of the window relative to the widget.
Entrance animation	Set the window's entry animation.
Exit animation	Set the window's exit animation.

6.4.2.12. 2D Barcode

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Read value	Set the data or variable to be encoded in the 2D barcode.
Barcode type	Select the barcode type.
QR error correction level	Available when QR Code is selected; set the error correction level.
Aztek error correction level	Available when Aztec Code is selected; set the error correction level (range: 5–95).
Cell color	Set the color of the barcode cells.

6.4.2.13. PDF Reader

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
File path	Specify the path of the PDF file on the external storage device.
Storage location	Select the storage location where the PDF file is stored (e.g., USB1, USB2, SD Card).
Current page	Display or set the page number currently being viewed.
Total page	Display the total number of pages in the PDF file.
Zoom factor	Set the display percentage of the PDF content on the screen.

6.4.2.14. Backup

Item	Description
General	
Widget type	Display the Widget type.
Name	Set the Widget name.
Comment	Set the Widget description.
Source type	Set the source of the backup file. **Alarm** Alarm class filter: Select the alarm classes to be backed up. Alarm tag filter: Select the alarm tags to be backed up. **Data Sampling** Data sampling: Select the data logs to be backed up. **Audit Trail**
Range	
Mode	Select Relative or Interval for the backup time range.
Start anchor	Set the start time alignment when the time range mode is Relative. **None:** Do not align the time. **Top of minute:** Align the start time to the minute of the backup trigger time. **Top of hour:** Align the start time to the hour of the backup trigger time. **Top of day:** Align the start time to the date of the backup trigger time.
Start offset	Set the offset value of the start time. Click the clock icon to expand the quick setting interface.
Duration	Set the duration of the backup time range.
Start time	Set the start time when the time range mode is Interval.
End time	Set the end time when the time range mode is Interval.
Output	
Storage location	Set the storage device and folder for the backup files.
File name	Set the name of the backup file. **File name rules** ${sourceObject.logName}: Use the name of the data

	source. {Now:time format}: Use the time when the backup is executed. {Start:time format}: Use the start time of this backup range. {End:time format}: Use the end time of this backup range. For time format rules, refer to the [Date / Time format] setting below in the same setting group.
Max. records per file	Set the maximum number of records per file. Set to 0 to store all records in a single file. The configurable range is 500 to 5,000,000 records. When the number of records in a single file exceeds the limit, the data is automatically split into multiple files.
File format	Set the file format. Currently, only CSV is supported.
Encoding	Set the file encoding. Currently, only UTF-8 is supported.
UTF-8 BOM	Set whether to use UTF-8 BOM format.
Date / Time format	Set the display format of the record time.
Runtime	
Status	Display the backup status information. 0: Idle 1: Backing Up 2: Failed
Progress	Display the backup progress, shown as 0–100 (%).

6.5. Method

Widgets have methods representing their capabilities. EasyBuilder X converts these capabilities into methods for use by other functions, such as clicking once, changing state, getting input focus, etc. Some methods require additional parameters to determine their behavior.

6.5.1. Introduction to Widget Methods

6.5.1.1. Text Input / Numeric

Item	Description
Acquire input focus	Acquire input focus.
Release input focus	Release input focus.

6.5.1.2. Button

Item	Description
Execute task	Execute the task specified in the Task property.

6.5.1.3. Lamp (Controllable) / Switch

Item	Description
Execute task	Execute the task specified in the Task property.
Set state	Set the Widget to a specific state. Parameter: Specific state (INT).
Increase state	Increase the Widget's state by one. Parameter: Cyclical (BOOL). When selected, the state cycles from the minimum value after reaching the maximum value; otherwise, it stops at the maximum value.
Decrease state	Decrease the Widget's state by one. Parameter: Cyclical (BOOL). When selected, the state cycles from the maximum value after reaching the minimum value; otherwise, it stops at the minimum value.

6.5.1.4. Option List

Item	Description
Update content	When the source of the Option List is set to dynamic, this method must be used to configure the display content. Parameter: Number of options to update (INT) Parameter: Display text of options (String Array)

6.5.1.5. PDF Reader

Item	Description
Go to previous page	Navigate to the previous page.
Go to next page	Navigate to the next page.

6.5.1.6. Backup

Item	Description
Execute	Execute the backup and wait for completion.

Execute (Async)	Execute the backup without waiting for completion.

6.5.1.7. Date/Time & Duration

Item	Description
Popup editor	Open a pop-up editor to modify the value.

6.6. Event

Widget events occur when a Widget's state changes and can trigger predefined actions, such as updating values, switching pages, or invoking methods of other Widgets.

6.6.1. Introduction to Widget Events

6.6.1.1. Text / Text Input

Item	Description
Editing started	Triggered when the user starts editing the text box.
Editing finished	Triggered when the user finishes editing the text box or when the input focus moves away from it.
Text written	Triggered when the user finishes editing the text box.
Text changed	Triggered when the text box content changes.

6.6.1.2. Lamp (Controllable) / Switch / Button

Item	Description
Pressed	Triggered when the user presses the Widget.
Released	Triggered when the user releases the Widget after pressing.
Clicked	Triggered when the user clicks (presses and releases) the Widget.
State changed (Lamp, Switch)	Triggered when the state changes.
Held (Button only)	Triggered when the user holds the Widget for the time specified in Property. The event begins execution after holding for the Delay Time and triggers repeatedly at the configured interval.

6.6.1.3. Numeric

Item	Description
Editing started	Triggered when the user starts editing the Numeric Widget.
Editing finished	Triggered when the user finishes editing the widget or when the input focus moves away from it.
Value changed	Triggered when the Numeric Widget value changes.
Value written	Triggered when the user finishes editing the Numeric Widget

6.6.1.4. Option List

Item	Description
Selection changed	Triggered when the selected item in the Option List changes.
Value changed	Triggered when the value in the Option List changes.
Error notification	Triggered when an error occurs.

6.6.1.5. Slider

Item	Description
Slider pressed	Triggered when the Slider is pressed down.
Slider released	Triggered when the Slider is released.
Slider moved	Triggered when the Slider handle moves.

Value changed	Triggered when the Slider value changes.

6.6.1.6. Bar Graph

Item	Description
Normal left	Triggered when the Bar Graph exits the normal state.
Normal entered	Triggered when the Bar Graph enters the normal state.
Target left	Triggered when the Bar Graph exits the target state.
Target entered	Triggered when the Bar Graph enters the target state.
Low limit left	Triggered when the Bar Graph exits the low limit state.
Low limit entered	Triggered when the Bar Graph enters the low limit state.
High limit left	Triggered when the Bar Graph exits the high limit state.
High limit entered	Triggered when the Bar Graph enters the high limit state.

6.6.1.7. PDF Reader

Item	Description
File opened	Triggered when the PDF file is opened.
File closed	Triggered when the PDF file is closed.
Current page changed	Triggered when the current page of the PDF changes.

6.6.1.8. Backup

Item	Description
Status changed	Triggered when the backup status changes.
Completed	Triggered when the backup is completed.
Failed	Triggered when the backup fails.

6.6.1.9. Date/Time & Duration

Item	Description
Value changed	Triggered when the value changes.

6.6.1.10. Event (Action)

After an event is triggered for a Widget, the following actions can be added as subsequent operations.

Operation	Description
Set value	Set the properties of a Widget or the value of a variable. For example, directly adjust the variable's value to the

	maximum or change the position of the Widget. **=:** Assign “Value” to “Target”. **+=:** Add “Value” to “Target” and store the result back in “Target”. **-=:** Subtract “Value” from “Target” and store the result back in “Target”.
Call object method	Call methods of other Widgets, such as setting the Widget’s state or directly acquiring input focus.
Change screen	Switch the displayed screen.
Popup window	Display a popup window.
Close window	Close the popup window.
Send keyboard input	Send the specified keyboard command.
Import / Export string table	Import and export string tables.
Add if condition	Add an if-else statement to set the actions based on logic.
Add while loop	Add a while statement, which executes the actions within the loop only if the condition is met.
Wait	Wait for the specified time before checking if a variable or parameter meets its condition.
Delay	Delay for the specified time.

7. Custom Model

7.1. Functions and Uses of Custom Models 7-2
7.2. Enabling the Interface and Panel Functions 7-2
7.3. Object-Oriented Structure (Property, Method, Event) 7-2
7.4. Creating a Custom Model 7-2
7.5. Custom Model Editing Interface 7-3
7.6. Function Settings and Examples 7-4
7.7. Deployment, Use, and Maintenance 7-13

7.1. Functions and Uses of Custom Models

In EasyBuilder X, Custom Models are a core part of object-oriented design. Their main purpose is to package multiple basic Widgets, such as Numerics, Pictures, and Buttons, into a new Custom Widget with a specific function. Once defined, the Custom Widget can be reused unlimited times throughout the project. If the model is modified later, all instances in the project are updated automatically. This chapter explains the structure of Custom Models, how to configure them, and how to use them in a project.

7.2. Enabling the Interface and Panel Functions

Before creating Custom Models, the corresponding management panel must be enabled.

1. On the top toolbar, open the [View] tab.
2. In the [Panel] section, select [Custom Model].
3. Once enabled, the [Custom Model] panel appears in the workspace.

7.3. Object-Oriented Structure (Property, Method, Event)

Custom Models follow the object-oriented principle of encapsulation, separating internal implementation details from the external interface. Understanding the difference between public and private members is the foundation for designing stable and maintainable widgets.

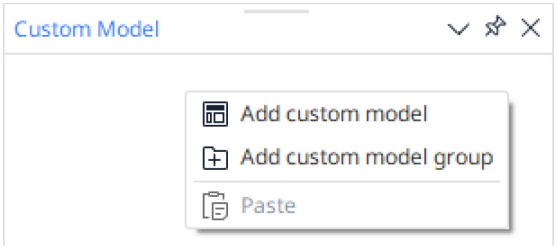
- **Private:** Private properties can only be bound to or accessed by widgets inside the model. By default, all properties, methods, and events of widgets within a Custom Model are private and cannot be directly bound to or accessed from outside the model.

- **Public:** Public properties are properties, methods, or events that can be accessed directly from outside the Custom Widget. These interfaces are defined and exposed by the user, and are typically used to provide the widget's basic characteristics for external operation, control, or binding.

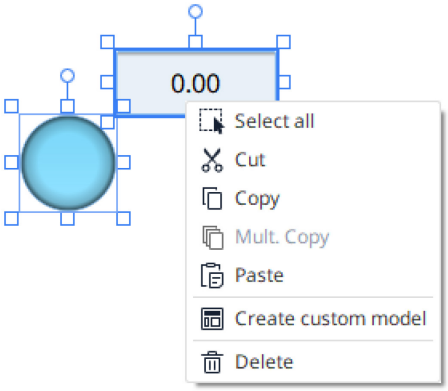
7.4. Creating a Custom Model

A Custom Model can be created in either of the following ways:

- Method 1: In the [Custom Model] panel, right-click and select [Add custom model].

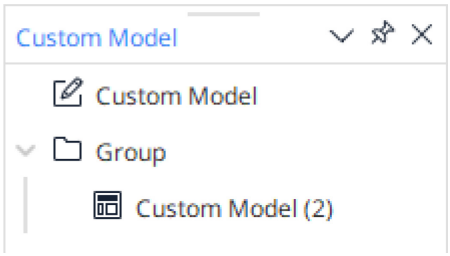


- Method 2: On the screen, select multiple Widgets to combine, right-click, and select [Create custom model].



7.5. Custom Model Editing Interface

In the [Custom Model] panel, right-click to open the following options:
Add custom model: Add a custom model.
Add custom model group: Add a custom model group.
This panel lists all Custom Models in the EasyBuilder X project. It can be used to add, delete, and group Custom Models for project management.



Status	Description
	Editing Mode The model is open in the editing area and its internal layout and logic can be modified in real time.
	Non-Editing Mode Double-click the model to enter editing mode.

7.6. Function Settings and Examples

7.6.1. Custom Properties

Custom properties are variables used to define the state of a model. By configuring properties, you can determine which characteristics of the Custom Model are exposed externally.
In the Properties panel, click [Add property] and select an appropriate data type.

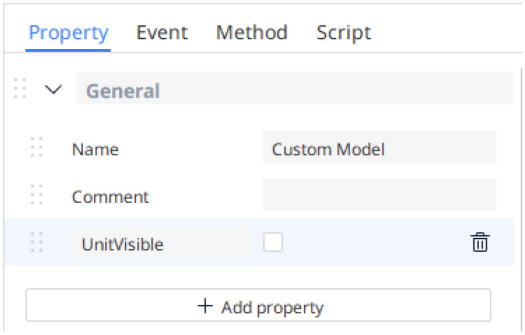
Data Type	Description
Basic Type	Includes BOOL, BYTE, WORD, DWORD, LWORD, SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, STRING, and WSTRING.
Date / Time	Includes data types related to time (TIME/LTIME) and date.
Structured data type	Reference a data type defined in the project's Data Type Library. This type of property contains multiple data fields, each of which can use a different data type and has a defined structure and name. It is suitable for complex data structures composed of multiple variables, such as all operating parameters of a motor.
Export internal property	Set a widget property inside the Custom Model as public. This directly exposes a specific internal widget property, such as position or color, for external access.

Example of Custom Property Configuration: Global Linked Control

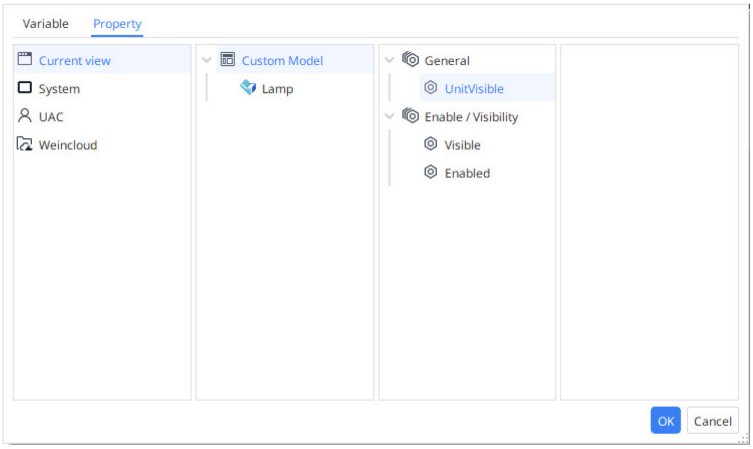
Scenario: The custom model is a "motor station" that contains multiple visual Widgets. When the system is not running, an external address can be used to show or hide all Widgets in the motor station at once.

Steps:

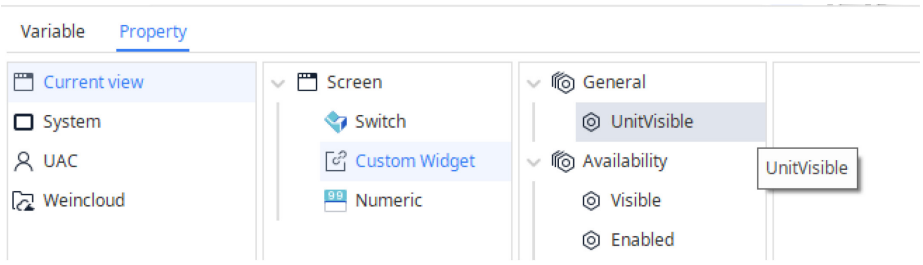
1. In the Property panel of the Custom Model, click [Add property].
2. Select Bool as the data type, and name the property UnitVisible.



3. Select all widgets inside the model. Next to the Visible property in the Property panel, click the button and bind it to the public property UnitVisible.



4. Click [Apply changes] to confirm. After the Custom Widget is deployed, writing to UnitVisible externally controls the visibility of all widgets inside the model at the same time.

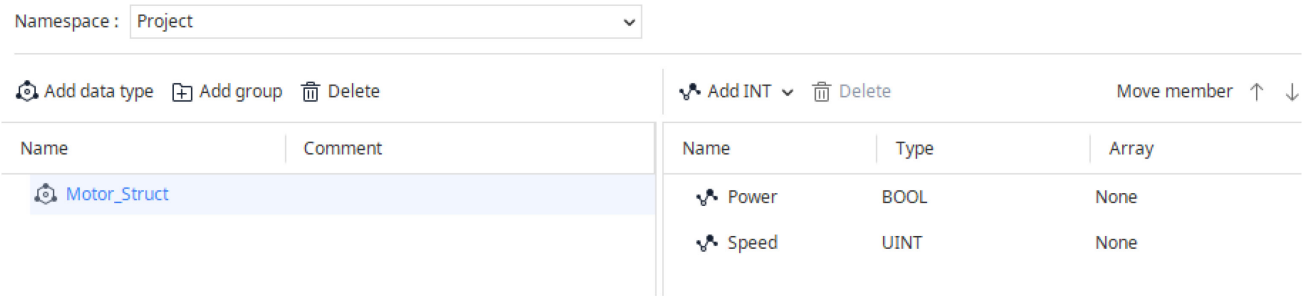


Example of Custom Property Settings: Structured Data Type Property Control

Scenario: The Custom Model is a monitoring station for managing motor parameters at different stations. By entering a motor number through an external Numeric widget, the enabled status and speed information of the corresponding motor can be switched and displayed in real time.

Steps:

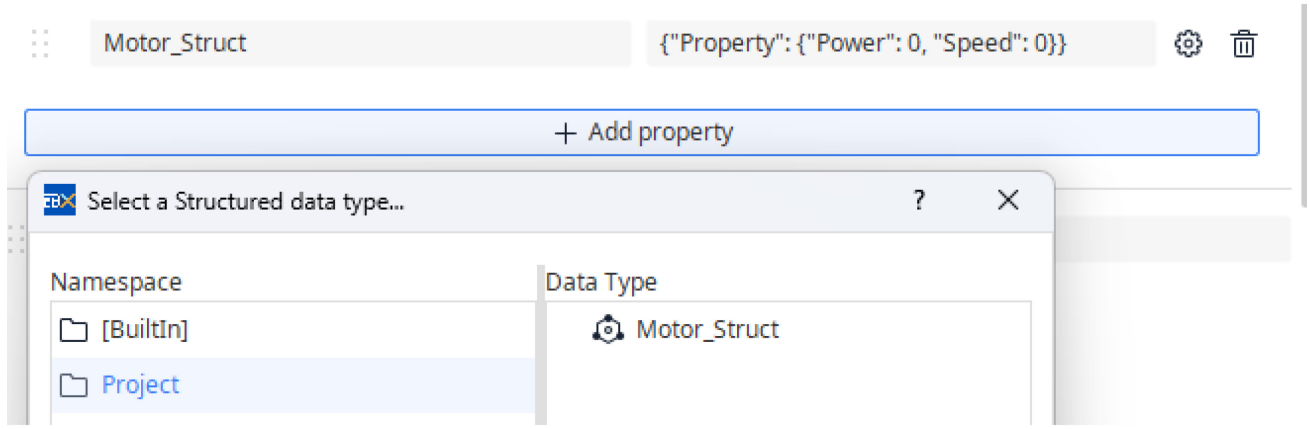
1. Define the motor enabled status and speed in the Data Type Library.



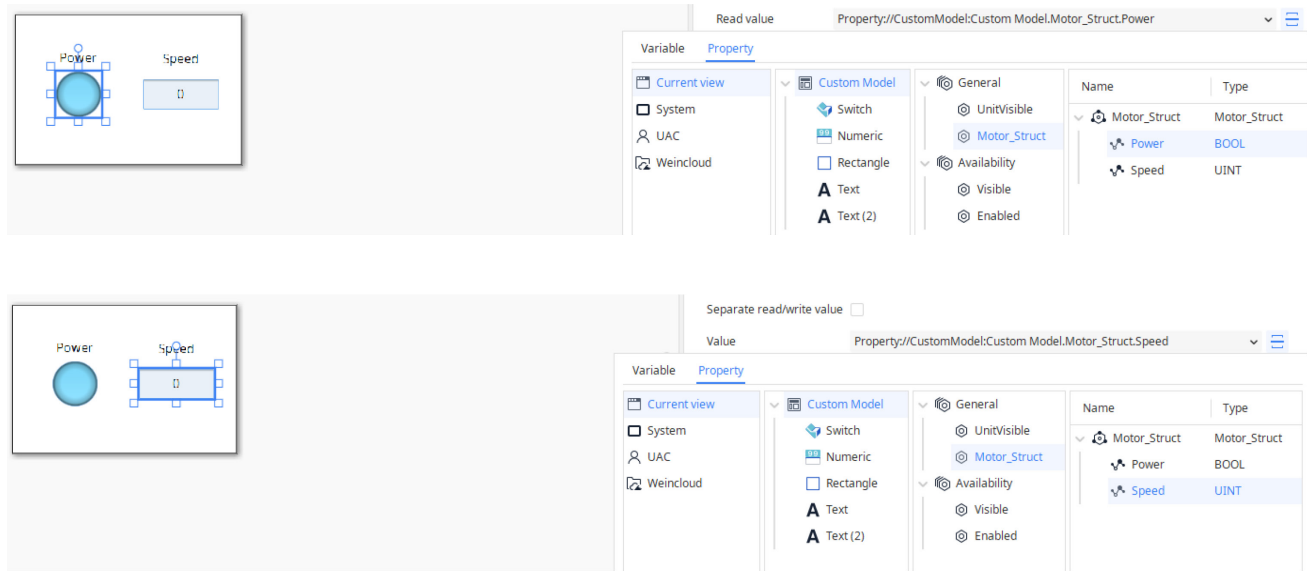
2. In the Variable Library, create an array structure for motors 0–2 and a variable for the motor number.

Name	Data type	Array	Access right	Retentive	Initial value
Motor_Array	Motor_Struct	[0..2]	Read/Write	false	DataType : Motor_Struct (array)
Motor_No	USINT	None	Read/Write	false	0

3. Add a structured data type property to the Custom Model, and select the created Motor_Struct.



4. In the Custom Model, bind each widget to the Motor_Struct property.

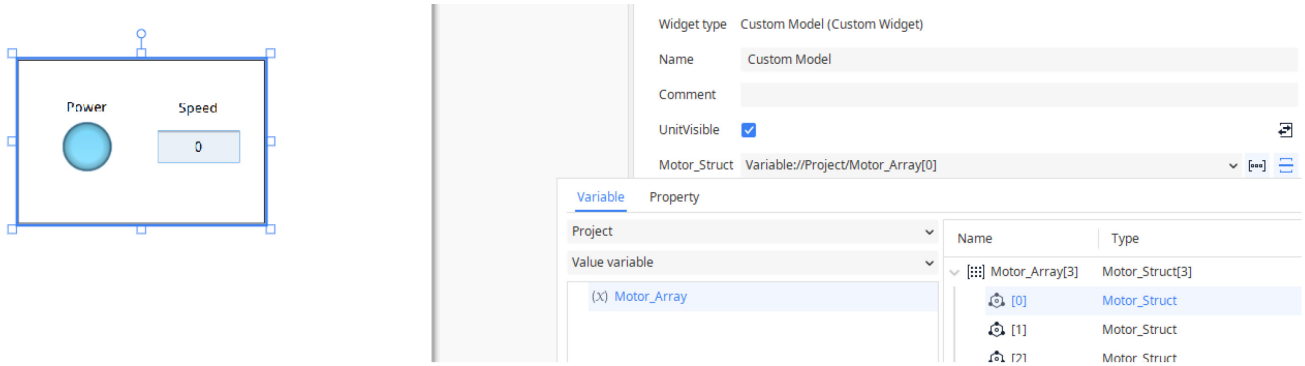


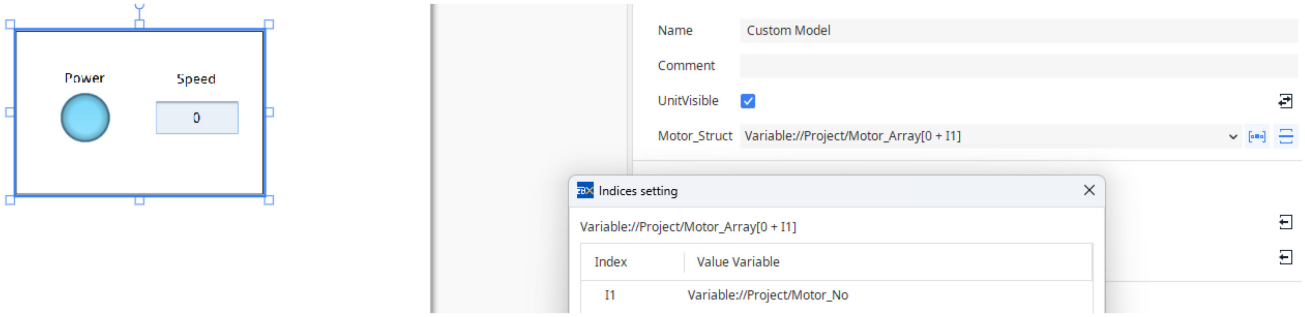
5. Add the Custom Model and a Numeric widget to the screen, then bind the Numeric

widget to the Motor_No variable.

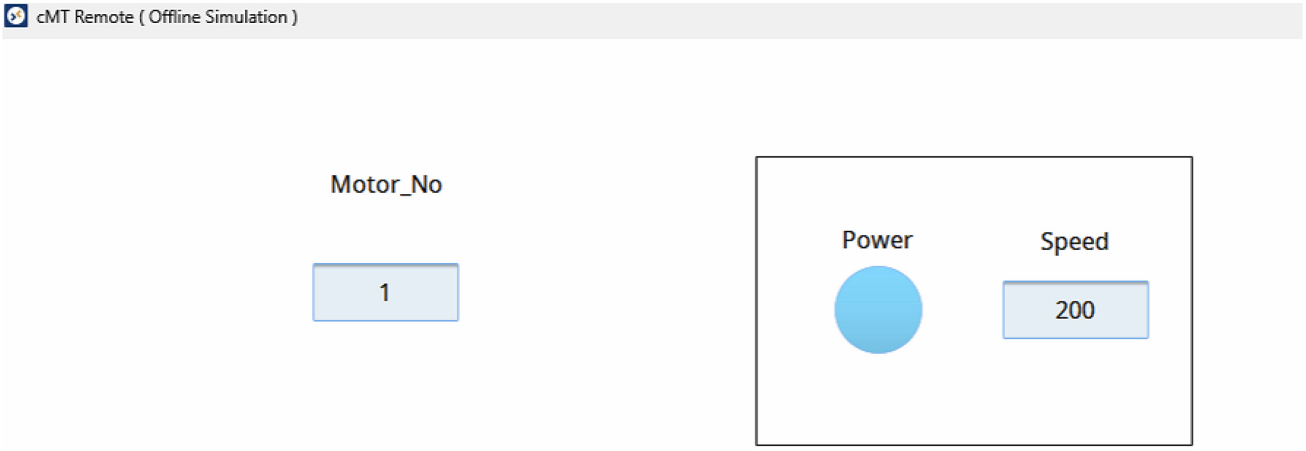


6. Bind the Custom Widget property to the Motor_Array variable, and bind the index to the Motor_No variable.





7. During simulation, when the value of Motor_No changes, the screen displays the status of the motor with that number.



7.6.2. Custom Methods

A method defines the behavior of a Custom Model. It describes an operation that an external widget can perform on the model. Methods allow external Widgets to operate on its internal state (private properties) and provide an interaction interface.

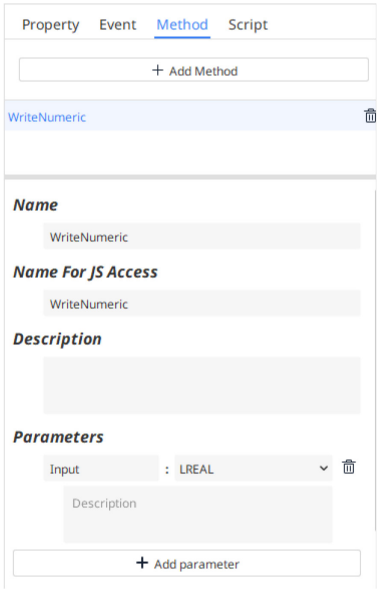
Item	Description
Name	The method name.
Name for JS Access	Set the name used to access this method in JavaScript.
Description	Description of the method.
Parameters	Define the input data required when the method is executed. For example, a “Set Speed” method may require a numeric parameter.
Action	Define the actual behavior to be executed inside the Custom Model when the method is called, such as setting a value or calling a method of an internal widget.

Example of Custom Method Configuration: Writing a Value by Parameter

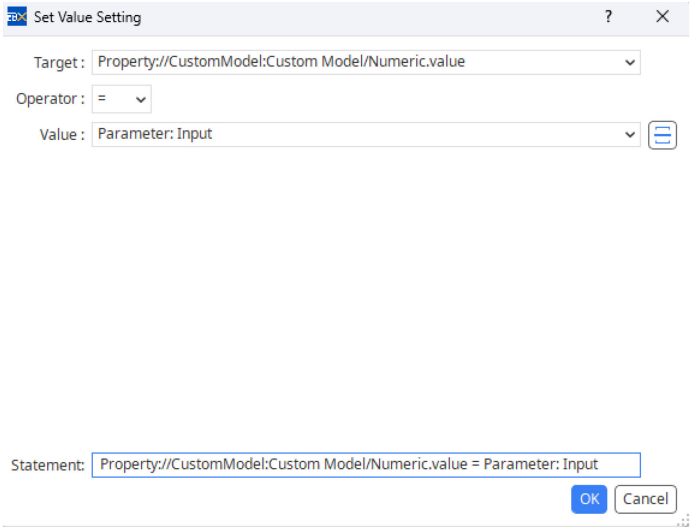
Scenario: The custom model is a dashboard that displays a production target. After a user enters a value in a Numeric Widget on the main screen, a method of the Custom Model can be called externally to pass the value into the model and update the displayed content.

Steps:

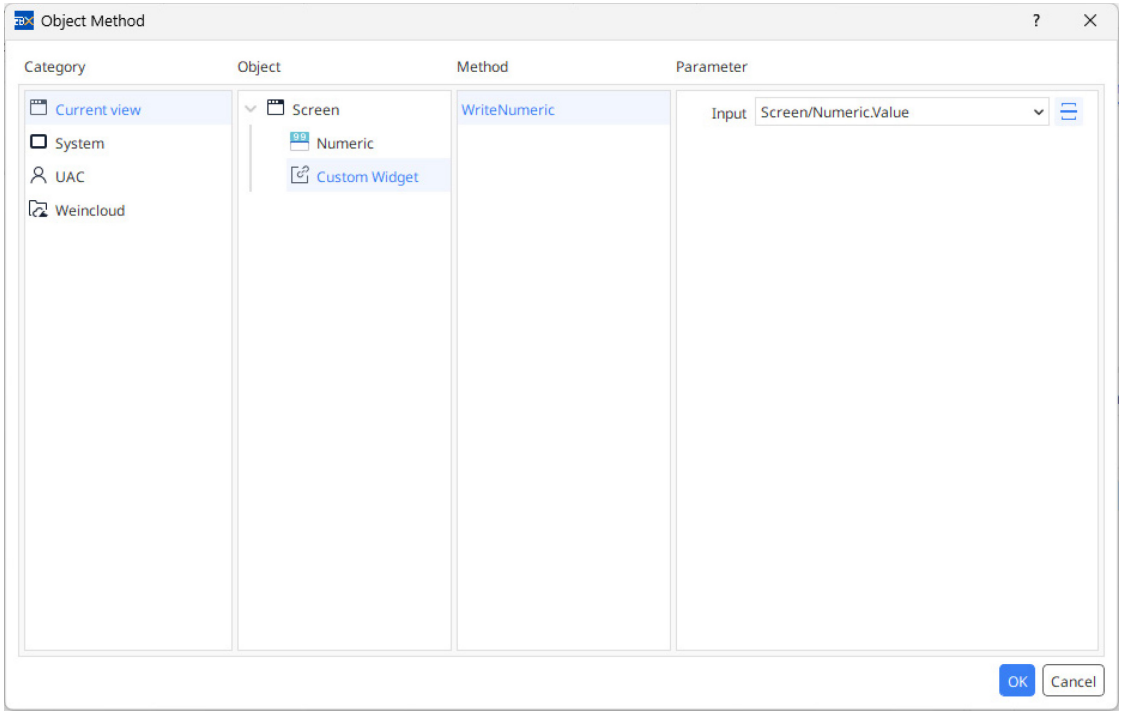
1. Add a Custom Model, and place a Numeric widget in the model.
2. On the model’s [Method] tab, click [Add Method] and name it WriteNumeric.
3. Click [Add parameter] and add a parameter named Input with data type LREAL.



4. In the Action field, add a [Set value] action.
5. In the pop-up dialog box, set the target to the Value property of the internal [Numeric] widget. Set the operation to =. For the value, click the link button on the right and select the method parameter Input.



6. Click [Apply Changes] to confirm. The Custom Model setup is complete.
7. On the main screen, place one Custom Widget and one Numeric Widget.
8. Select the event of the Numeric Widget, choose [Value changed], and add the [Call object method] action.
9. In the dialog box, select the WriteNumeric method of the Custom Widget, bind the Input parameter to the Value property of the Numeric Widget, and click OK.



After the setup is complete, when the value of the Numeric Widget changes, the value inside the Custom Widget also changes.

7.6.3. Custom Events

An event is a signaling mechanism. It is triggered when an object performs a specific action or changes state, and is used to notify other parts of the program to execute corresponding actions. These actions are handled by handlers or functions that define what should happen when the event occurs. Events are commonly associated with user operations such as button clicks or keyboard input.

In a Custom Model, the events of internal widgets are private. Through custom event configuration, when an internal widget event is triggered, the corresponding parameters can be passed to the outside.

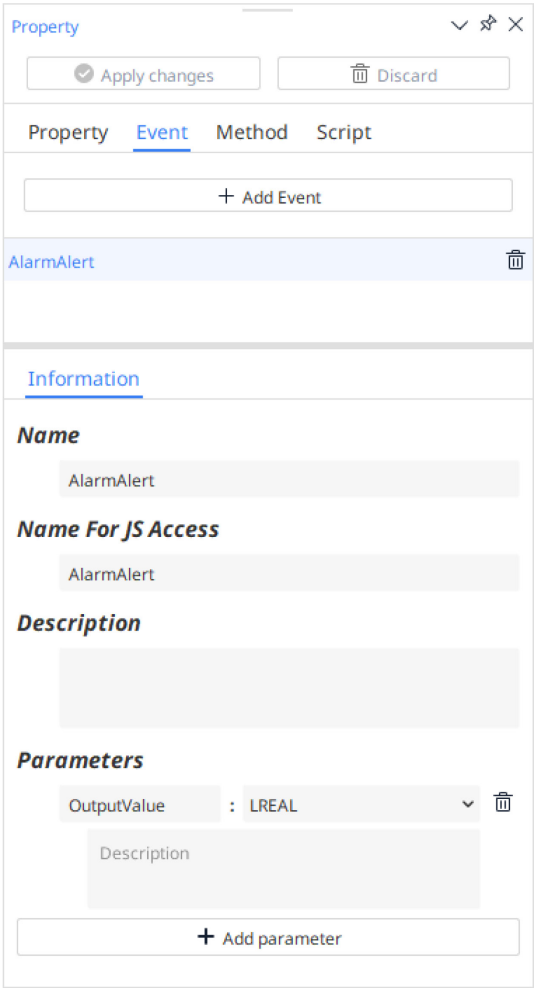
Item	Description
Name	The event name.
Name for JS Access	Set the name used to access this event in JavaScript.
Description	Description of the event.
Parameters	Define the data passed out with the event. For example, when a motor alarm occurs, an “alarm code” can be passed out with the event.

Example of Custom Event Configuration: Passing a Status by Parameter

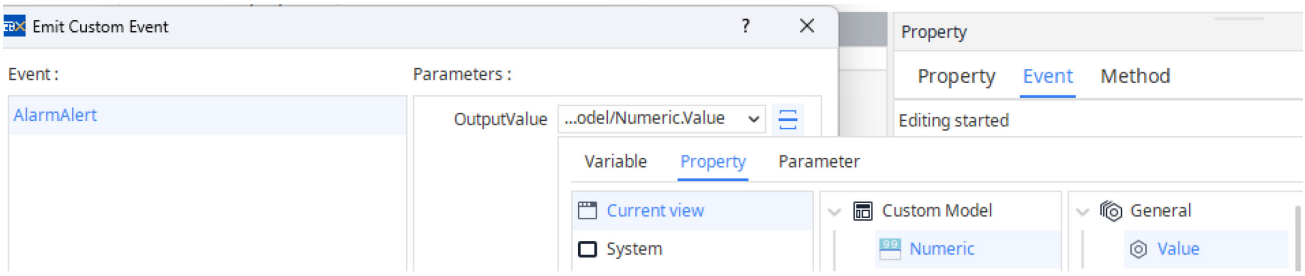
Scenario: When the pressure value inside the Custom Model exceeds the alarm limit, the model must actively notify the external system and pass out the current pressure value at the same time, so that an external Widget on the main screen can display the alarm data.

Steps:

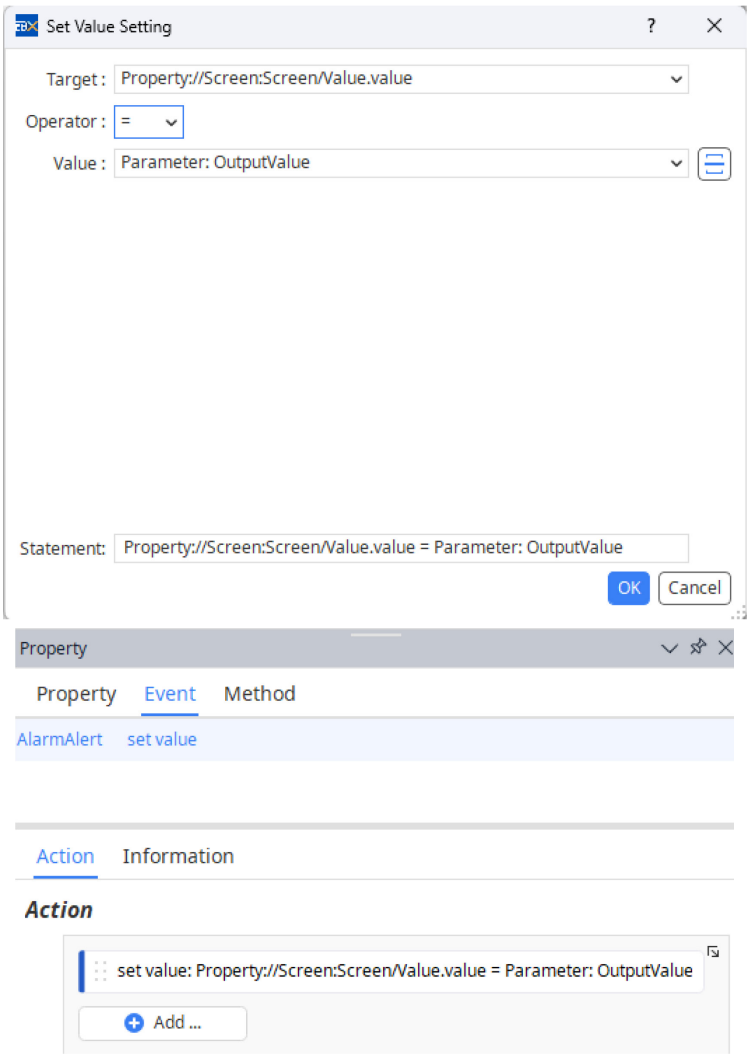
1. On the [Event] tab, click [Add Event] and name it AlarmAlert.
2. Click [Add parameter] and add a parameter named OutputValue with data type LREAL.



3. In the event settings of the Numeric Widget inside the Custom Model, click [Value changed] and add the [Emit custom event] action.
4. Select the defined AlarmAlert event. In the parameter list, click the bind button next to OutputValue and bind it to the Value property of that Numeric Widget.



5. Click [Apply changes] to confirm. The Custom Model setup is complete.
6. Place one Custom Widget and one Numeric Widget on the screen.
7. Select the event of the Custom Widget, choose AlarmAlert, and add the [Set value] action.
8. Set the target to the Value property of the Numeric Widget, bind the value to the OutputValue parameter of the Custom Widget, and click OK.



After the setup is complete, when the value inside the Custom Widget changes, the value of the Numeric widget also changes.

7.7. Deployment, Use, and Maintenance

7.7.1. Deploying a Custom Widget

After the model is defined and [Apply changes] is clicked, it can be used multiple times in the project. There are two deployment methods:

- **Drag-and-drop deployment:** Select the model from the [Custom Model] panel and drag it directly onto the target screen. This method is available only when the model is in Not Editing status in the panel.
- **Widget toolbar deployment:** If the model is currently in Editing status, or if menu-based operation is preferred, open the [Widget] tab on the top toolbar, select the desired model from the [Custom Model] menu, and place it on the screen.

7.7.2. Instance Concept

Each widget deployed on a screen is an instance of that model. Different variables can be bound to different instances. They share the same logic and appearance, but their operating data remains independent.

7.7.3. Centralized Maintenance and Synchronized Updates

Custom models provide an efficient way to maintain a project:

- **Maintenance mode:** To modify the appearance or logic of all Widgets of the same type in the entire project, open the model and enter Editing status.
- **Synchronized update:** After making changes in the model editing area and clicking [Apply changes], all deployed instances in the project are updated automatically. No manual changes are required one by one.

8. String Library

8.1. String Library 8-2
8.2. Usage Instructions 8-3
8.3. Multi-language Settings 8-9
8.4. Font Application Logic 8-11

8.1. String Library

Click [Project] » [String] to open the String Library settings page.

The String Library is primarily used for managing and organizing the display text in different languages. Its main purposes include:

- **Centralized Text Management:** All display text is centralized in one location, making it easier to manage and edit.
- **Multilingual Support:** Supports text in multiple languages, enabling users of diverse language backgrounds to switch as needed, thereby enhancing the product's usability.
- **Consistency Improvement:** Ensures consistency in the strings used throughout the application, thereby enhancing the user experience.
- **Simplified Maintenance:** When text modifications are needed, changes can be made directly in the String Table, eliminating the need to update different controls or pages individually, which saves time and effort.
- **Facilitated Translation and Localization:** The String Table can be provided to translators, simplifying the translation process and facilitating application localization.
- **Dynamic Content and Variable Support:** Dynamically loads text based on user settings or context and supports variable parameters for real-time data display, improving interactivity and display flexibility.

8.1.1. Steps to Create a String

1. Click on [Project] » [String] to open the String Library settings page.
2. Select the String Table where the String is to be created.
3. In the String Library, click [Add string] to add a new String.
4. The interface will display the strings contained within the String Table.
5. Double-click to edit the String ID, English text, and Comment.

8.1.2. String Library Properties

In the String Library, String Tables and Strings can be created. A String Table is a form that stores Strings. When the number of Strings increases significantly as the project design expands, new String Tables can be established based on the usage type of the Strings, allowing users to more quickly find the Strings they need.

String Table Properties	Description
Add table	Add a new String Table.
Edit name	Edit the name of the String Table.

Delete	Delete the selected String Table.
Export	Export the String Table as an Excel file, available in .csv or .xls/.xlsx formats.
Import	Import the String Table from an Excel file, available in .csv or .xls/.xlsx formats.
String Operation Properties	**Description**
Add string	Add a new String.
Edit string	Edit the text of the String.
Delete	Delete the selected String.
Delete unused strings	Remove all Strings that are not used in the project.
String Properties	**Description**
String ID	The ID number of the String. Each String in a String Table is assigned a unique ID, which cannot be duplicated.
In Use	Indicates whether the String is used in the project.
English	The display language of the String.
Comment	Add or modify comments for the String.

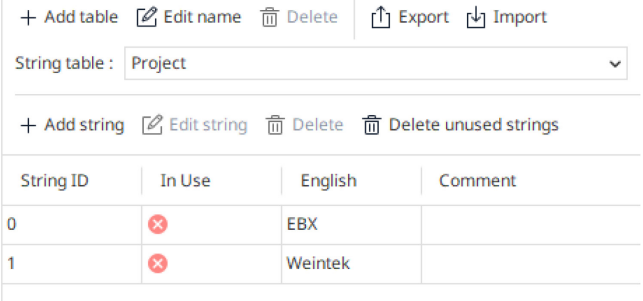
8.2. Usage Instructions

8.2.1. Static Text Display

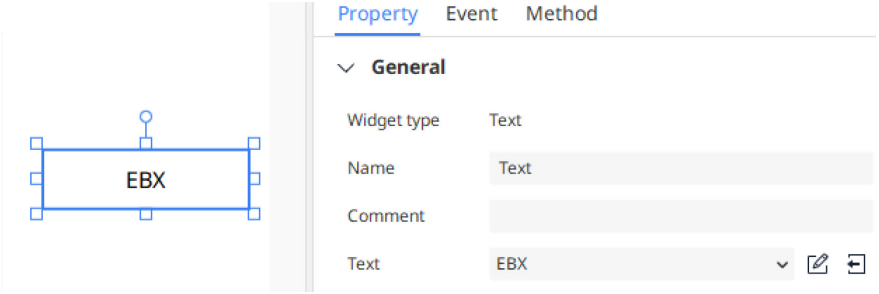
Display fixed String content.

Implementation Steps

1. Create a String.
   a. Click [Add string] in the String Library.
   b. In the “English” column, enter the content of the String to be displayed.



2. Create a Widget and bind the String.
   a. Add a Text Widget on the screen.
   b. Bind the Text property of the Text Widget to the String.



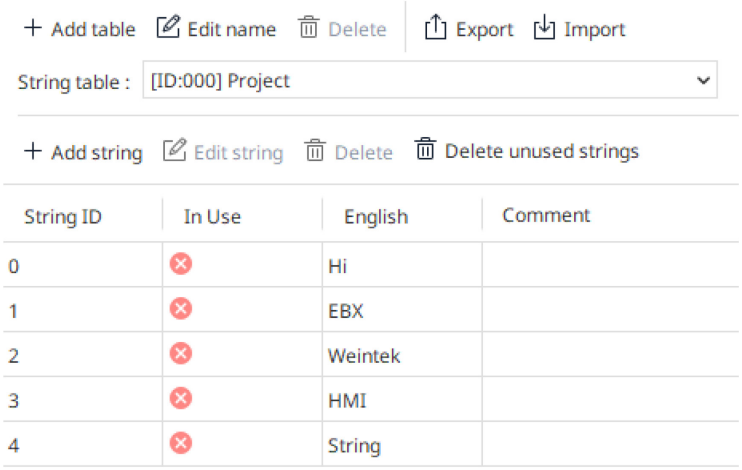
3. After completing the steps, run offline simulation, and the Text Widget will display the bound String content.

8.2.2. Dynamic Text Display

Display different String content based on value status.

Implementation Steps

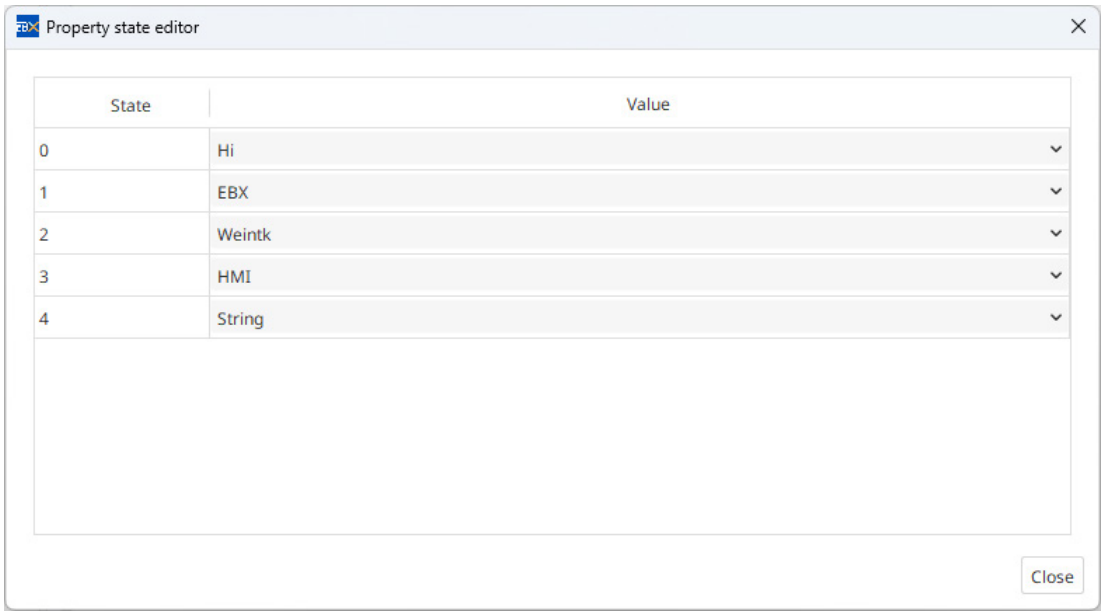
1. Create a String.
   a. Click [Add string] in the String Library.
   b. In the “English” column, enter the content of the String to be displayed.



2. Create a multi-state Widget (e.g., Lamp) and bind the String.
   a. In the Variable Library, create a numeric variable named “Dynamic_text” to serve as the reference for displaying the String.
   b. Create a Lamp Widget and a Numeric Widget on the screen.
   c. Set the [No. of states] property of the Lamp Widget, for example, to 5.
   d. Bind all states of the Lamp's Text property to the corresponding Strings.

e. Bind the Read Value, Current State of the Lamp Widget, and Value of the Numeric Widget to the variable “Dynamic_text”.

f. After completing the steps, run offline simulation, and the text of the Lamp Widget will display the corresponding text based on the value of the Numeric Widget.





8.2.3. Export/Import String Table

Dynamically manage the string table content.

Implementation Steps

1. Create two Button Widgets, then add a String Table task to each.
   a. Configure one for Export String Table and the other for Import String Table.
   b. Specify the target storage device and file name path. Supported formats: .csv, .txt.
   c. After downloading the project to the HMI, export the current String Table, edit its content, and then import the String Table file again to apply the updated entries.

When operating the String Table, view the result code and description.

Parameter	Result Code (UDINT)
Result Codes and Descriptions	0: Operation completed successfully
	5: An unexpected exception occurred during the

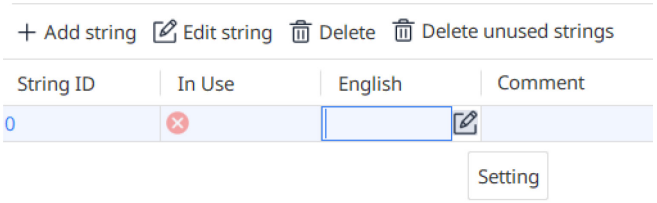
	operation.
	103: the file exceeds the maximum line count.
	2000: The source file was not found.
	2002: The file name is empty.
	2004: Failed to write the file.
	40003: The target file already exists.
	40010: A numeric field contains non-numeric data.
	40028: The CSV does not match the specification.
	40041: The external storage device was not found.
	40044: Failed to open the file.
	40045: Unsupported byte order mark encoding.

8.2.4. Display Strings with Inserted Variables

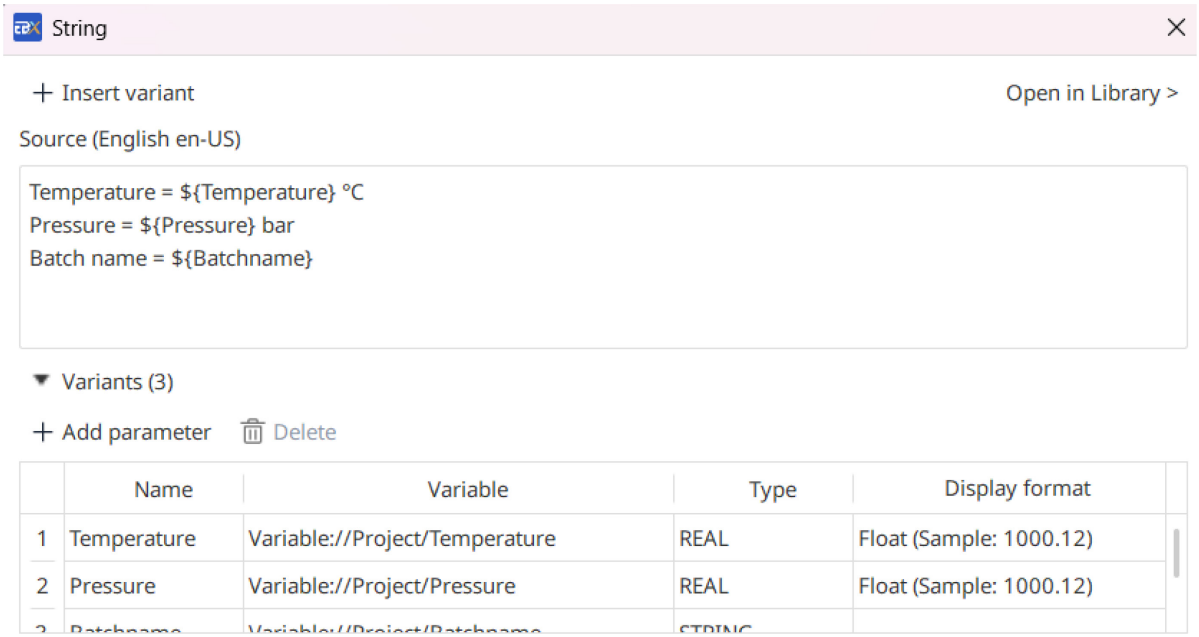
Insert variable parameters into a string so that a single string can dynamically update based on real-time data.

Implementation Steps

1. Create Variables and String
   a. In the Variable Library, create the variables / strings to be displayed (e.g., Temperature, Pressure, and Batchname).
   b. In the String Table, add a new string entry.
   c. In the string content field, click the [Setting] icon to open the editor.

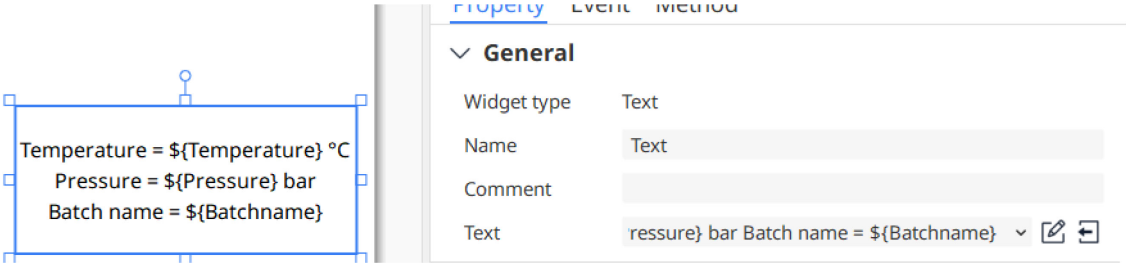


   d. In the string editor, enter text as needed (line breaks can be added if required). Then select [Insert variant] to insert the variables created in step a.

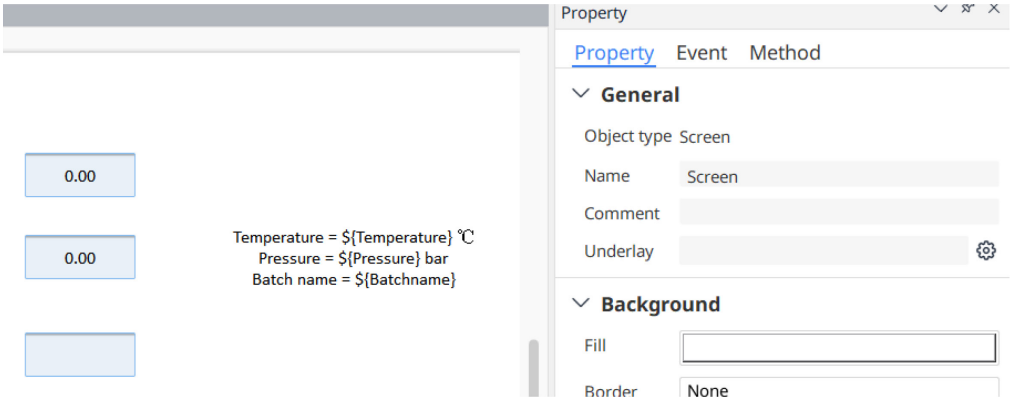


2. Create a Text Widget and Bind the String

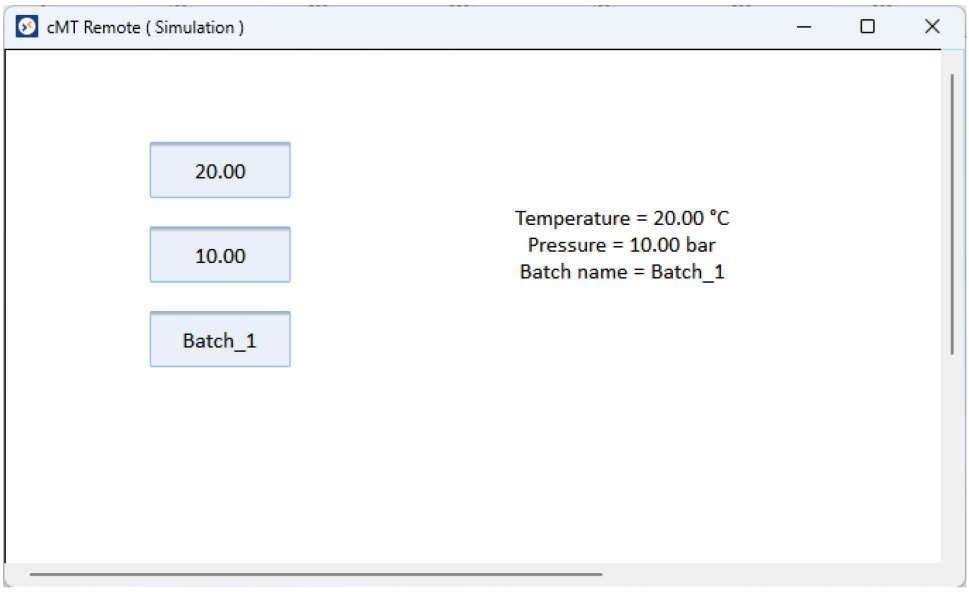
a. Add a Text Widget to the screen and bind its Text property to the string containing variables.



b. Add Numeric and Text Widgets to the screen and bind their values to the variables Temperature, Pressure, and Batchname.



c. After completing the setup, run offline simulation. When the values are changed via the Widgets, the inserted variable values in the Text Widget will update in real time.

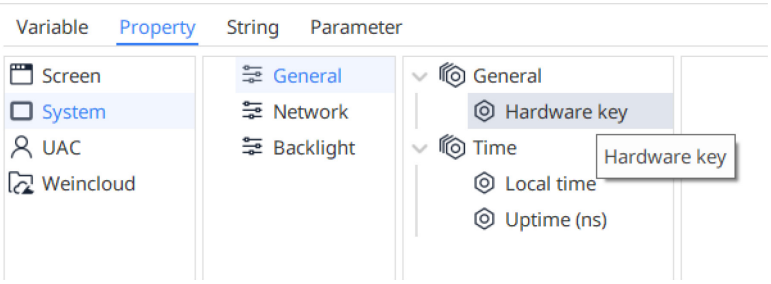


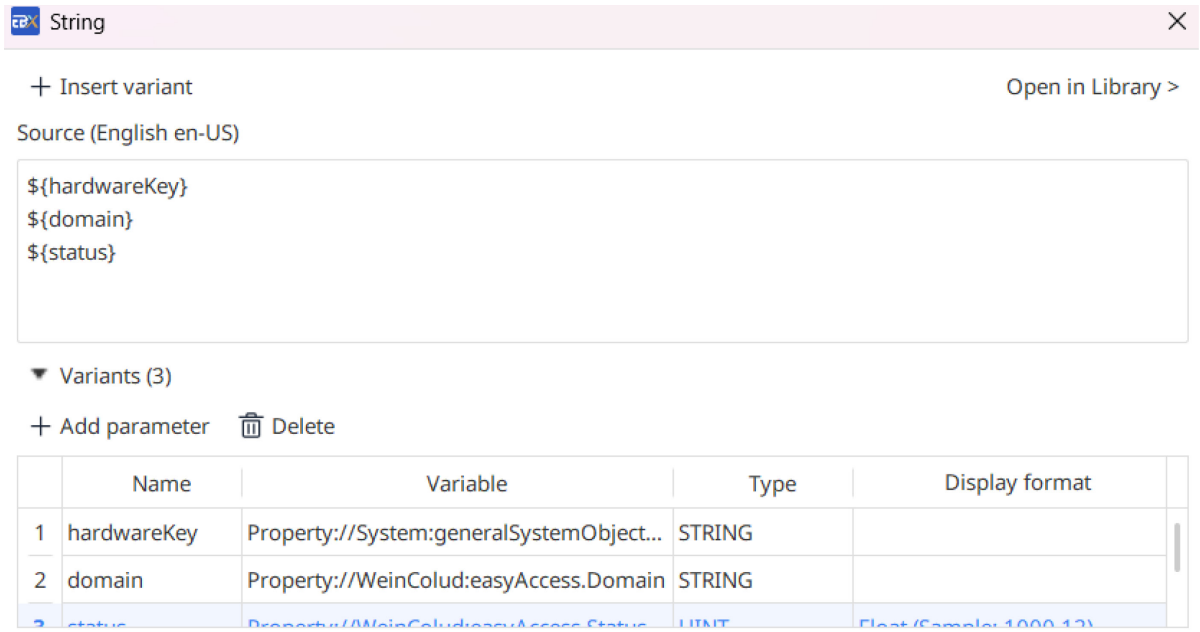
8.2.5. Display Read-Only System Parameters

Variable parameters inserted into a string can be bound to read-only system properties, such as the HMI hardware key (HW key), the Weincloud domain, and the EasyAccess 2.0 connection status.

Implementation Steps

1. Add Parameters and Insert Variant
   a. In the String Table, add a new string entry.
   b. In the string content field, click the [Setting] icon to open the editor.
   c. When adding a parameter, a system property can be bound directly.





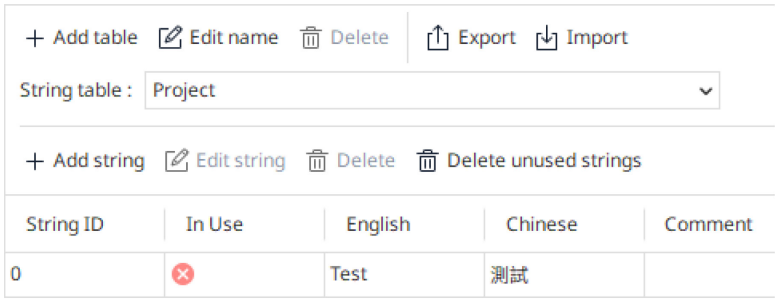
8.3. Multi-language Settings

To use multiple languages, go to [Project] » [Language] to open the language settings page. The language name, font, and language code can be configured for each language.

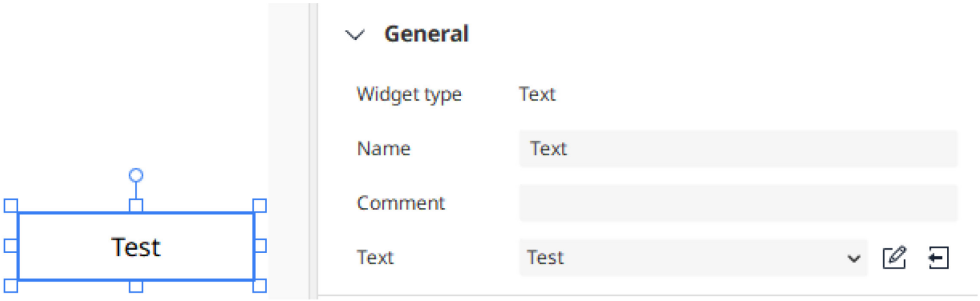


Implementation Steps

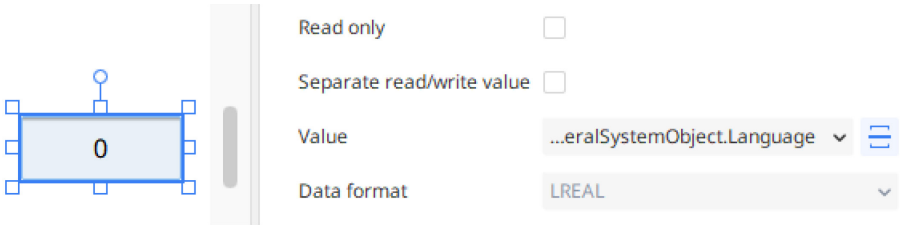
1. Create two languages.
   a. Open [Project] » [Language], and add languages.
   b. Set Language 0 as English and Language 1 as Chinese, and assign appropriate fonts.
2. Add string table entries.
   a. In the String Library, add a new string.
   b. Enter the desired text in both the English and Chinese fields.



3. Create a Text widget and a Numeric widget.
   a. Add a Text widget to the screen, and set its content to the string created in the previous step.



   b. Add a Numeric Widget that points to Property » System » General » Language.



4. Verify the result.
   a. Run offline simulation or download the project to the HMI.
   b. When the numeric value is set to 0, the HMI displays Language 0 (English); when set to 1, it displays Language 1 (Chinese).

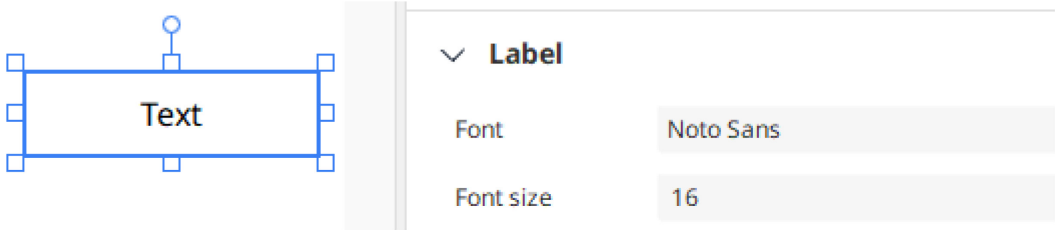


Note that when multiple languages are configured in the project, if the string for the current language is empty, the content for Language 0 is displayed.

8.4. Font Application Logic

EasyBuilder X provides multiple font settings for text display. This section describes the font application logic.
In general, widget text is displayed using the highest-priority available font. If the text cannot be displayed with a higher-priority font, the system falls back to the next lower-priority font. The priority levels are listed below from highest to lowest.

High Priority: Display the text content using the font set in the widget's Label property.
Description: When a Text widget is added, the user can specify the desired display font through the font setting in the Label properties.
If the selected font cannot display the text content, the system uses the medium-priority font to display the content.



Medium Priority: Use the font defined in the language settings as the fallback font for text display.
Description: When the high-priority font cannot display the text content, the system uses the font set in the language settings.
If the font in the language settings still cannot display the text content, the system uses the low-priority font to display the text content.



Low Priority: The system automatically falls back to a font from the font library that can display the text.
Description: When neither the high-priority nor medium-priority font can display the text content, the system automatically selects a suitable font from the font library. This may be the system default font Noto Sans or another font.

9. Condition

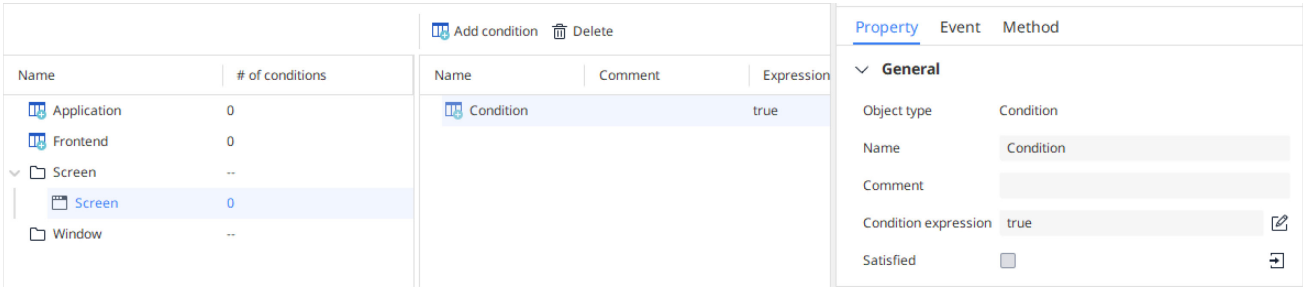
9.1. Condition 9-2
9.2. Use Case: Temperature Monitoring System 9-4

9.1. Condition

Condition a logical control tool in EasyBuilder X that triggers corresponding actions when specified conditions are satisfied. These conditions are based on the values or states of Variables and can be used to control the logical flow and behavior within Screens or Windows. Each Screen or Window requires individual setup for Condition objects, and multiple Condition objects can exist within each Screen or Window. Conditions can also be set for the “Application” (global) or “Frontend” (Viewer layer) levels. For details, refer to Chapter 22: Variant Scope.

9.1.1. Steps to Create a Condition

1. Click on [Project] » [Condition] to open the Condition settings page.
2. On the left, select the Screen or Window where the Condition will be created.
3. On the right, click [Add condition] to add a new Condition.
4. Configure the relevant settings in the Property panel for the Condition.



9.1.2. Condition Properties

Condition Properties	Description
Name	Set the name of the Condition.
Comment	Add or modify a comment for the Condition.
Condition expression	Click the edit button on the right to open the Logic Expression dialog box. Use logical and comparison operators to set the condition expression. For example: ${Namespace.INT} > 1 && ${Namespace.BOOL} == true.
Satisfied	Read-only. Set the Variable triggered when the condition is satisfied (BOOL format).

9.1.3. Steps to Set a Condition Expression

9.1.3.1. Variable Operand

Configure the variables used in the condition expression.

9.1.3.2. Comparison Operator

Comparison operators are used to compare variables or values:

Comparison operator	Description
less than (<)	Returns true when the value on the left is less than the value on the right.
less than or equal (<=)	Returns true when the value on the left is less than or equal to the value on the right.
greater than (>)	Returns true when the value on the left is greater than the value on the right.
greater than or equal (>=)	Returns true when the value on the left is greater than or equal to the value on the right.
equal (==)	Returns true when both sides are equal.
not equal (!=)	Returns true when both sides are not equal.

9.1.3.3. Logic Operator

Logic operators are used to combine multiple conditions:

Logic operator	Description
logic and (&&)	Returns true when all conditions are true.
logic or (||)	Returns true when at least one condition is true.
logic not (!)	Reverses the result of a condition.

9.1.3.4. Condition Expression Examples

Example 1: Numeric Comparison

${Namespace.INT} > 3

This expression is true when the value of Namespace.INT is greater than 3.

Example 2: Boolean Evaluation

${Namespace.BOOL} == true

This expression is true when the value of Namespace.BOOL is true.

Example 3: String Comparison

${Namespace.STRING} == "example"

This expression is true when the value of Namespace.STRING is "example".

Example 4: Logical Combination

${Namespace.INT} > 3 && ${Namespace.BOOL} == true

This expression is true when Namespace.INT is greater than 3 and Namespace.BOOL is true.

Example 5: Logical Not Operation

!${Namespace.BOOL}

This expression is true when Namespace.BOOL is false (because “!” is the logical not operator).

9.1.4. Condition Event Description

A condition has three states: Undetermined, Satisfied, and Unsatisfied. When a condition is created within a specific scope, its initial state is Undetermined. Once the condition becomes Satisfied or Unsatisfied, its state changes accordingly.

Event	Description
Condition satisfied	Triggers the event action when the condition changes to Satisfied. The trigger timing is divided into “Undetermined to Satisfied” and “Unsatisfied to Satisfied”.
Condition unsatisfied	Triggers the event action when the condition changes to Unsatisfied. The trigger timing is divided into “Undetermined to Unsatisfied” and “Satisfied to Unsatisfied”.

9.2. Use Case: Temperature Monitoring System

In industrial automation, a temperature monitoring system is a common application scenario. This section will demonstrate how to use the Condition object in EasyBuilder X to create a simple temperature monitoring system.

9.2.1. System Overview

In this use case, a condition will be established with a variable named “Temperature” to record the current temperature. The alarm threshold is set at 75 degrees, and when the temperature exceeds this limit, the system will trigger an alarm.

9.2.2. Create Variables

First, create a variable in EasyBuilder X to record the temperature value.

1. Click on [Project] » [Variable].
2. Click [Add INT] to create a new variable, and set its name to “Temperature”.
3. Click [Add BOOL] to create another variable, and set its name to “Alarm”.

9.2.3. Create a Condition

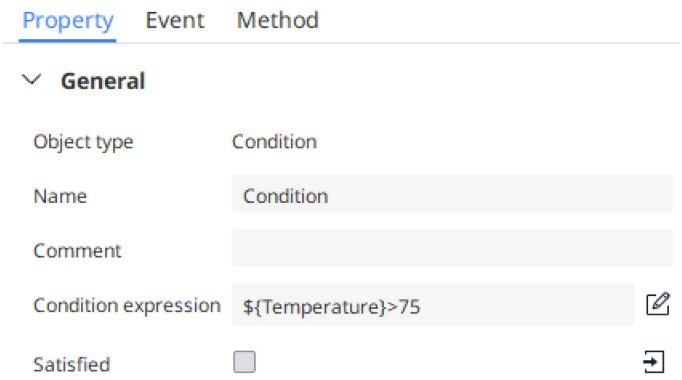
Next, create a Condition to monitor the temperature value.

1. Click [Project] » [Condition] to open the Condition settings page.

2. Select the Screen where the Condition will be created in the left interface.
3. Click [Add condition] in the right interface to create a new Condition.
4. Configure the relevant settings in the Condition's Property panel.

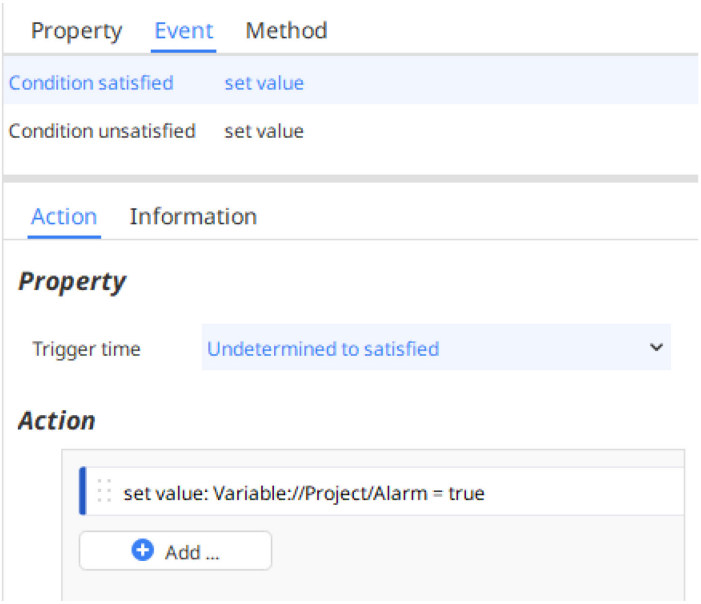
9.2.4. Set the Condition Expression

1. In the Property panel, locate the "Condition expression" field and click the edit button to open the Logic Expression dialog box.
2. Set the condition expression to “${Temperature}>75”.



9.2.5. Set Actions for Condition Satisfied and Unsatisfied

1. Switch to the Event tab in the Property panel.
2. Click [Condition satisfied] and then click the "+" icon to add an Action.
3. Select [Set value].
4. In the Set Value Setting Dialog, configure the variable Alarm=true.
5. Click OK to save the settings.
6. Click [Condition unsatisfied] and then click the "+" icon to add an Action.
7. Select [Set value].
8. In the Set Value Setting Dialog, configure the variable Alarm=false.
9. Click OK to save the settings.



9.2.6. Operation Instructions

1. Add a Numeric Widget to the Screen and bind its Value to the variable Temperature.
2. Add a Lamp Widget to the Screen and bind its Value to the variable Alarm.
3. Simulate temperature changes by modifying the value in the Numeric Widget.
4. When the Temperature variable exceeds 75, observe that the system sets the Alarm variable to true (Condition satisfied).
5. When the Temperature variable falls below 75, observe that the system sets the Alarm variable to false (Condition unsatisfied).

10. UAC

10.1. Overview of UAC 10-2
10.2. UAC Management Interface 10-2
10.3. UAC Property 10-2
10.4. UAC Method 10-3
10.5. Usage 10-5

10.1. Overview of UAC

The User Access Control (UAC) system in EasyBuilder X enables management of user operations on the HMI. Through configuration of login/logout, user addition/deletion, and assignment of Roles and Permissions, widget properties can be bound directly to those permissions, enforcing access control at runtime.

10.2. UAC Management Interface

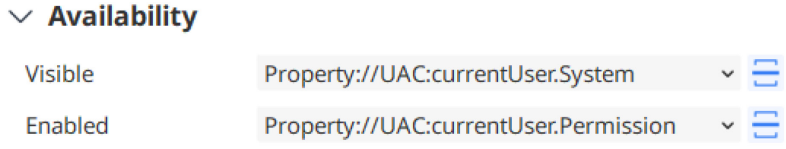
Navigate to [Project] » [UAC] to open the UAC management interface, which provides User, Role, and Security tabs.
On the User tab, define usernames and passwords, assign one or more roles, and manage users via groups. For security, passwords are masked by default and can only be revealed when entering a new password.

On the Role tab, add, rename, or modify roles and their associated permissions, and map permissions to each role.
A user's effective permissions come from the roles assigned on those two tabs.

On the Security tab, password complexity requirements can be defined, including minimum length, uppercase and lowercase letters, digits, and special characters.
The maximum number of failed login attempts and the lockout duration (seconds) can also be configured to strengthen account protection.
If a password set on the User tab does not meet the defined complexity requirements, compilation will fail.

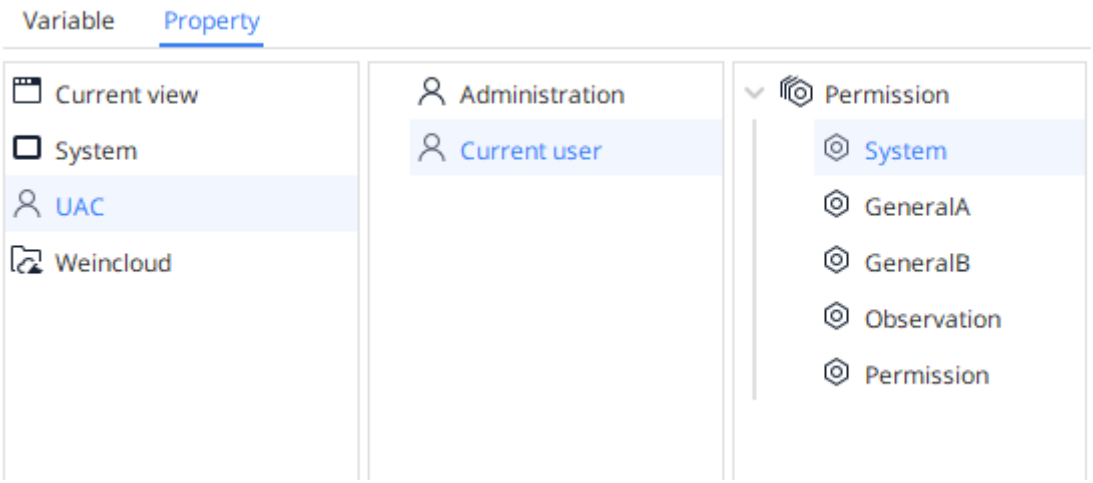
10.3. UAC Property

Each UAC permission in EasyBuilder X is represented as a Boolean property (default OFF). After a successful login, granted permissions switch to ON. Binding a widget's Availability property to those UAC permissions enforces access control at runtime.



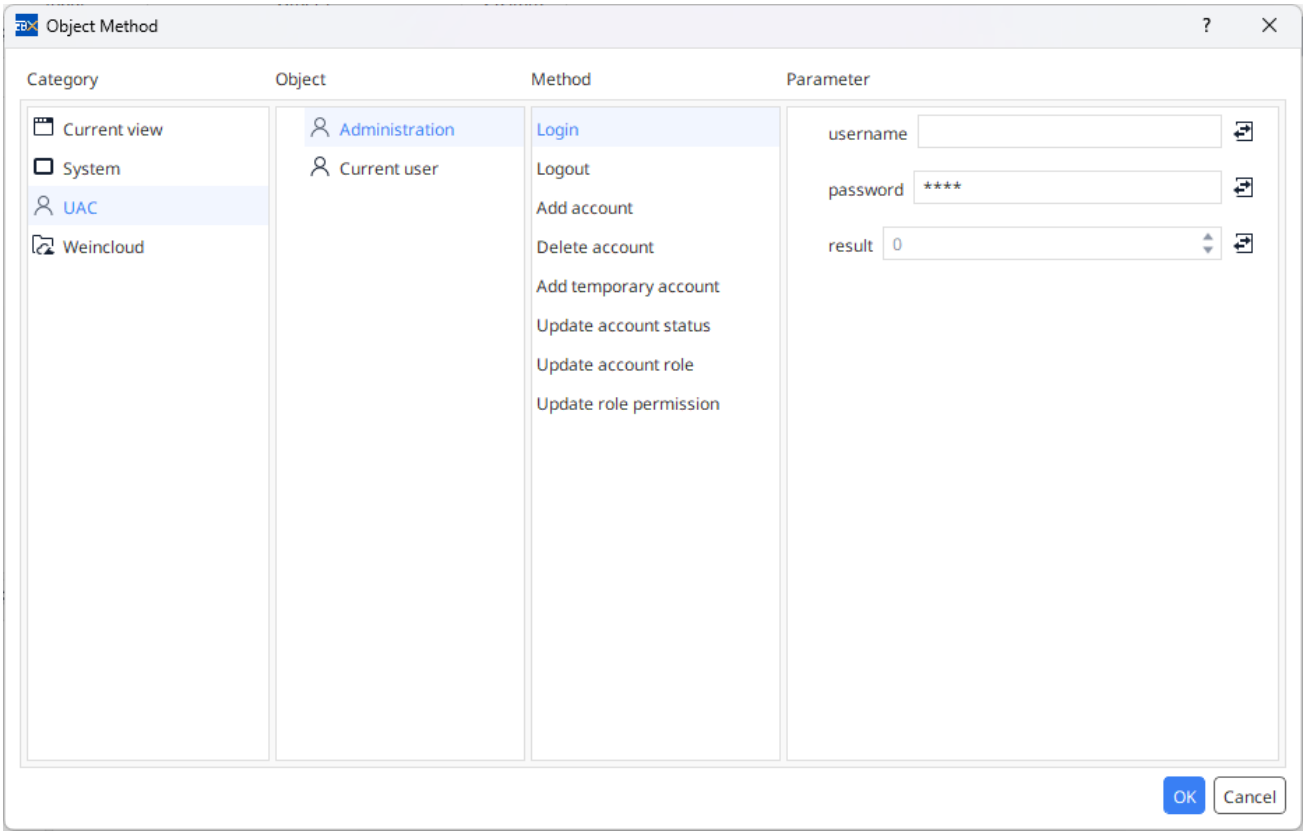
When configuring the binding, select [UAC] under [Property] and choose the desired permission level. After logging in to the HMI, widgets will be displayed or become operable

according to their configured permissions.



10.4. UAC Method

UAC methods are global objects that can be invoked using the [Call object method] action (e.g., in an event-triggered Action or a widget's Task).
For example, when configuring the general properties of a button, go to [Task] » [Add] and select [Call object method]. Then, in the [Category] settings, choose [UAC], and select the desired object and method.



Method	Description	Parameters
Login	Log in the user	Username (String) Password (String) Result (UDINT)
Logout	Log out the user	Result (UDINT)
Change password	Change the user password	Old Password (String) New Password (String) Result (UDINT)
Add account	Create a new user	Username (String) Password (String) Role name (String) Result (UDINT)
Delete account	Remove a user	Username (String) Result (UDINT)
Add temporary account	Create a temporary user	Username (String) Password (String) Role name (String) Valid period in days (UINT) Valid period in minutes (UINT) Preserved after reboot (BOOL) Result (UDINT)
Update account status	Set whether the current user account is valid.	Username (String) Status: 0=Invalid, 1=Valid (BYTE) Result (UDINT)
Update account role	Add or remove a role for a user	Operation: 0=add, 1=delete (BYTE) Username (String) Role name (String) Result (UDINT)
Update role permission	Add or remove a permission for a role	Operation: 0=add, 1=delete (BYTE) Role name (String) Permission name

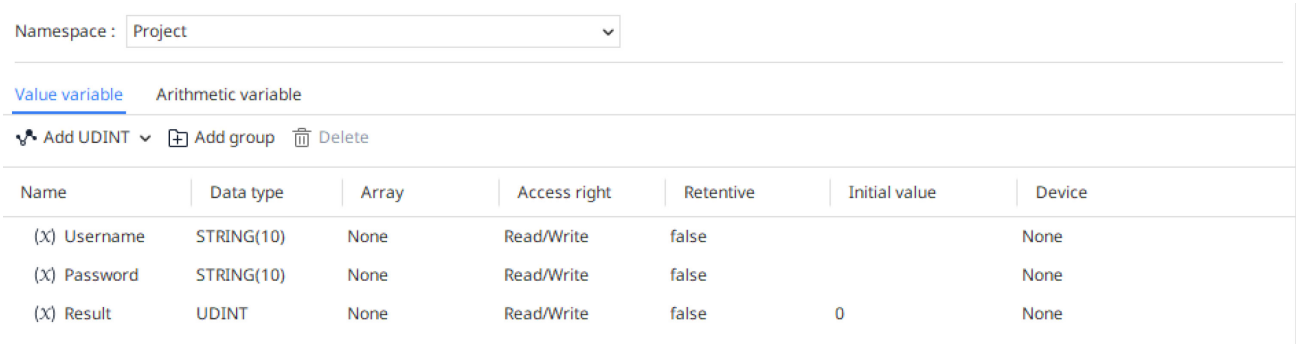
	(String) Result (UDINT)

Result Code	Description
0	No error.
2	Internal error.
100	Invalid format.
1003	Service unavailable.
3001	Remote disconnected.
4001	Operation denied.
6004	Database error.
30000	Unknown error.
30002	Incorrect login information.
30003	Account expired.
30004	Account not activated.
30005	Session limit reached.
30006	Client locked out.
30101	Password does not meet the complexity requirements.
30102	Invalid import data.
30300	Invalid resource.
30301	Resource in use.
30302	Resource is a system default.
30303	Resource quota reached.

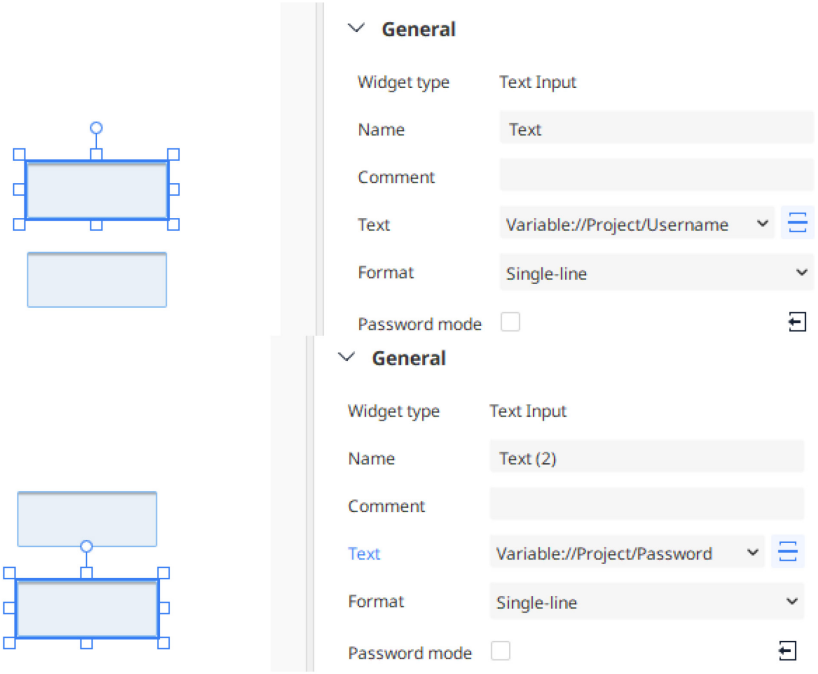
10.5. Usage

10.5.1. Implementing Login/Logout

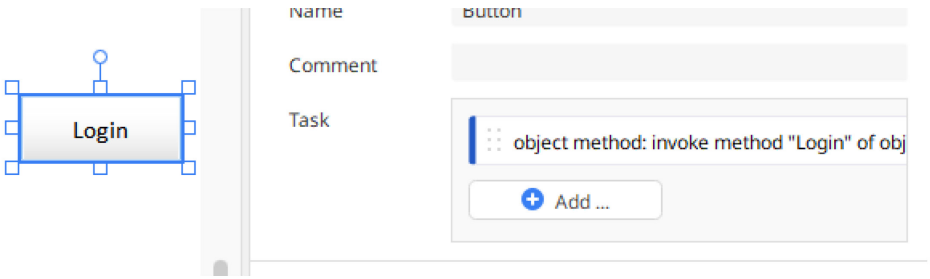
1. On the UAC tab, add usernames and passwords.
2. In Variable Library, create two variables for username and password.

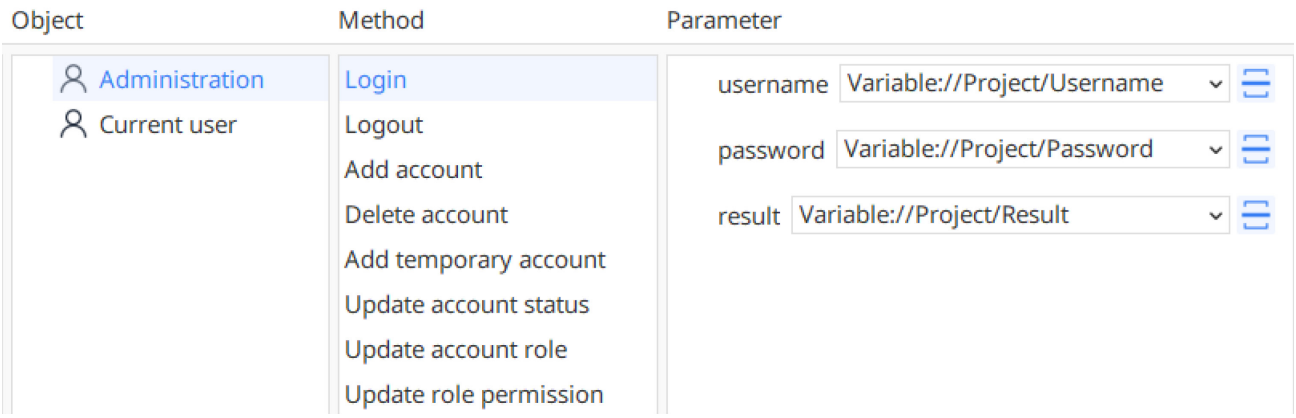


3. Place two Text widgets on the screen and bind their Value to those variables.



4. Add two Button widgets: one for the Login method and one for the Logout method, binding each method's parameters to the corresponding variables. In [Property] » [Task], add a new task, choose [Call object method], and select [UAC] » [Administrator] » [Login] (Logout).





5. Bind widget Availability properties to UAC permissions (see Section 10.3) to implement login/logout functionality.

11. Data Logging

11.1. Data Sampling 11-2
11.2. Trend Display 11-4
11.3. Data Display 11-7

11.1. Data Sampling

The Data Sampling object in EasyBuilder X records variable data using different sampling modes (e.g., time-triggered or condition-triggered) and and stores the data as history files for later analysis or reference.

11.1.1. Steps to Create Data Sampling

1. Click [Data / History] » [Data Sampling] to open the Data Sampling settings page.
2. Click [Add data sampling] to add a new Data Sampling object.
3. In the Property Panel, set the desired options—sampling mode, bound variables, history-file storage mode, etc.

11.1.2. Data Sampling Properties

Item	Description
General	
Name	Set Data Sampling name.
Comment	Set Data Sampling description.
Sampling Configuration	
Mode	Select how sampling is triggered: Time-based: sample data from the bound variables at a fixed time interval. Trigger-based: sample data only when the object's Sample Method is called.
Time interval	Specify the interval for time-based sampling.
Data	Bind one or more variables for recording.
Log	
Storage location	Currently, only saving all history records in a single file is supported. Set the storage location here: **No storage**: Do not save files. **HMI Memory**, **USB Disk 1/2, SD Card**: Requires pre-defined folder name.
Log name	Set the file name.
Size limit	Use the configured record count setting as the file size limit.
Record count	Set the number of records per file (range: 500–5,000,000).

Log mode	Only cyclic recording is supported. When the record count exceeds the set limit, the oldest data will be overwritten.
Flush period (seconds)	Data sync interval. Only 60 seconds is supported.

11.1.3. Data Sampling Events

During data sampling, the system provides event monitoring so that users are informed of synchronization status changes or errors.
The following are detailed descriptions of the events:

Data Flush Status Changed

Triggered when the sync status changes. Common uses:

- 0: Disconnected – alerts that the external device is not connected.
- 3: Storing – confirms that data is being synchronized.

Parameters	status (UINT): External device sync status code.
Status	0: Disconnected: External device not connected.
	1: Connecting: External device is connecting.
	2: Connected: External device connected successfully.
	3: Storing: External device is saving data.

Data Flush Error Occurred

Triggered when an error occurs during synchronization. Common uses:

- 4000: Access denied – notifies the user to check the permission settings.
- 6002: Database is corrupted – advises the user to back up the data and repair the database.

Parameters	error (UINT)：Synchronization error code.
Error	0: No error.
	1: Unknown error.
	100: Invalid database parameter or record ID.
	2000: Record not found.
	3000: Failed to connect to the external device or database.

	4000: Access denied.
	6000: Failed to open the database or table.
	6001: Failed to write to the table.
	6002: The database is corrupted or the data format is inconsistent.
	6003: SSL/TLS handshake failed.

11.1.4. Data Sampling Methods

The Data Sampling object provides a set of methods for precise control over sampling, synchronization, and data management.

Method	Description
Sample (Trigger-based)	Manually trigger a sample to record the current data values.
Pause sampling (Time-based)	Pause time-based data sampling.
Resume sampling (Time-based)	Resume time-based data sampling.
Clear buffer	Clear sampled data stored in the buffer.
Flush	Manually trigger data synchronization to an external storage device or target location—ideal for immediate backup when an external device is connected.

11.2. Trend Display

The Trend Display widget visualizes data trends by binding to a Data Sampling object, drawing trend graphs so users can monitor historical or real-time data changes.

11.2.1. Trend Display Properties

Trend Display offers various properties to customize its appearance and behavior. The following are the property groups with detailed descriptions:

Item	Description
General	
Name	Set the Widget's name.
Comment	Set the Widget's description.
Data sampling	Bind a Data Sampling object.
Channels setting	Enable channels and configure details for each.
Custom data source enabled	Allow control of the displayed history range.
Data source (nth date or file)	Specify the historical data source to display, such as a specific date or the number of days ago. 0 represents today, 1 represents yesterday, and so on.
Data source range mode	Select how to define the range for multiple historical data records. Number of days – Display the number of days specified in [Data source range]. Specific days – Use [Data source] as the start date and [Data source range] as the end date.
Enable watch line	Allow a vertical line to mark and inspect a specific time point.
Watch line color	Set the color of the watch line.
Enable watch line fade out	Automatically hide the watch line after a delay.
Watch line elapsed time (second)	Set the fade-out time of the watch line in seconds.
Enable timestamp output	Enable the timestamp parameter for the Watch Line Set event.
Output date in timestamp	Include the date in the timestamp parameter.
Trend	
Font	Set the font for timestamps and other text.
Range on time axis (second)	Define the time span shown on the X-axis.
Time axis mode	Choose Absolute (real time) or Relative (elapsed).

Sampling datetime format	Format for the date and time shown with the latest sample (top-left).
Sampling datetime color	Set the color of the sampling timestamp text.
Enable grid	Display grid lines on the chart.
Interval on time axis (second)	Set the spacing between vertical grid lines.
Division on value axis	Specify the number of horizontal grid divisions.
Grid color	Set the grid line color.
Scale color	Set the color of axis labels.
Scale datetime format	Format for dates and times on the time axis.
Scale datetime position	Select the position of the time scale: **Below time axis**: Displayed below the time axis. **Above time axis**: Displayed above the time axis.
Show channel value scale while channel invisible	Keep the Y-axis scale visible even when its channel is hidden.
Value scale alignment	**Select Center on tick** (centered on tick marks) or **Full text** (full label).
Display	
Show options button	Show an option button for chart controls.
Support channel visibility option in option button	Include a channel-visibility toggle.
Support value scale visibility option in option button	Include a Y-axis scale toggle.
Support value axis scrolling option in option button	Include a Y-axis scroll toggle.
Enable value axis scrolling	Enable touch operation to allow vertical scrolling along the Y-axis.
Zoom button visible	Enable the zoom buttons for zooming in or out of the chart.
Enable zooming along time axis	Allow pinch-zoom on the X-axis.
Enable zooming along value axis	Allow pinch-zoom on the Y-axis.
Outline	
Picture	Set a background image.
Picture color	Some system pictures allow color selection.

Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Background	
Fill	Set the background color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Availability	
Visible	Set whether the Widget is visible.
Enabled	Set whether the Widget allows control.

11.2.2. Trend Display Events

Trend Display supports event triggers for monitoring data changes or user interactions. The following events are available:

Watch line set

Triggered when a watch line is placed on the chart. Use this event to capture channel values at a specific time point.

Parameters	Description
timestamp	UDINT: Time at which the event is triggered.
channel values	STRING: Channel data values at the watch line.

Data sampled

Triggered each time new data is sampled. Use this event to log sampling times or initiate post-processing.

Parameters	Description
timestamp	UDINT: Time at which the event is triggered.

11.3. Data Display

The Data Display widget shows data collected by Data Sampling. It must be bound to a Data Sampling object so users can view historical records or real-time changes.

11.3.1. Data Display Properties

Data Display provides various property settings to customize its appearance and behavior. The properties are grouped as follows:

Item	Description
General	

Name	Set Data Display name.
Comment	Set Data Display description.
Data Sampling	Bind to a Data Sampling object.
Display	
Show header	Show or hide the column header.
Columns	Configure each column (name, width, alignment, order).
Sequence number display format	Set format for sequence numbers.
Date/time format	Set format for date and time of samples.
Sort order	Sort data in ascending or descending time order.
Show options button	Enable an option button for user operations.
Appearance	
Style	Set the style of the Data Display.
Header text color	Set the text color of the column header.
Header background color	Set the background color of the column header.
Row background color	Set the background color of the column.
Grid color	Sets the grid line color.
Horizontal padding	Set spacing between text and column edge.
Stretch last column	Stretch last column to fit widget size.
Background	
Fill	Set background color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Outline	
Picture	Set a background image to enhance or identify display appearance.
Picture color	Some system pictures allow color selection.
Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Label	
Font	Set text font.
Font size	Set text size.
Bold	Enable bold text.

Italic	Enable italic text.
Color	Set text color.
Availability	
Visible	Set whether the Widget is visible.
Enabled	Set whether the Widget allows control.

12. Project Setting

12.1. Overview 12-2
12.2. General 12-2
12.3. Remote 12-2
12.4. Security 12-2
12.5. HMI Model 12-3

12.1. Overview

Project Setting manages key aspects of an HMI project file. It enables detailed adjustment of system behavior, security mechanisms, and extension features to match application requirements. Access it via [Project] » [Project Setting].

12.2. General

General settings are used to configure HMI operation-related functions.

Setting	Description
Options	
CSV delimiter	Specify the separator used when importing or exporting the String Table.

12.3. Remote

Remote settings are used to configure functions related to connections to this HMI.

Setting	Description
cMT Remote	
Max connection count	Set the maximum number of simultaneous connections to this HMI via cMT Remote. The allowed range is 0–10.

12.4. Security

Security settings safeguard the project file contents against unauthorized access, tampering, and disclosure.

Setting	Description
Project file password	
Enable file password	When enabled, the project file is password-protected. The correct password is required to open it. This prevents unauthorized access or modification.
Communication security	
Enable secure communication	When enabled, gRPC communication is encrypted.

Password requirements:

- Maximum length: 32 characters
- Allowed characters:
  Uppercase letters A-Z
  Lowercase letters a-z
  Digits 0-9
  Special Symbols: ! @ # $ % ^ & * ( ) - _ + = { } [ ] : ; < > . , ? ~ |
- The correct file password must be entered to disable file password protection or change the password.

12.5. HMI Model

The HMI model used in the project can be changed here. Because different models may have different serial ports and screen resolutions, this function provides a convenient way to adjust widget display settings, making it easier to switch the project between different HMI models and download it.

Setting	Description
Model Information	
Series	Select a different HMI model series.
Orientation	Select landscape or portrait display mode.
Apply	Available only after the model has been changed. Clicking this opens the [Adjust Objects] window.
Adjust Objects	**Resize popup/embedded windows** Adjust the size of popup windows and embedded windows, and to scale window objects proportionally. **Adjustment behavior for objects when windows and screens are resized** ● No adjustment ● Linear resize ● Width-driven resize (fixed aspect ratio) ● Height-driven resize (fixed aspect ratio) ● Average-driven resize (fixed aspect ratio)

13. Alarm

13.1. Alarm 13-2
13.2. Alarm Settings 13-3
13.3. Alarm Display 13-8
13.4. Alarm Bar 13-10

13.1. Alarm

The Alarm function is an essential mechanism for ensuring equipment safety and production stability. When an abnormal condition occurs—such as parameters exceeding limits, equipment faults, or improper operation—the HMI immediately notifies the user through an alarm. On-screen prompts alert personnel to take timely action, helping reduce risks and losses.

The purpose of alarms is to provide real-time notifications, support judgment and decision-making, and act as a core safeguard. For traceability and analysis, the system also records complete alarm histories, including the time of occurrence and clearance, along with related device and parameter information. These records reveal the frequency and causes of issues and serve as a valuable reference for preventive maintenance.

Alarms can be categorized by severity, such as notification, alarm, warning, and critical, enabling users to prioritize responses. With historical logs and statistical analysis, users can monitor equipment conditions more effectively, improve productivity, and strengthen quality control. Alarm records can also support accountability and compliance with industry safety regulations.

In EasyBuilder X, the alarm function consists of two parts: Alarm and Alarm Display.

- **Alarm**: Used to configure and manage alarm rules by binding data sources, setting trigger conditions, and defining severity levels. When conditions are met, it triggers events, generates messages, and logs history. It can also require user acknowledgment or trigger linked actions to ensure timely handling of abnormalities.
- **Alarm Display**: A widget for viewing and interacting with alarms. It shows active alarms on the HMI screen and provides functions such as filtering, sorting, acknowledgment, shelving, and history query. With Alarm Display, users can quickly grasp system status and respond appropriately based on alarm priority and details.

13.1.1. Steps to Create Alarm

1. Click [Data / History] » [Alarm] to open the Alarm settings page.
2. In [Alarm manager] » [Property], set the file storage location.
3. Click [Add alarm] to add a new Alarm object.
4. In the Property Panel, configure settings such as alarm name, message, class level, and bound variables.
5. In the Alarm Class tab, define alarm classes and their priorities. Alarms can be displayed in priority order.
6. In the Alarm Tag tab, define alarm tags as needed to organize alarms by category.
7. After creating the alarm, go to the screen and select [Data / History] » [Alarm Display] to

add an Alarm Display widget to show triggered alarms.

13.2. Alarm Settings

Alarm is used to configure and manage alarm conditions. It serves as a core element for abnormality detection and event notification. By combining data sources with trigger conditions, users can define alarm rules, set thresholds, assign severity levels, specify acknowledgment methods, and configure reset conditions.

13.2.1. Alarm Manager

Alarm Manager centrally manages all alarm records within a project, handling the storage, output, and maintenance of alarm data. Through property settings, users can specify the storage location (e.g., internal memory or external USB device), define file names, set capacity limits, and determine the maximum number of records to be kept. The system supports cyclic logging to ensure the latest alarm information is always preserved during long-term operation. A periodic write interval can also be configured so that data is refreshed to the target location at regular intervals.

In addition to basic properties, Alarm Manager can respond to system status changes through events. For example, it can trigger actions when the data refresh status changes or when an error occurs during writing, allowing users to take further steps. Operational functions include acknowledging all alarms at once, clearing temporary data, or forcing buffered records to be written immediately—ensuring that important information is not lost.

13.2.1.1. Alarm Manager Properties

Alarm Manager properties define how alarm records are stored and managed. Users can specify the storage location (internal memory or external USB device), name the log file, and set both capacity limits and the maximum number of records. Cyclic logging ensures the log is continuously updated with the latest records, even during extended operation. The write interval can also be configured, balancing timeliness and reliability of alarm record updates.

Property	Description
Log	
Storage location	Set the storage location for alarm records.
Log name	Define the file name for alarm records.
Size limit	Use the configured record count setting as the file size limit.
Record count	Set the maximum number of records for the historical alarm file. The configurable range is 500

	to 5,000,000.
Log mode	Only cyclic logging is supported; oldest records are deleted when the limit is reached.
Flush period (seconds)	Only 60-second sync interval is supported.

13.2.1.2. Alarm Manager Events

The Alarm Manager provides an event monitoring function that allows users to promptly detect synchronization status changes or error occurrences.

The following are detailed descriptions of the events:

Data Flush Status Changed

Triggered when the sync status changes. Common uses:

- 0: Disconnected – alerts that the external device is not connected.
- 3: Storing – confirms that data is being synchronized.

Parameters	status (UINT): External device sync status code.
Status	0: Disconnected: External device not connected.
	1: Connecting: External device is connecting.
	2: Connected: External device connected successfully.
	3: Storing: External device is saving data.

Data Flush Error Occurred

Triggered when an error occurs during synchronization. Common uses:

- 4000: Access denied – notifies the user to check permissions settings.
- 6002: Database is corrupted – advises the user to back up the data and repair the database.

Parameters	error (UINT): Synchronization error code.
Error	0: No error.
	1: Unknown error.
	100: Invalid database parameter or record ID.
	2000: Record not found.
	3000: Failed to connect to the external device or database.

	4000: Access denied.
	6000: Failed to open the database or table.
	6001: Failed to write to the table.
	6002: The database is corrupted or the data format is inconsistent.
	6003: SSL/TLS handshake failed.

13.2.1.3. Alarm Manager Methods

The Alarm Manager provides methods that allow users to flexibly control alarm records. Typical methods include acknowledging all alarms at once, clearing the temporary buffer, and immediately forcing buffered records to be written to storage.

Method	Description
Acknowledge all	Acknowledge all alarms at once.
clearBuffer	Discard the remaining data in the buffer without writing it to the log file.
Flush	Write buffered data to the storage location immediately.

13.2.2. Alarm Properties

The Alarm properties define the basic information and trigger conditions of an alarm. Users can set the name, comment, ID, and class, and determine the trigger basis by binding tags or using conditional expressions. Properties also allow specifying display text and auxiliary message text, as well as enabling or disabling condition detection to control whether the alarm is actively monitored.

Property	Description
General	
Name	Set the alarm name.
Comment	Add alarm description.
ID	Set the alarm ID.
Class	Define the alarm class.
Tags	Set multiple alarm tags.
Condition expression	Define the trigger condition for the alarm.
Message	Set the text displayed when the alarm is triggered. Embedded String variables from the String Table are also supported.

Info text	Provides detailed alarm information. It can be triggered by the “Request Info” method of the Alarm Display widget.
Condition detection	Set whether to monitor the variable and trigger alarms dynamically.
State	Display Alarm states: 0: Normal 1: Active 2: Acknowledged 3: Inactive

13.2.3. Alarm Events

The Alarm provides state transition events for real-time notification when the alarm status changes. Both the new state and the previous state are reported. An “Info Requested” event is also provided, and is triggered when information for the alarm is requested. user

Event	Description
State changed	Triggered when the alarm state changes, and reports both the new state and the previous state. Parameters: state, previous state. For state values, refer to the state list in Alarm Properties (0: Normal, 1: Active, 2: Acknowledged, 3: Inactive).
Info requested	Triggered when alarm information is requested.

13.2.4. Alarm Methods

The Alarm methods provide direct control over alarms. Currently, the main method is Acknowledge, which sets an alarm to Acknowledged. This allows users to update the alarm status once the issue has been noticed and handled, helping the system distinguish between acknowledged and unprocessed alarms.

Method	Description
Acknowledge	Acknowledge the alarm state.

13.2.5. Alarm Class

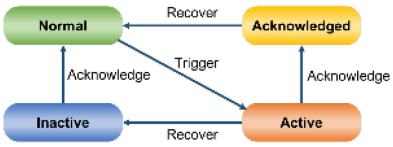
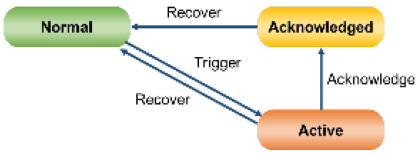
The Alarm Class is used to group and categorize different types of alarms. With Alarm Class, users can assign severity levels, define handling modes, and specify whether alarms require

acknowledgment, are logged, or have priority assigned.
By applying Alarm Classes, the system can enforce consistent management rules based on the importance and response requirements of alarms, preventing confusion caused by scattered individual settings.

Property	Description
General	
Name	Set Alarm Class name.
Comment	Set Alarm Class description.
Mode	Set the operation mode when an alarm is triggered.
Log	Define whether the alarm is stored in the history log.
Priority	Set the priority of the alarm. Alarms are sorted starting from the highest priority.
Active text color	The text color in the active state.
Active background color	The background color in the active state.
Inactive text color	The text color in the inactive state.
Inactive background color	The background color in the inactive state.
Acknowledged text color	The text color in the acknowledged state.
Acknowledged background color	The background color in the acknowledged state.

The Mode property defines how an alarm behaves in relation to acknowledgment, clearance, and display. The following modes are available:

Mode	Acknowledge-ment	Acknowledge-ment Count	When Condition Clears	Display
Alarm without acknowledgement	No	0	Cleared immediately	Informational message
Alarm with acknowledgement	Yes	1	Remains until acknowledged	Critical alarm

				
Alarm with optional acknowledgement	Optional	0 or 1	Cleared when condition is not met; acknowledgement not required 	Medium-priority alarm, balancing flexibility and logging

13.2.6. Alarm Tag

Alarm Tag provides a way to label and categorize alarms by source, type, or other criteria, making them easier to filter and group in the Alarm Display or historical records.
This feature is especially useful in large projects or cross-device monitoring applications, where multiple tags can be combined to create multi-dimensional alarm views. For example, a single alarm can be assigned tags such as Production Line 1, Temperature Monitoring, and High Priority.
Up to three Alarm Tags can be added in an alarm's properties and displayed in the Alarm Display, improving system flexibility and usability while helping users quickly identify critical events and take appropriate follow-up action.

13.3. Alarm Display

The Alarm Display is a widget used to present the status of defined alarms in a project. Triggered alarms are displayed on screen in real time according to the configured conditions, helping users monitor and manage alarm information.

13.3.1. Alarm Display Properties

Alarm Display provides various property settings for customizing its appearance and behavior. The properties are grouped as follows:

Property	Description
General	
Name	Set Alarm Display name.
Comment	Set Alarm Display description.
Alarm classes	Select the Alarm Classes to be displayed.

Source of item data	Select to display either current alarms or historical alarms.
Display	
Show column header	Display column names.
Show caption	Enable the caption.
Columns	Select columns to display, including name, width, and order.
Alarm ID format	Set the format for displaying the alarm ID.
Date/time format	Set the format for displaying the alarm trigger time.
Sorting order	Define the sorting order based on alarm occurrence time.
Appearance	
Style	Set the display style of the Alarm Display.
Style color	Set the style color.
Header text color	Set the text color of the column header.
Horizontal padding	Set spacing between text and column edge.
Stretch last column	Stretch last column to fit widget size.
Caption	
Caption font size	Set the caption font size.
Caption text color	Set the caption text color.
Caption background color	Set the caption background color.
Caption text	Set the caption content.
Background	
Fill	Set background color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Outline	
Picture	Set a background image to enhance or identify display appearance.
Picture color	Some system pictures allow color selection.
Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Label	

Font	Set text font.
Font size	Set text size.
Bold	Enable bold text.
Italic	Enable italic text.
Availability	
Visible	Set whether the Widget is visible.
Enable	Set whether the Widget allows control.

13.3.2. Alarm Display Methods

The Alarm Display methods provide direct control of the Alarm Display. Current methods mainly include acknowledging all alarms, acknowledging selected alarms, and requesting information for selected alarms.

Method	Description
Acknowledge all	Acknowledge all active alarms.
Acknowledge	Acknowledge the selected alarm.
Request info	Request information about the selected alarm.

13.4. Alarm Bar

The Alarm Bar is a widget used to display alarm information. It presents alarm content in a list or summary format, allowing users to quickly grasp the current alarm status.
Compared with the Alarm Display, the Alarm Bar occupies less screen space, making it suitable for placement at the top or bottom of the screen as an auxiliary display area for real-time monitoring. In contrast, the Alarm Display provides more detailed alarm information and is typically used for comprehensive alarm management screens.

13.4.1. Alarm Bar Properties

Alarm Bar provides various property settings for customizing its display. The properties are grouped as follows:

Property	Description
General	
Name	Set Alarm Bar name.
Comment	Set Alarm Bar description.
Alarm classes	Select the Alarm Classes to display.
Transition interval	Set the interval for switching between alarm messages.

Show current/total count	Display the current alarm index and total number of alarms on the alarm bar.
Display	
Columns	Select columns to display, including name, width, and order.
Alarm ID format	Set the format for displaying the alarm ID.
Date/time format	Set the format for displaying the alarm trigger time.
Sorting order	Define the sorting order based on alarm occurrence time.
Background	
Fill	Set background color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Outline	
Picture	Set a background image to enhance or identify display appearance.
Picture color	Some system pictures allow color selection.
Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Label	
Font	Set text font.
Font size	Set text size.
Bold	Enable bold text.
Italic	Enable italic text.
Availability	
Visible	Set whether the Widget is visible.

14. Recipe

14.1. Recipe Database 14-2
14.2. Recipe View 14-4
14.3. Use Case: Managing Production Batch Parameters 14-6

14.1. Recipe Database

The Recipe Database is used to store and manage recipe data. Users can define the recipe schema and its data fields. Each recipe can contain multiple elements, and each element can be bound to a variable in the HMI project.

14.1.1. Steps to Create Recipe Database

1. Click [Data / History] » [Recipe Database] to open the Recipe Database settings page.
2. In the [Use recipe data from] drop-down list, select whether to overwrite the recipe data stored in the HMI during project download.
3. In the Recipe area on the left, click [Add recipe] to add a recipe object, and rename it as needed.
4. When a Recipe Database is created, the system automatically creates a fixed string element named “Record Name” in the element list. This element serves as the unique identifier for each recipe record, and its properties cannot be modified or deleted.
5. In the Element area in the middle, click [Add element] to add recipe data elements.
6. Configure each element’s name, data type, data range, and bound variable.
7. In the Property panel on the right, configure more detailed settings for the selected element.
8. After configuring the recipe elements, switch to the [Record] tab to add recipe data in advance.

14.1.2. Recipe Database Properties

Property	Description
General	
Name	Set the recipe name.
Comment	Add recipe description.
Audit Trail	
Enable (Add/Update/Delete)	When selected, Audit Trail is triggered when an user adds, updates, or deletes a recipe, and the changes are recorded.
Enable (Apply)	When selected, Audit Trail is triggered and the action of applying the recipe to the device is recorded.
Confirmation type	Select None, Acknowledgement, or Electronic Signature. If enabled, additional confirmation is required before the action is performed.

	Otherwise, the action is denied.
Comment required	When selected, the user must enter a non-empty comment in the dialog box.

14.1.3. Element Properties

Property	Description
General	
Name	Set the element name.
Comment	Add element description.
Recipe Element	
Data type	Set the data type of the recipe element.
Display format	Set the display format of the recipe element.
Bind to	Bind the recipe element to a project variable. When a recipe record is read or written, the variable value synchronizes with the recipe data.

14.1.4. Recipe Database Methods

The Recipe Database object provides methods for reading, adding, editing, and deleting recipe records.

Method	Description
Apply to device	Load the recipe record specified by the [Record Name] parameter, and write its data to the target variables bound to the recipe elements.
Record count	Count the total number of records currently stored in the Recipe Database, and output the result to the [Record Count] parameter.
Add record	Add a new recipe record with the specified [Record Name]. Each element is initialized to the default value defined in the recipe structure.
Add record from device	Read the current values from the bound target variables and use them to create a new recipe record. The new record is created with the specified [Record Name].
Update record	Read the current values from the bound variables and use them to update the existing recipe record identified by the specified [Record Name].
Delete record	Delete the existing recipe record identified by the

	specified [Record Name] from the recipe.
Delete all records	Delete all records from the recipe.

Operation Result Codes (UDINT Output):

Code	Description
0	No error.
4	The operation was cancelled.
50001	The specified recipe record was not found. (Check the record name.)
50003	The maximum number of recipe records has been reached. (Remove existing records.)
50004	The recipe record data is invalid. (Check the value type and data range of each element.)
50005	Failed to transfer the recipe to or from the device.
50006	Failed to write to the bound variables when applying the recipe. (Check the variable type and range bound to each element.)
50007	Recipe record names must be unique within the same recipe. (Use a record name that is not already used in this recipe.)
50009	The recipe record name exceeds the maximum length limit.
50010	Invalid recipe name.
50011	Invalid parameter. (Check the record name parameter binding.)
50012	Failed to write to the database. (Check disk space or write permissions.)
50013	A command is in progress. (Wait for the previous command to complete.)
50014	Unknown error.
50401	Unsupported data type.
80006	An Audit Trail error occurred; the request was rejected. (Check Audit Trail storage.)

14.2. Recipe View

The Recipe View widget displays recipe data from the Recipe Database in a table on the HMI screen. Users can browse and edit records directly when “Read only” is disabled.

14.2.1. Recipe View Properties

Property	Description
General	
Name	Set the Widget’s name.
Comment	Set the Widget's description.
Recipe	Bind to a Recipe Database to display. All existing recipes are listed here.
Read only	When selected, users cannot edit data in the Recipe View on the HMI screen.
Display	
Show header	Display the element names as table headers.
Show search bar	Display a search bar for quickly locating recipe records.
Columns	Select which elements to display in the table. Clicking the gear icon opens the column display configuration window for adjusting display settings for each element.
Sorting by	Set the sorting criteria using any element in the recipe.
Appearance	
Style	Set the display style of the Recipe View. If Default is selected, the colors for the column fields, text, background, and grid lines must be configured manually.
Style color	Set the style color.
Header text color	Set the text color of the column header.
Horizontal padding	Set spacing between text and column edge.
Stretch last column	Stretch last column to fit widget size.
Background	
Fill	Set background color and pattern.
Radius	Set the corner radius.

Border	Set the border style, width, and color.
Outline	
Picture	Set a background image.
Picture color	Some system pictures allow color selection.
Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Label	
Font	Set text font.
Font size	Set text size.
Bold	Enable bold text.
Italic	Enable italic text.
Color	Set text color.
Availability	
Visible	Set whether the Widget is visible.
Enable	Set whether the Widget allows control.

14.2.2. Recipe View Events

The Recipe View supports event triggers that respond to user actions when selecting records in the table.

Selection Changed

This event is triggered when the user selects a different record in the Recipe View.
It is commonly used to load or display detailed recipe information immediately after a recipe is selected.

Parameter	Description
Record Name	STRING: Name of the newly selected record. If no record is selected, the value is an empty string.

14.3. Use Case: Managing Production Batch Parameters

The Recipe Database is suitable for applications that require storing and switching between multiple sets of production parameters, such as process settings for different products or batch-specific values.

Example: Quickly Switching Product Recipes

Suppose a production line must apply different temperature and pressure settings for different products (Product A and Product B).

1. Create the Recipe Database Schema
   a. In the Recipe Database, create a recipe named ProductRecipe.
   b. Add two elements to ProductRecipe and bind them to variables in the project. These variables must be declared in [Project] » [Variable] in advance.

Element Name	Data Type	Bind to Variable
SetTemp	INT	TempSetPoint
SetPressure	INT	PressureSetPoint

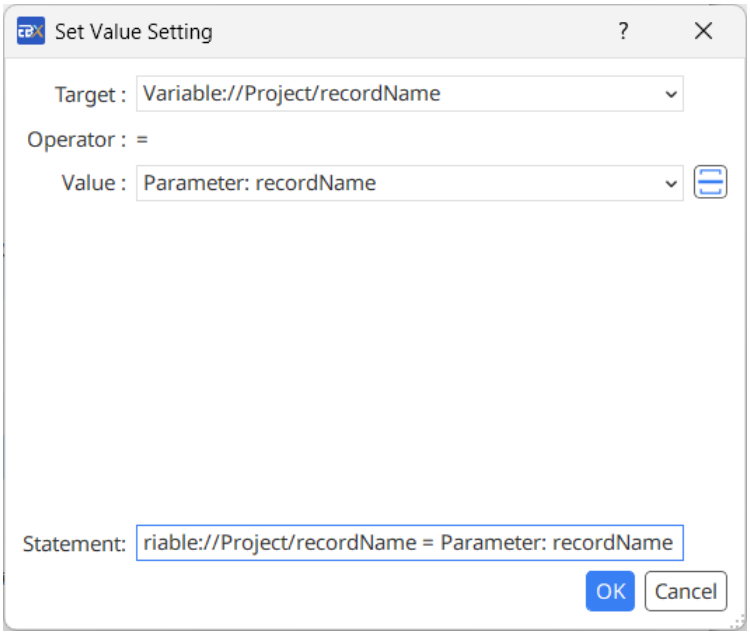
2. Enter Recipe Records
   a. Open the Record tab of ProductRecipe.
   b. Add two records, and enter the recipe identifier string in the [Record Name] field:

Record Name	SetTemp	SetPressure
ProductA	150	5
ProductB	220	8

3. Operate Using the Recipe View
   a. Place a Recipe View object on the screen.
   b. In the General properties, set Recipe to ProductRecipe.
   c. Clear the Read-Only option to allow operators to edit recipe parameters when needed.
   d. Set the event: Set the [Event] of the Recipe Viewer widget to [Selection changed]. Action setting: Add a [Set Value] action, then configure it in the dialog box that follows:

- Target: Click the menu button on the right. In the list, find and select the pre-declared string variable, for example, the recordName STRING variable under the Project directory.
- Value: Click the menu button on the right. In the pop-up window, switch to the [Parameter] tab, then select [recordName] from the list.

Description: When the user selects a record in the table , the name of that record is immediately written to the string variable recordName.



4. Place Widgets on Screen

a. Numeric: Place two Numeric widgets and bind them to TempSetPoint and PressureSetPoint. Also place a Text widget and bind it to the recordName variable.

b. Button: Place a Button widget and set its action to [Call object method] to call the [Apply to device] method of ProductRecipe. In the parameter settings on the right, set the source of [Record Name] to the previously saved string variable (Project/recordName).

5. Run the Project

a. The user can browse the parameters for Product A and Product B directly in the Recipe View.

b. When Product B is selected, the string variable recordName immediately receives and displays "Product B".

c. When the button is pressed (Apply to Device), the Recipe Database reads the values from the selected recipe (SetTemp = 220, SetPressure = 8) and writes them to the bound variables TempSetPoint and PressureSetPoint.

d. The PLC program can then monitor these variables to switch production parameters automatically.

e. To modify a recipe, the user can edit values directly in the Recipe View, and the Recipe Database will update the record accordingly.

15. Weincloud

15.1. Weincloud 15-2
15.2. EasyAccess 2.0 15-2
15.3. Dashboard 15-4

15.1. Weincloud

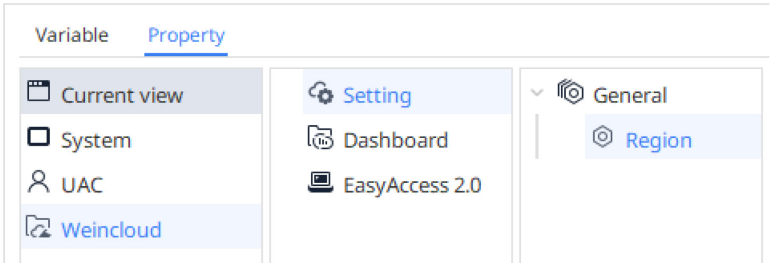
Weincloud is a cloud platform designed for industrial automation, enabling remote monitoring and management of industrial equipment. It provides a centralized solution for connecting to and controlling machines distributed across multiple locations.
With Weincloud, enterprises can implement smart manufacturing and IoT applications to improve production efficiency and reduce operating costs. Its core features include EasyAccess 2.0 and Dashboard. This chapter explains the Weincloud-related settings in EasyBuilder X.

15.1.1. Server Settings

Weincloud operates two independent servers: the Global server and the China server. Before using Weincloud platform services, configure the server location for the HMI project in [IIoT] » [Weincloud].

15.1.2. Weincloud Properties

Weincloud-related properties are global objects and can only be bound to read-only widgets.



As shown above, bind the Numeric widget's "Read Value" to Property » Weincloud » Setting » General » Region.

Property	Data Format	Description
General		
Region	USINT	Display the connected server location. 0: Global / 1: China

15.2. EasyAccess 2.0

This section provides the settings for EasyAccess 2.0 connection parameters and related information within the project file. It allows access to the connection status, connected user information, and configuration of properties required for adding to domain, such as the Session ID, Password, and proxy server.

15.2.1. EasyAccess 2.0 Properties

Property	Data Format	Description
General		
Status	UINT	Display the status of the server connection. 0: Disabled / 1: Connecting / 2: Online / 3: Occupied
Domain	STRING	Display the domain name of the HMI.
Connected user count	UINT	Display the number of active EasyAccess 2.0 connections to the HMI.
Connected users	STRING	Display information about the connected users.
Last connection result	USINT	Display the connection result error code. Refer to the error code list for details.
Session ID	STRING	The ID required when the HMI is activated and joins a domain.
Session password	STRING	The password required when the HMI is activated and joins a domain.
Push Notification URL	STRING	Display the URL used to subscribe to push notifications.
Proxy		
Enable	BOOL	Display whether the proxy server is enabled.
Type	UINT	Display the proxy server type. 0: HTTP / 1: SOCKSv4 / 2: SOCKSv5
IP Address	USINT	Display the proxy server IP address.
Port	UINT	Display the proxy server port number.

15.2.2. EasyAccess 2.0 Method

Enabling or disconnecting the EasyAccess 2.0 server connection, as well as using a proxy server, is performed by calling EasyAccess 2.0 object methods.

Method	Description
Connect	Enable the connection to the EasyAccess 2.0 server.
Disconnect	Disconnect from the EasyAccess 2.0 server.

Enable proxy	Preconfigure the following parameters, or bind them to variables, to enable the proxy server using the specified values. Type (USINT) IP Address (IPv4) Port (UINT) Authentication (BOOL) Username (STRING) Password (STRING)
Disable proxy	Disable the proxy server.

15.3. Dashboard

The Dashboard is one of the Weincloud cloud service features. Using the Dashboard in EasyBuilder X, internal HMI variables can be uploaded to Weincloud to enable remote monitoring and data visualization.

15.3.1. Configuration Steps

1. Click the [IIoT] tab on the menu bar.
2. Select Global or China based on the account region.
3. Click [Sync]. The Dashboard login window will appear.
4. In the login window, enter the Domain Name, Username, and Password.
5. After a successful login, select the target machine name from the drop-down list in the upper-left corner of the sync interface.
6. Click [OK] to load the tags into the EasyBuilder X project.
7. In the tag list, click the bind button of the Value setting and select the corresponding variable from the variable library.
   Note: Currently, binding is supported only for variables with Read/Write permissions.

15.3.2. Dashboard Properties

To configure the overall communication parameters of the Dashboard, select the root node named “Dashboard” in the tag tree. After the root node is selected, the property panel displays the available settings.

Property	Description
General	
Authorization state	Display the current authentication status.
Region	Display the region of the Weincloud server.
Domain name	Display the currently connected Weincloud domain name.
Keep alive time	Set the interval for checking and maintaining the server connection.
Buffer usage	Display the current communication buffer usage percentage.

15.3.3. Dashboard Event

Event	Description
Connection status changed	Triggered when the connection status between the Dashboard and the target server changes.

Connection Status Codes (UINT Output):

Code	Description
0	Disconnected
1	Connecting
2	Connected

Error Codes (UINT Output):

Code	Description
0	No error
2	Internal error
3000	Error – failed to connect to server
3003	Error – domain name error
4000	Authorization – failed
4001	Error – access denied
4003	Error – SSL certificate verification failed

100000	Unknown error
100100	Warning – start while running
100101	Warning – unmatched tag detected
100102	Error – not a simulation hardware key
100103	Error – not activated
100202	Error – illegal client ID
100205	Error – internal SSL error
100301	Authorization – unknown error
100302	Authorization – empty response
100303	Authorization – JSON parse error
100304	Authorization – missing JSON key
100305	Certificate – unknown error
100306	Certificate – directory error
100307	Certificate – empty response
100308	Certificate – JSON parse error
100309	Certificate – missing JSON key
100310	Device – unknown error
100311	Device – empty response
100312	Device – JSON parse error
100313	Device – invalid JSON format
100314	Device – missing JSON key
100315	Tag – unknown error
100316	Tag – empty response
100317	Tag – JSON parse error
100318	Tag – invalid JSON format
100319	Tag – missing JSON key
100400	Curl – unknown error

15.3.4. Dashboard Method

Method	Description
Connect	Establish a connection to the target Weincloud server.
Disconnect	Disconnect from the target Weincloud server.

15.3.5. Dashboard Tag Properties

Select an individual tag to configure its detailed settings.

Property	Description
General	
Name	Display the tag name.
Enabled	Select to enable data synchronization between this tag and Weincloud.
Data type	Display the data type defined for this tag.
Value	Set the content of this tag. A fixed value can be entered directly, or click bind button on the right to bind it to a corresponding variable in the project variable library to enable real-time synchronization between HMI data and the cloud.

16. Macro

16.1. Overview 16-2
16.2. Macro Editor 16-2
16.3. Configuration 16-3
16.4. Syntax 16-5
16.5. Statements 16-10
16.6. Subfunctions 16-16
16.7. Built-In Functions 16-18
16.8. How to Create and Execute Macros 16-22
16.9. User-Defined Function Library 16-23
16.10. Compilation Error Messages 16-24

16.1. Overview

Macro commands provide additional functionality beyond the standard application features. During HMI operation, macro commands can automatically execute various instructions, such as performing complex calculations, processing strings, and enabling data interaction between the user and the project file.

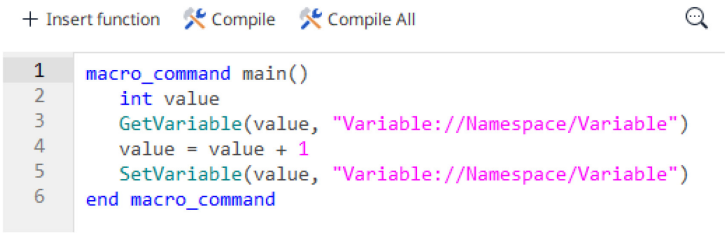
This chapter introduces the syntax, usage, and editing methods of macro commands, helping users quickly become familiar with the powerful macro functionality in EasyBuilder X.

16.2. Macro Editor

Open the Macro Editor by selecting [Project] » [Macro] . The editor provides the following features:

- Display line numbers

  When the Macro Editor is opened, line numbers are automatically displayed on the left side of the editing area.

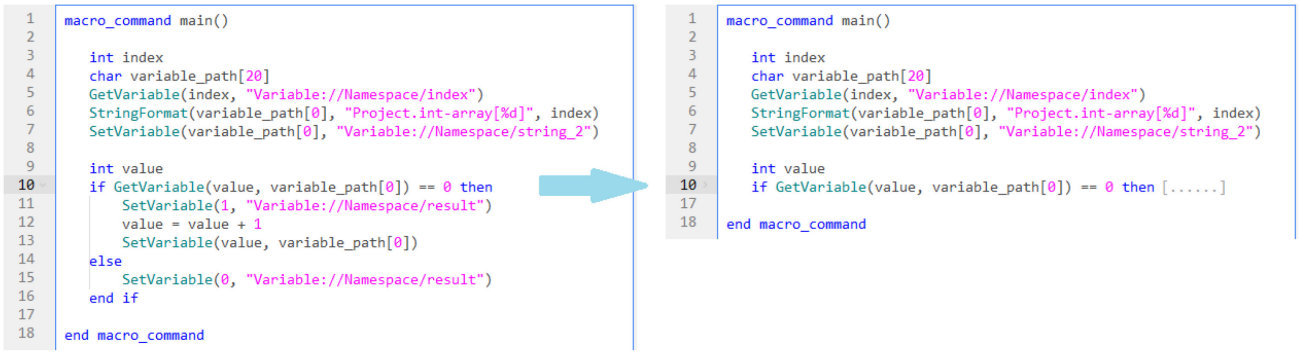


Right-click within the editing area to open the context menu, as shown in the figure below. Functions that are currently unavailable appear grayed out. For example, the Copy function becomes available only when text is selected; therefore, it is disabled when no text is selected. Keyboard shortcuts are also provided as indicated in the menu.

- Undo / Redo
- Cut / Copy / Paste / Delete
- Select All
- Toggle All Outlining

  The Macro Editor provides a code folding feature to help users navigate code more easily. Code folding allows the editor to hide sections of code that belong to the same block. Hidden code is displayed as […] in the editing area.

  A tree structure appears on the left side of the editor. Click the symbol next to the line number to collapse or expand a code block, as shown in the figure below.



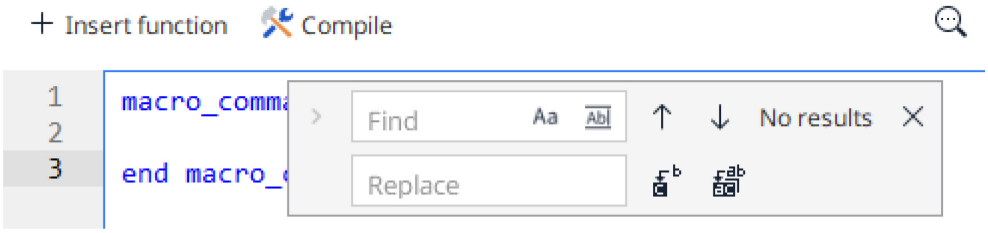
Code enclosed within specific keywords is referred to as a code block. The default code blocks include the following:

Loop statements: for – next / while - wend

Conditional statements: if – end if

Multi-branch statements: select case – end select

- The Macro Editor also provides Find / Replace functionality for searching and replacing text.



16.3. Configuration

Macros consist of various statements. These statements include constants, variables, and operators. The statements are arranged in a specific order so that, when executed, they produce the desired result.

The general structure of a macro is as follows:

```
Global Variable Declarations ---------------------------------- Optional

Sub Function Block Declarations ------------------------------- Optional
    Local Variable Declarations
End Sub

macro_command main() ------------------------------------------ Required
    Local Variable Declarations
    [Statements]
end macro_command ---------------------------------------------- Required
```

A macro must contain one and only one main function, which serves as the entry point for macro execution. The format is:

macro_command function_name()

end macro_command

Variable declarations must appear before macro statements; otherwise, the macro will fail to compile.
Local variables are typically used within the main macro function or within user-defined sub-functions. Their scope is limited to the function in which they are declared.
Global variables are declared before all macro functions and remain valid throughout the entire macro. If a local variable and a global variable share the same name, the local variable takes precedence.

The following example shows a simple macro that includes variable declarations and function calls. Double slashes // indicate comments; text following // is not executed.

```
macro_command main()
    int value                                          // local variable declaration
    GetVariable(value, "Variable://Namespace/Variable")       // function call
    value = value + 1
    SetVariable(value, "Variable://Namespace/Variable")      // function call
end macro_command
```

16.4. Syntax

16.4.1. Constants and Variables

16.4.1.1. Constants

A constant is a fixed value that can be used directly in statements. The supported formats are shown below:

Constant Type	Notes	Examples
Decimal integer		345, -234, 0, 23456
Hexadecimal	Must begin with 0x	0x3b, 0xffff, 0x237
Character / String	A single character must be enclosed in single quotation marks. A string must be enclosed in double quotation marks. To include a double quotation mark in a string, use \".	‘a’, "data", "name"
Boolean		true, false

The following example demonstrates the use of constants:

```
macro_command main()
    short A, B // Declare A and B as short integer variables
    A = 1234
    B = 0x12 // 1234 and 0x12 are constants
end macro_command
```

16.4.1.2. Variables

A variable is a name that represents a data value. In macros, the value of a variable can change as statements are executed.

Variable Naming Rules

- Must begin with a letter.
- The variable name can contain up to 60 characters.
- A single macro can declare up to 4 KB of variables.
- Reserved system keywords, such as bool and int, cannot be used as variable names.

The following variable types are available:

Variable Type	Description	Range
bool (Boolean)	1 bit	0, 1
char (Character)	8 bits (byte)	+127 to -128
short (Short Integer)	16 bits (word)	+32767 to -32768
int (Integer)	32 bits (double word)	+2147483647to -2147483648
float (Floating Point)	32 bits (double word)	
unsigned char	8 bits (byte)	0 to 255
unsigned short	16 bits (word)	0 to 65535
unsigned int	32 bits (double word)	0 to 4,294,967,295
long (Long Integer)	64 bits (four words)	+281474976710655 ~ -281474976710655
unsigned long (Long Integer)	64 bits (four words)	0 ~ 281474976710655
double (Double Precision Floating Point)	64 bits (four words)	

Variable Declarations

In macro commands, all variables must be declared before they are used. To declare a variable, specify the variable type followed by the variable name.

Example:

```
int             a
short           b, switch
float           pressure
unsigned short  c
```

Array Declarations

Macros support one-dimensional arrays (zero-based index). To declare an array variable, specify the type, the variable name, and the number of elements enclosed in brackets []. The array length can range from 1 to 4096. Each macro supports a maximum of 4096 variables.

Example:

```
int         a[10]
short       b[20], switch[30]
float       pressure[15]
```

The minimum array index is 0 and the maximum is (array size - 1).

Example:

```
char data [100]        //   array size is 100
```

In this case, the smallest index is 0 and largest index is 99 (100-1=99)

Variable and Array Initialization

Variables can be initialized in two ways:

- Using assignment statements (=)

Example:

```
int a
float b[3]
a = 10
b[0] = 1
```

- Assign values during declaration

```
char a = '5', b = 9
```

Array variable declaration is a special case. When initializing an entire array, the values can be placed inside braces { } in the array declaration, with each value separated by a comma. For string-related variables, double quotation marks can also be used to assign a value.

Examples:

```
float data[4] = {11, 22, 33, 44}    // now data[0] is 11, data[1] is 22....
char str[10] = "test";
```

Strings in macros support escape sequences (\) for displaying special characters. The supported escape sequences are listed below:

Basic Control Characters

Escape Sequence	Description	Note
\n	Line feed	LF
\r	Carriage return	CR
\t	Horizontal tab	Tab key
\v	Vertical tab	Tab key
\b	Delete the previous character	Backspace
\f	Form feed	
\a	Alert sound	

Quotes and Backslash

Escape Sequence	Description	Note
\\	Backslash \	“C:\\Windows”
\’	Single quotation mark '	“I\’m happy”
\”	Double quotation mark "	“He said \”Hello\””
\?	Question mark ?	“Question mark \?”

16.4.2. Operators

Operators specify how data is manipulated and calculated. In a statement, the value of the variable on the left side of the operator is determined by the expression on the right side.

Operator	Description	Example
=	Assignment operator	pressure = 10

Arithmetic Operators	Description	Example
+	Addition	A = B + C
-	Subtraction	A = B – C
*	Multiplication	A = B * C
/	Division	A = B / C
% or mod	Modulo (returns the remainder)	A = B % 5 or A = B mod 5

Note:

By default, integer values are treated as integer data types. When division is performed between two integers and the result contains a decimal part, the decimal portion is automatically truncated.

To preserve the decimal result, add .0 to either the dividend or divisor to convert the calculation to floating-point.

Examples:

A = 3 / 2 = 1 » 3 and 2 are integers, so the result is also an integer.

B = 3 / 2.0 = 1.5 » 3 is an integer and 2.0 is a floating-point number, so the result is a floating-point value.

C = 3.0 / 2 = 1.5 » 3.0 is a floating-point number and 2 is an integer, so the result is a floating-point value.

Comparison Operators	Description	Example
<	Less than	if A < 10 then B = 5
<=	Less than or equal to	if A <= 10 then B = 5
>	Greater than	if A > 10 then B = 5

>=	Greater than or equal to	if A >= 10 then B = 5
==	Equal to	if A == 10 then B = 5
<>	Not equal to	if A <> 10 then B = 5

Logical Operators	Description	Example
and	Logical AND	if A < 10 and B > 5 then C = 10
or	Logical OR	if A >= 10 or B > 5 then C = 10
xor	Logical Exclusive OR	if A xor 256 then B = 5
not	Logical NOT	if not A then B = 5

Shift and bitwise operators are used to manipulate the bits of character, short integer, and integer variables. Within a statement, these operators are evaluated from left to right, meaning the operator appearing on the left is executed first, followed sequentially by those on the right.

Shift Operators	Description	Example
<<	Shifts bits to the left by the specified number of positions	A = B << 8
>>	Shifts bits to the right by the specified number of positions	A = B >> 8

Bitwise Operators	Description	Example
&	Bitwise AND	A = B & 0xf
|	Bitwise OR	A = B | C
^	Bitwise XOR	A = B ^ C
~	Bitwise Note (one's complement)	A = ~B

Operator Precedence

The precedence of all operators described above, from highest to lowest, is as follows:

1. Operators within parentheses
2. Arithmetic operators
3. Shift and bitwise operators
4. Comparison operators
5. Logical operators
6. Assignment operators

Reserved Keywords

The keywords and function names mentioned above are reserved for macros. They cannot be used as variable names, array names, or function names.

16.5. Statements

16.5.1. Definition Statements

Definition statements are used to declare variables and arrays.

type	name

Defines a variable named “name” with type “type”.
Example:

```
int A          // Defines variable A as an integer
```

type	name[array length]

Defines an array variable named “name”, with size “array length”, and type “type”.
Example:

```
int B[10]      // Defines B as a one-dimensional integer array with length 10
```

16.5.2. Assignment Statements

Assignment statements use the assignment operator (=) to assign the result of an expression on the right side to the variable on the left side.
An expression is composed of variables, constants, and operators, and produces a value when evaluated.

VariableName	Expression

Example

```
A = 2          // Assigns the value 2 to variable A
```

16.5.3. Logical Statements

Logical statements perform actions based on the result of a Boolean expression. The syntax is shown below.

Single-Line Format

```
If <Condition> then
   [Statements]
else
   [Statements]
end if
```

Example:

```
if a == 2 then
      b = 1
else
      b = 2
end if
```

Block Format

```
If <Condition> then
   [Statements]
else if <Condition-n> then
[Statements]
else
   [Statements]
end if
```

Example:

```
if a == 2 then
      b = 1
else if a == 3 then
      b = 2
else
      b = 3
end if
```

Syntax Description	
if	Must appear at the beginning of the statement.
<Condition>	Required. This is the controlling expression. If the value of <Condition> is 0, it is considered FALSE; if it is non-zero, it is considered TRUE.
then	Placed before the statements to be executed when <Condition> evaluates to TRUE.
[Statements]	Optional in block format, but required in single-line format when no else statement is present. The statements are executed when <Condition> is TRUE.
else if	Optional. One or more else if statements may be used. The statements are executed when the corresponding <Condition-n> evaluates to TRUE.
<Condition-n>	Optional. Same definition as <Condition>.
else	Optional. Executed when all previous conditions evaluate to FALSE.
end if	Required. Used to terminate the if-then statement.

16.5.4. Select-Case Statements

The select-case statement is used to perform multiple conditional evaluations. Its function is similar to an if-else statement. Based on the value of the specified variable, the program executes the statements under the matching case. Execution continues until a break statement is encountered, after which control moves to end select.
The syntax structure is shown below.

Format without a Default Case

```
Select Case [variable]
Case [value]
    [Statements]
    break
end Select
```

Example:

```
Select Case A
    Case 1
        b=1
    break
end Select
```

Format with a Default Case (Case else)

```
Select Case [variable]
Case [value]
    [Statements]
    break
    Case else
    [Statements]
    break
end Select
```

Example:

```
Select Case A
    Case 1
        b=1
    break
    Case else
        b=0
    break
end Select
```

Multiple cases in the same block

```
Select Case [variable]
Case [value1]
    [Statements]
Case [value2]
    [Statements]
    break

end Select
```

Example:

```
Select Case A
    Case 1
        break
    Case 2
        b=2
        break
    Case 3
```

```
            b=3
        break
    end Select
```

Syntax Description	
Select Case	Must appear at the beginning of the statement.
[variable]	Required. The value of this variable is compared with each case.
Case else	Optional. Represents the default case. If the value of [variable] does not match any case, the statements under Case else are executed. If no default case is defined, execution exits the select-case structure when no match is found.
break	Optional. When a matching case is found, the statements under that case are executed until a break statement is reached, then control jumps to end select. If break is omitted, execution continues through subsequent statements until end select is reached.
end Select	Marks the end of the select-case statement.

16.5.5. Loop Statements

Loop statements repeatedly execute a task based on a specified condition. Two types of loop statements are supported.

16.5.5.1. for-next Statements

The for-next statement is typically used when the number of loop iterations is fixed. A variable is used as a counter to control the number of iterations and determine when the loop ends.

```
for [Counter] = <StartValue> to <EndValue> [step <StepValue>]
    [Statements]
next [Counter]
```

Or

```
for [Counter] = <StartValue> down <EndValue> [step <StepValue>]
    [Statements]
next [Counter]
```

Example:

```
  for a = 0 to 10 step 2
      b = a
  next a
```

Syntax Description	
for	Must appear at the beginning of the statement.
[Counter]	Required. A numeric variable used as the loop counter to control the number of iterations.
<StartValue>	Required. The initial value assigned to the counter.
to/down	Required. Determines whether the counter increases or decreases. “to” increments the counter by <StepValue>, while “down” decrements the counter by <StepValue>.
<EndValue>	Required. The termination value used to test whether the loop should end.
step	Optional. Specifies that a step value other than 1 will be used.
[StepValue]	Optional. The increment or decrement value of the counter. If omitted, the default value is 1.
[Statements]	Optional. The block of statements executed during each iteration of the loop.
next	Required. Marks the end of the for-next statement.
[Counter]	Optional.

16.5.5.2. while-wend Statements

The while-wend statement is used to execute a loop when the number of iterations is not predetermined. A condition is used to determine when the loop should end. When the condition evaluates to TRUE, the statements inside the loop are executed repeatedly until the condition becomes FALSE.

```
while <Condition>
        [Statements]
wend
```

Example:

```
while a < 10
        a = a + 10
wend
```

Syntax Description	
while	Required. Marks the beginning of the loop statement.
<Condition>	Required. The controlling expression. When the condition evaluates to TRUE, the loop continues; when it evaluates to FALSE, the loop terminates.
wend	Indicates the end of the while-wend statement.

16.5.5.3. Other Control Commands

break	Used in for-next and while-wend statements. When encountered, execution immediately exits the loop and continues with the statement following the loop.
continue	Used in for-next and while-wend statements. When encountered, the current iteration ends immediately and the next iteration begins.
return	Can be used to return a value in a user-defined function. When used in the main function, it forces the macro to exit the main function immediately.

16.6. Subfunctions

Subfunctions can help reduce repetitive code in macros. A subfunction must be defined before it is used and can contain any type of variables and statements.
In the main function, a subfunction can be called by placing its name followed by parameters in parentheses. After execution, the subfunction returns the result to the calling statement in the main function, where it can be used in an assignment statement or condition.
A subfunction may or may not return a value, and the parameter list may be empty. When calling a subfunction, the call must follow the format defined in its declaration.

Subfunction with Return Value

```
sub type <name> [(parameters)]
    Local variable declarations
    [Statements]
    [return [value]]
end sub
```

Example:

```
sub int Add(int x, int y)
    int result
    result = x + y
    return result
end sub

macro_command main()
    int a = 10, b = 20, sum
    sum = Add(a, b)
end macro_command
```

or:

```
sub int Add()
        int result, x=10, y=20
        result = x + y
        return result
  end sub

macro_command main()
        int sum
        sum = Add()
end macro_command
```

Subfunction without Return Value

```
sub <name> [(parameters)]
        Local variable declarations
        [Statements]
end sub
```

Example:

```
sub Add(int x, int y)
        int result
        result = x + y
  end sub

macro_command main()
        int a = 10, b = 20
        Add(a, b)
end macro_command
```

or:

```
sub Add()
        int result, x=10, y=20
        result = x + y
end sub

macro_command main()
        Add()
end macro_command
```

Syntax Description	
sub	Required. Marks the beginning of a subfunction definition.
type	Optional. Specifies the return data type of the subfunction. A subfunction does not have to return a value.
(parameters)	Optional. These parameters receive values passed from the main function. The data types of the passed parameters must match the types defined in the parameter declarations. For example: sub int MyFunction(int x, int y) Here, x and y must be integer values passed from the main function. The function can be called as follows: ret = MyFunction(456, pressure) In this example, pressure must also be an integer to match the parameter type defined in the function declaration. Note that the arguments in the function call may be constants or variables. After the function is executed, an integer value is returned and assigned to the variable “ret”.
Local variable declaration	Variables used inside the subfunction must be declared before use, in addition to the parameters passed to the function. In the example above, **x** and **y** are variables that can be used within the subfunction. Global variables can also be used in a subfunction.
[Statements]	The statements executed within the subfunction.
[return [value]]	Optional. Returns a value to the calling statement and ends the subfunction execution. If a return type is defined in the subfunction declaration, the return statement must be included.
end sub	Required. Marks the end of the subfunction.

16.7. Built-In Functions

The EasyBuilder X macro system provides several built-in functions for tasks such as reading data from devices, writing data to devices, data processing, and mathematical calculations.

16.7.1. Function List

This document provides only a brief description of each function. For detailed usage instructions, refer to the function descriptions in the Macro Editor. Click [Insert Function] to view detailed information about each built-in function.

Function	Description
Variable Functions	
GetVariable	Reads a single variable value from the HMI into a local variable.
GetVariables	Reads multiple variable values from the HMI into local variables.

SetVariable	Writes a single value to an HMI variable.
SetVariables	Writes multiple values to HMI variables.
Flow Control Functions	
ASYNC_TRIG_MACRO	Triggers the specified macro and executes it asynchronously.
DELAY	Pauses macro execution for a specified period of time.
SYNC_TRIG_MACRO	Triggers the specified macro and waits until it finishes before continuing execution.
Data Operation Functions	
FILL	Fills data into an array.
GETBIT	Reads the specified bit of a value.
HIBYTE	Retrieves the high byte of a 16-bit value.
HIWORD	Retrieves the high word of a 32-bit value.
INVBIT	Inverts the specified bit of a value.
LOBYTE	Retrieves the low byte of a 16-bit value.
LOWORD	Retrieves the low word of a 32-bit value.
SETBITOFF	Sets the specified bit to 0.
SETBITON	Sets the specified bit to 1.
SWAPB	Swaps the high and low bytes of a 16-bit value.
SWAPW	Swaps the high and low words of a 32-bit value.
Data Type Conversion Functions	
ASCII2DEC	Converts ASCII character data to decimal format.
ASCII2DOUBLE	Converts ASCII character data to a double-precision floating-point value.
ASCII2FLOAT	Converts ASCII character data to a floating-point value.
ASCII2HEX	Converts ASCII character data to hexadecimal format.
BCD2BIN	Converts BCD format data to binary format.
BIN2BCD	Converts binary format data to BCD format.
DATE2ASCII	Converts the current date to ASCII format.
DATE2DEC	Converts the current date to decimal format.
DEC2ASCII	Converts decimal data to ASCII format.
DOUBLE2ASCII	Converts double-precision floating-point data to ASCII format.
FLOAT2ASCII	Converts floating-point data to ASCII format.
HEX2ASCII	Converts hexadecimal data to ASCII format.
StringBin2DecAsc	Converts an integer to a decimal string.
StringBin2HexAsc	Converts an integer to a hexadecimal string.
StringDecAsc2Bin	Converts a decimal string to an integer.
StringDecAsc2Float	Converts a decimal string to a floating-point number.

StringFloat2DecAsc	Converts a floating-point number to a decimal string.
StringHexAsc2Bin	Converts a hexadecimal string to an integer.
String Processing Functions	
String2Unicode	Converts a string to a Unicode string.
StringCat	Concatenates two strings.
StringCompare	Compares two strings (case-sensitive).
StringCompareNoCase	Compares two strings (case-insensitive).
StringCopy	Copies a string.
StringExcluding	Extracts characters from the beginning of a string until a specified character is encountered.
StringFind	Searches for a substring within a string.
StringFindOneOf	Finds the first occurrence of any character from one string in another string.
StringFormat	Formats a string with dynamic parameters and stores the result in a local variable.
StringIncluding	Extracts characters from the beginning of a string including a specified character.
StringInsert	Inserts a string into another string at a specified position.
StringLength	Returns the length of a string.
StringMD5	Generates an MD5 message digest string.
StringMid	Extracts a substring from a string.
StringReverseFind	Searches for a substring within a string starting from the end.
StringToLower	Converts all characters in a string to lowercase.
StringToReverse	Reverses a string.
StringToUpper	Converts all characters in a string to uppercase.
StringTrimLeft	Trims specified characters from the beginning of a string.
StringTrimRight	Trims specified characters from the end of a string.
Unicode2Utf8	Converts a Unicode string to UTF-8 format.
UnicodeCat	Concatenates two Unicode strings.
UnicodeCompare	Compares two Unicode strings (case-sensitive).
UnicodeCopy	Copies a Unicode string.
UnicodeExcluding	Extracts characters from the beginning of a Unicode string until a specified character is encountered.
UnicodeFormat	Formats a Unicode string with dynamic parameters and stores the result in a local variable.
UnicodeLength	Returns the length of a Unicode string.
Utf82Unicode	Converts a UTF-8 string to a Unicode string.

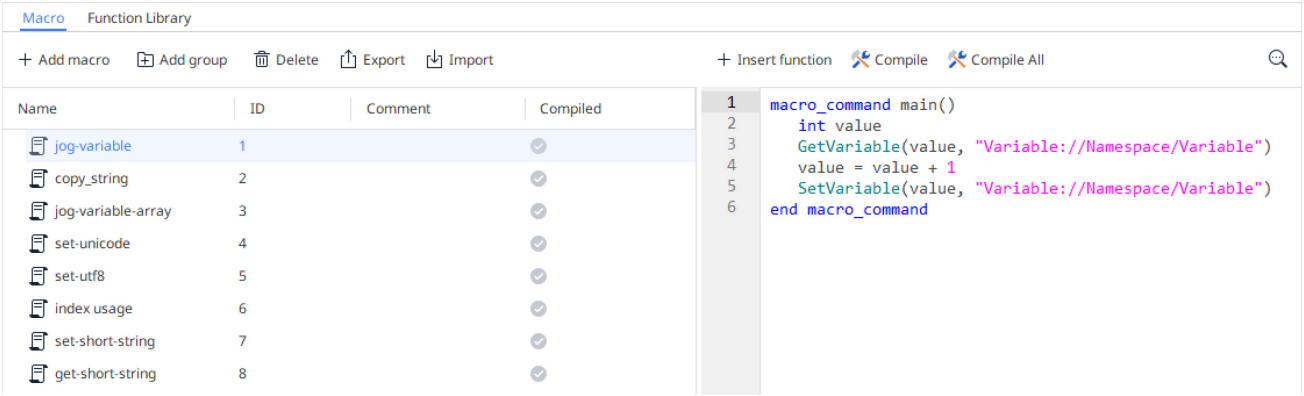
Mathematical Functions	
ACOS	Calculates the arccosine.
ASIN	Calculates the arcsine.
ATAN	Calculates the arctangent.
CEIL	Returns the smallest integer greater than or equal to the value.
COS	Calculates the cosine.
COT	Calculates the cotangent.
CSC	Calculates the cosecant.
CUBERT	Calculates the cube root.
FLOOR	Returns the largest integer less than or equal to the value.
LOG	Returns the natural logarithm.
LOG10	Returns the base-10 logarithm.
POW	Performs exponential calculation.
RAND	Generates a random number.
ROUND	Rounds to the nearest integer.
SEC	Calculates the secant.
SIN	Calculates the sine.
SQRT	Calculates the square root.
TAN	Calculates the tangent.
Statistical Functions	
AVERAGE	Calculates the average value of an array.
HARMEAN	Calculates the harmonic mean of an array.
MAX	Returns the maximum value in an array.
MEDIAN	Returns the median value in an array.
MIN	Returns the minimum value in an array.
STDEVP	Calculates the population standard deviation.
STDEVS	Calculates the sample standard deviation.
Checksum Functions	
ADDSUM	Calculates the checksum by summing elements in a one-dimensional array.
BCC	Calculates the checksum using XOR on a one-dimensional array.
CRC	Calculates a 16-bit CRC checksum for a one-dimensional array.
CRC16_CCITT	Calculates a CRC16_CCITT checksum for a one-dimensional array.
CRC16_CCITT_FALSE	Calculates a CRC16_CCITT_FALSE checksum for a one-dimensional array.
CRC16_X25	Calculates a CRC16_X25 checksum for a one-dimensional array.

CRC16_XMODEM	Calculates a CRC16_XMODEM checksum for a one-dimensional array.
CRC8	Calculates an 8-bit CRC checksum for a one-dimensional array.
XORSUM	Calculates a checksum using XOR operations on a one-dimensional array.
Other Functions	
TRACE	Monitors and prints variable values in a macro for debugging purposes.

16.8. How to Create and Execute Macros

16.8.1. Creating a Macro

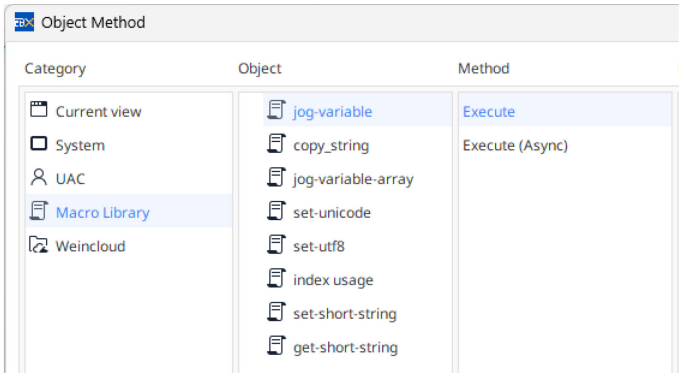
Click the Macro icon  in [Project] to open the Macro Library.



Setting	Description
Add macro	Create a new macro.
Add group	Create a group for organizing macros.
Delete	Delete the selected macro.
Export	Save the selected macro as a *.ebm file.
Import	Import a *.ebm file into the current project.
Insert function	Insert an available built-in function while editing a macro in the right-side editor.
Compile	Compile the macros.

16.8.2. Executing Macros

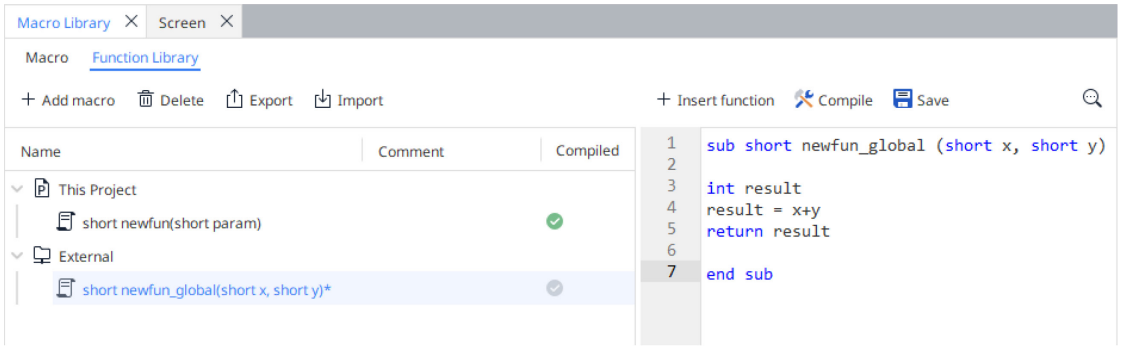
To execute a macro, use the Call Object Method action. A macro can be triggered from any action using either synchronous or asynchronous execution.



16.9. User-Defined Function Library

When using the Macro Editor, you can search for required functions within the built-in library to minimize definition time. However, if there are specific, frequently used functions that are not available in the built-in library, you can define and save your own custom functions. When you need to define the same function again, you can simply call up the saved function from the Function Library, making the editing process much more efficient. Furthermore, the Function Library significantly enhances the portability of user-defined functions.
Before creating a new function, it is recommended to check whether the required function already exists in the built-in function library or the online function library.

16.9.1. Function Library

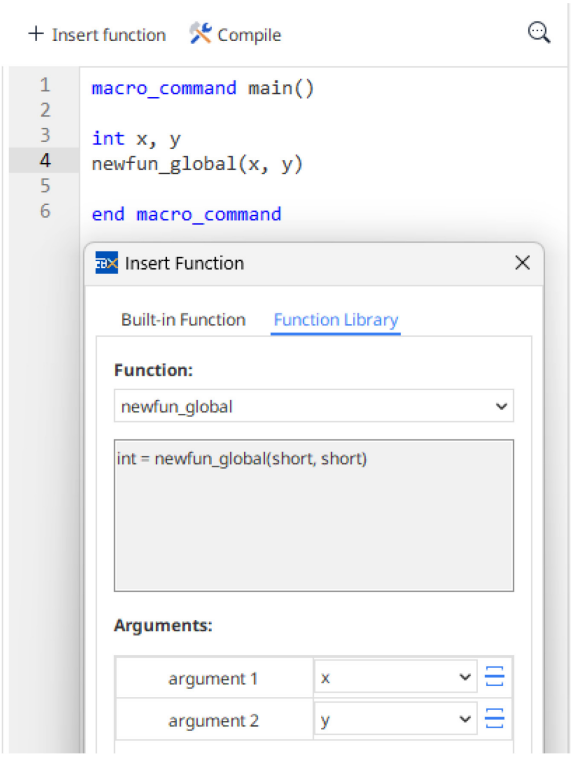


Setting	Description
Add macro	Create a new function.
Delete	Delete the selected function.
Export	Save the selected functions as an *.mlb file.
Import	Import an *.mlb file into the current project or the external function library.
Insert function	Insert an available function while editing a macro in the editor.
Compile	Compile the macros.

The Function Library has two file sources. One is This Project, which stores functions in the current project file (*.ebxprj*) and makes them available only to the current project. The other is External, which allows the functions to be used by all projects on the same computer.

16.9.2. Using the Function Library in Macros

In the Macro Editor, select Insert Function to choose a user-defined function from the Function Library and insert it directly into the macro.



16.10. Compilation Error Messages

- Compilation errors are displayed in the following format:

Error Message	Description
syntax_error	Syntax error.
array_data_exceeds_size	Data length exceeds the array size.
function_redefinition	Function redefined.
variable_redefinition	Variable redefined.
identifier_undeclared	Variable used without declaration.
undefined_data_type	Undefined data type.
parameter_type_error	Incorrect data type used for function parameter.
parameter_size_error	Incorrect number of parameters used in function call.
illegal_break_statement	Invalid break statement. break can only be used in for or while statements.
illegal_continue_statement	Invalid continue statement. continue can

	only be used in for or while statements.
macro_not_found	Macro not found.
float_var_cannot_contain_in_calcuation	Floating-point operators are not allowed in this expression.
unary_expression_error	Invalid unary expression.
function_not_found	Function not found.
expected_const_express	A constant expression is required.
function_return_error	This function should not return a value.
function_no_return	This function must return a value.
exceeds_array_size	Array index exceeds the defined array size.
parameter_must_be_array	The parameter must be defined as an array.
value_out_of_range	Variable value exceeds the valid range of its data type.
main_declare_error	Global variables cannot be declared in the function library.
cannot_declare_another_sub	Only one subfunction can be declared.
sub_cyclic_dependency	Cyclic dependency detected among subfunctions.
can_not_declare_global_var_in_sub	Global variables cannot be declared inside the function library.
invalid_operand_type	Bitwise, shift, and % operators require integer operands.
macro_command_entry_not_found	The entry function macro_command main() was not found.

- Notes on Using Macros

1. A single EasyBuilder X project can contain up to 500 macros and 500 function library entries.
2. Macros may cause the HMI to stop responding under certain conditions, such as:
   - An incorrect infinite loop in a macro
   - An array size exceeding the macro variable capacity

Device communication speed may affect macro execution speed. Conversely, excessive macro usage may slow down communication with the device.

17. Picture Library

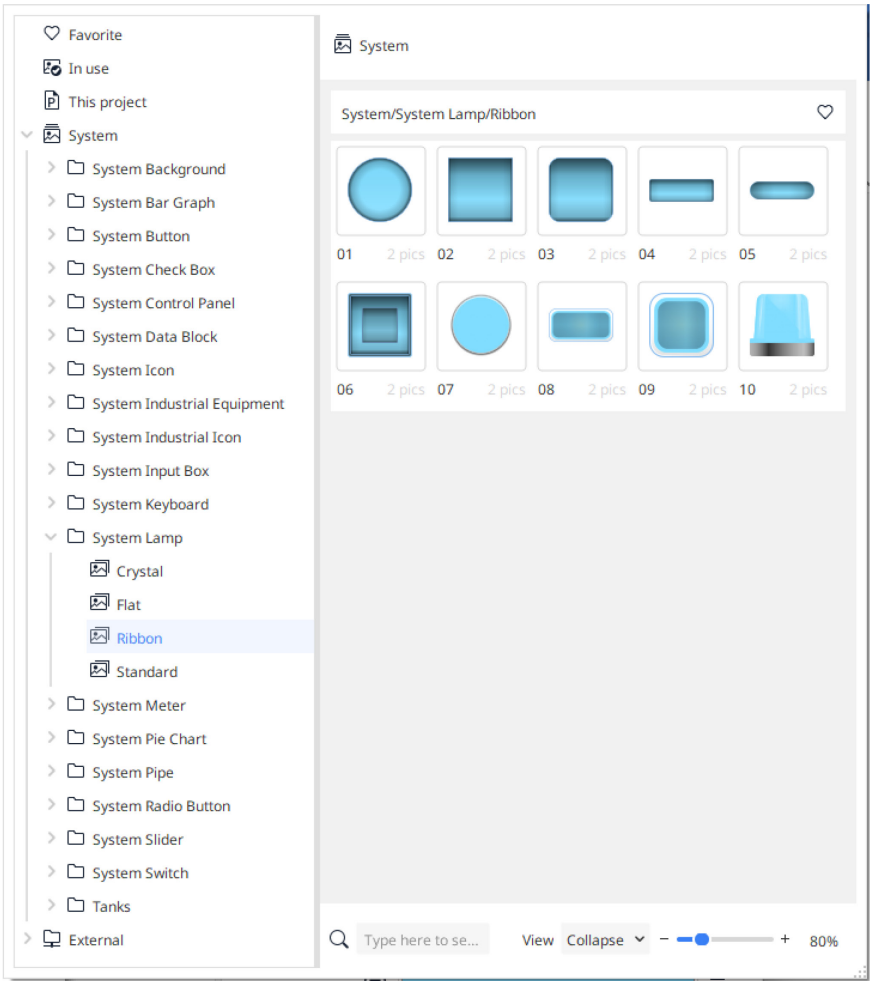
17.1. Overview 17-2
17.2. Picture Selection Menu 17-2
17.3. Managing Search Paths for External Picture Libraries 17-3
17.4. Managing the Project Picture Libraries 17-4
17.5. Picture Panel 17-5

17.1. Overview

The Picture Library is the main tool for managing and selecting pictures used in a project. It provides access to built-in system pictures, project pictures, and custom external picture library resources.

17.2. Picture Selection Menu

When editing widget properties, click [Outline] » [Picture] to open the picture selection menu. The interface includes a category tree on the left and a picture display area on the right.



17.2.1. Category Tree

In [File] » [Preferences] » [Picture Library], set the picture library categories to display.
The category tree contains the following categories, which define the source of each picture:

Category	Description
Favorite	Displays picture libraries marked as favorites.

In Use	Lists pictures currently referenced by Widgets in the project for quick reuse.
This Project	Displays pictures embedded in the current project.
System	Displays built-in system picture resources.
External	Displays external picture library files (.flbx) stored in the configured search paths.

17.2.2. Search and Filter

The picture selection menu includes a search function for locating pictures by keyword.

17.2.3. View Modes

The picture selection menu provides two view modes:

View	Description
Collapse	If a picture contains multiple states, they are grouped into one item, with the total number of included pictures shown beside it.
Expand	Displays all picture states individually for easier review.

17.3. Managing Search Paths for External Picture Libraries

For resources under the External category, the system uses a flexible path-recording mechanism to support project management and team collaboration.
The source and display behavior of picture libraries can be configured in [File] » [Preferences] » [Picture Library].

17.3.1. Picture Library Search Paths

This section is used to manage multiple picture library search directories.
To add a search path, follow these steps:

1. Click the browse button next to the path field and select the target folder.
2. Restart EasyBuilder X. After the path is changed, the software must be restarted to detect and load external picture library files (.flbx).

Recursive Search

The system automatically searches subfolders. For example, if the search path is set to C:\EBX_Resources, the system will also search all subfolders under it, such as C:\EBX_Resources\Library\Buttons, for picture library files.

17.3.2. Path Recording and Auto-Repair Mechanism

To support folder relocation and development across different computers, the system records

an external picture library path in two parts: the **search path** and the **relative path**.

Path Recording Example
Assume the configured search path is:
C:\PictureLibraries
The actual picture library file is stored at:
C:\PictureLibraries\Button\CNC_Lathe.flbx
The system records the path as:
[Search Path] + \Button\CNC_Lathe.flbx

Auto-Repair Process
When the picture library folder is moved, such as from a local drive to a network drive, only the search path needs to be updated. There is no need to reconfigure the Widgets.

Example
Suppose the entire folder C:\PictureLibraries is moved to Z:\NetworkShare\PictureLibraries.
Operation: In Preferences, change the original search path from C:\PictureLibraries to Z:\NetworkShare\PictureLibraries.
Result: When the project is opened, if the file cannot be found in the original path, the system searches the new search path for the same relative path, \Button\CNC_Lathe.flbx, and automatically restores the link.

17.3.3. Picture Library Categories

Select which categories are displayed in the picture menu, and use the up and down arrow buttons to adjust their order in the tree structure.
(Default order: Favorite → In Use → This Project → System → External)

17.4. Managing the Project Picture Libraries

Go to [Project] » [Pictures] to open the Picture Library management interface.

17.4.1. This Project

Provides a management interface for creating, editing, and deleting pictures stored in the project file.
The management interface also supports the system clipboard, which greatly simplifies picture import.
Operation: Copy image content from external graphics software, such as Photoshop or Paint, or from a screen capture tool, then press Ctrl+V in the management interface. The pasted image

is automatically added as a new state of the current picture item.

17.4.2. In Use

Lists the names, source paths, and usage counts of pictures currently in use.
If the picture source is external, click [Copy to Project File] to save the picture in the project file, preventing display issues if the external file is lost.

17.5. Picture Panel

Enable [View] » [Panel] » [Picture] to open the Picture Panel. This panel provides a docked workspace for quickly browsing pictures and applying them to the project or Widgets by drag and drop.

17.5.1. Interface Functions

Category switching: Use the drop-down list at the top of the panel to switch between picture categories.
Change system picture color: When the System category is selected, supported pictures can be recolored directly in the panel.
View mode: Supports switching between Collapsed and Expanded modes.

17.5.2. Drag and Drop Behavior

Pictures in the Picture panel can be dragged directly into the Picture Library or to a widget's Picture property.

Drag-and-Drop Target	Picture Panel View	Result
Picture Library management interface	**Expand**	A single picture is added as a new state picture in the target picture library.
Picture Library management interface	**Collapse**	All state pictures of the item are added to the target picture library.
Widget property	**Expand or Collapse**	The widget's original picture setting is replaced. The Picture Color property is also replaced if a color setting already exists in the panel.

18. cMT Viewer

18.1. cMT Remote 18-2
18.2. cMT Remote Interface 18-2

18.1. cMT Remote

cMT Remote is a tool designed for remote connection from a PC to cMT X series HMIs and for operating the project files running on them. It provides basic cMT Remote client management and settings, which are described in this chapter.

Limit: cMT Remote allows **up to 4 HMI connections at the same time**.

18.1.1. Opening cMT Remote

cMT Remote can be opened in either of the following ways:

- **From the EasyBuilder X editing interface: [Tools] » [cMT Remote]**
- **From the EasyBuilder X installation directory: cMTRemote.exe**

18.2. cMT Remote Interface

After opening cMT Remote, click the  icon in the upper-right corner of the interface to add a cMT X series HMI for connection. The HMI can be searched by name or IP address. Previously connected HMIs can also be viewed on the History tab.

18.2.1. cMT Remote Settings

Click the  icon in the upper-right corner of the cMT Remote interface to open the settings.

Item	Description
Information	
App. version	Display the current cMT Remote version.
General	
Change admin password	Required before accessing [more settings].
Restore previous connections	Set whether previously connected HMI project files are restored after cMT Remote is restarted.
Touch sounds	Set whether touch sound is enabled.
Display datetime bar	Set whether date and time information is displayed at the top of the cMT Remote window.
Scale to window size	Display the project file content by automatically fitting it to the window size.
No scaling (1:1)	Display the project file content at the HMI's original resolution. Content larger than the cMT

	Remote window must be viewed by using the scroll bars.
Show Start button	Set whether the Start button is displayed in the cMT Remote interface. If hidden, a corner position must be set. Press and hold that corner for 1.5 seconds to access the related settings again.
Language	
Language	Set the language of the cMT Remote interface.
Password Protection	
Change superuser password	In addition to the administrator password, the superuser password can be used to access protected functions.
Open project	If protection is enabled, the administrator or superuser password must be entered before connecting to the HMI.
Close project / Adjust placement	If protection is enabled, the administrator or superuser password must be entered before disconnecting from the HMI.
Server settings	If protection is enabled, the administrator or superuser password must be entered before accessing the HMI web server settings.
Clear history	If protection is enabled, the administrator or superuser password must be entered before clearing connection history.

To unlock password-protected functions, click the  icon in the upper-left corner of the cMT Remote interface and enter the administrator or superuser password. After unlocking, cMT Remote starts a 10-minute countdown before the functions are locked again. Any operation performed in cMT Remote during the countdown period will reset the countdown timer. To lock the functions again immediately, click the  icon to lock them manually.

19. Modbus TCP/IP Server

19.1. Modbus Server Settings 19-2
19.2. Mapping Table 19-2
19.3. Address Base 19-4
19.4. Enable In-Memory Data 19-4

19.1. Modbus Server Settings

Before configuring detailed Modbus Server parameters, ensure that the server has been added according to the steps in Chapter 4.

1. Click [Project] » [Server List].
2. In the list, click the created Modbus TCP/IP Server.

19.2. Mapping Table

The mapping table is used to bind project variables to the starting addresses of Modbus registers.

1. In the Modbus Server's Property panel, under Server Settings, click the settings icon next to [Mapping table] to open the mapping table editor, where mapping entries can be added, deleted, or modified.
2. In the mapping table editor, a valid Address must be set before a variable can be selected in the Variable field. If no address is set, the Variable field remains locked.
3. After the address is set, double-click the Address field then click the information icon ⓘ to open the browser. This tool displays the complete Modbus address list assigned to each member or element after a structure or array is expanded.

19.2.1. Address Types and Variable Limits

Address Type	Prefix	Supported Function Code	Supported Variable Type	Min. Unit
Holding Register	4x	0x03, 0x06, 0x10	16-bit or larger types (INT, DINT, REAL, etc.)	16-bit
Input Register	3x	0x04	16-bit or larger types (INT, DINT, REAL, etc.)	16-bit
Coil	0x	0x01, 0x05, 0x0F	BOOL only	1-bit
Discrete Input	1x	0x02	BOOL only	1-bit

19.2.2. Notes and Conflict Rules

1. Variables bound to 4x or 0x must have Read/Write permission.

2. Variables bound to 3x or 1x can have either Read/Write or Read Only permission.
3. Variables set to Write Only cannot be bound in the mapping table.
4. 8-bit types are not supported: Registers (4x, 3x) do not support 8-bit types such as SINT or USINT.
5. Rules for Arrays and Structures:
   Array: Must be bound as a complete block. The starting address corresponds to index 0. Total length = (number of elements) × (size of each element).
   Structure: Must be bound as an entire structure. Only the structure name is shown in the selection menu, and individual members cannot be selected separately. In addition, if the structure contains members smaller than 16-bit, such as SINT, USINT, or BOOL, the entire structure cannot be bound to 4x or 3x registers.
6. Overlapping address ranges for different mapping entries are strictly prohibited, as this will cause compilation errors in the project file.

19.2.3. Mapping Examples (Address Browser View)

The following examples show how the address browser displays detailed layouts when complex data types are mapped to Modbus addresses.

Example A: Array Type (INT[4] mapped to 4x11)

Name	Type	Address
INT[4]	INT[4]	[Server] 4x-11
└ [0]	INT	[Server] 4x-11
└ [1]	INT	[Server] 4x-12
└ [2]	INT	[Server] 4x-13
└ [3]	INT	[Server] 4x-14

Example B: Nested User-Defined Data Type (DataType mapped to 4x1)

This example shows address allocation when one DataType contains another DataType_2.

Name	Type	Address
DATATYPE	DataType	[Server] 4x-1
└ member	DataType_2	[Server] 4x-1
└ member	DINT	[Server] 4x-1 ~ 4x-2
└ member_2	INT	[Server] 4x-3
└ member_3	INT	[Server] 4x-4
member_4	DINT	[Server] 4x-5 ~ 4x-6

19.3. Address Base

Address Base controls how server addresses are displayed in the interface.

Option	Type	Address
Based on 1	4x1	Address numbering starts from 1.

19.4. Enable In-Memory Data

Server handling of valid address ranges without mapped variables.

Status	Behavior
Enabled	All valid addresses can be accessed. Addresses without mapped variables are read from or written to internal memory, and client requests always succeed.
Disabled	Only addresses defined in the mapping table can be accessed. If the client tries to read or write an unmapped area, the server returns error code 0x02 (Illegal Data Address).

Behavior Examples

Scenario: If variable A (REAL, 32-bit) is bound to starting address 4x1, it actually occupies two consecutive addresses: 4x1 and 4x2.

Case 1: External client reads address 4x1 or 4x2

Regardless of whether [Enable In-memory data] is enabled or disabled, the client can successfully send the read request and obtain the raw data.

Note: EasyBuilder X uses little-endian format. If the client reads only a single address, such as requesting only 4x1 with a length of 1, the returned value is the low-order word (LSW) of the 32-bit variable. Because the data is incomplete, it cannot be correctly interpreted as the original REAL type. To obtain the correct value, the client must request the full variable length, for example, starting from 4x1 with a length of 2.

Case 2: External client writes to the address

For variables that span multiple registers, the write operation must cover the entire variable.

Valid write: Start from address 4x1 and write length 2.

Invalid write: If the client tries to write from a middle address, such as writing only to 4x2, the server treats it as an illegal operation and returns exception code 0x02 (Illegal Data Address).

Case 3: External client reads address 4x3 (unmapped address)

When Enabled: The server returns data from internal buffer memory (default value: 0), and communication succeeds.

When Disabled: The server rejects the access and returns Modbus exception code 0x02 to the client, and communication fails.

20. Audit Trail

20.1. Overview 20-2
20.2. Settings 20-2
20.3. Usage 20-4
20.4. Audit Trail Viewer 20-7
20.5. Backup and Verification 20-9

20.1. Overview

Audit Trail is an operation management mechanism in EasyBuilder X. With proper configuration, it can require user confirmation before actions are performed on variables, recipe databases, and other functions, helping reduce the risk of unintended operations. It also records all operations for later review and audit traceability.

20.2. Settings

20.2.1. Audit Trail Properties

The Audit Trail properties are used to enable Audit Trail, configure logging, and display basic status and error code information.

Property	Description
General	
Enabled	Set whether Audit Trail is enabled.
Auto start logging	Set whether logging is enabled by default.
Runtime Status	
Status	Displays the current Audit Trail status. For details about each status, see [Audit Trail Events \ Status Changed].
Last error code	Displays the current error status. For details about each error, see [Audit Trail Events \ Fault Occurred]. The last error code retains the most recent error cause. To determine whether an error is currently present, read Status.

20.2.2. Audit Trail Events

Audit Trail supports events for status changes and errors, allowing users to define follow-up actions.

Status Changed

Triggered when the Audit Trail status value changes.

Parameter	Status (DINT): Current status code.
Status Values and	0: Disabled: Audit Trail is not enabled.

Descriptions	1: Stopped: Audit Trail is enabled but not recording.
	2: Logging: Audit Trail is recording normally.
	3: Faulted: An error has occurred.

Last Error Changed

Triggered when the last error code changes. See the [Fault Occurred] descriptions below for details.

Fault Occurred

Triggered when an error occurs while saving or writing to the database, causing the Audit Trail status to change to Faulted.

Parameter	Error Code (UDINT): Error code for the current error.
Error Codes and Descriptions	0: No error.
	6000: Failed to open the database.
	80000: An unknown error occurred.
	80001: Storage is unavailable.
	80002: Failed to create the internal folder.
	80004: Failed to initialize the database schema.
	80005: Failed to update the database or convert its format.
	80100: Failed to serialize the Audit Trail record
	80101: Failed to write the Audit Trail record to the database.
	80102: Failed to write to memory.

Data Flush Status Changed

Triggered when the data flush status changes.

Parameter	Status (UINT): Current status value.
Status Values and Descriptions	0: Disconnected
	1: Connecting
	2: Connected
	3: Storing

Data Flush Error Occurred

Triggered when an error occurs during data flush.

Parameter	Error Code (UINT): Error code for the current error.
Error Codes and Descriptions	0: No error.
	1: Unknown error.
	100: Invalid database parameter or record ID.
	2000: Record not found.
	3000: Failed to connect to the external device or database.
	4000: Access denied.
	6000: Failed to open the database or table.
	6001: Failed to write to the database.
	6002: The database is corrupted or the data format is inconsistent.
	6003: SSL/TLS handshake failed.

20.2.3. Audit Trail Methods

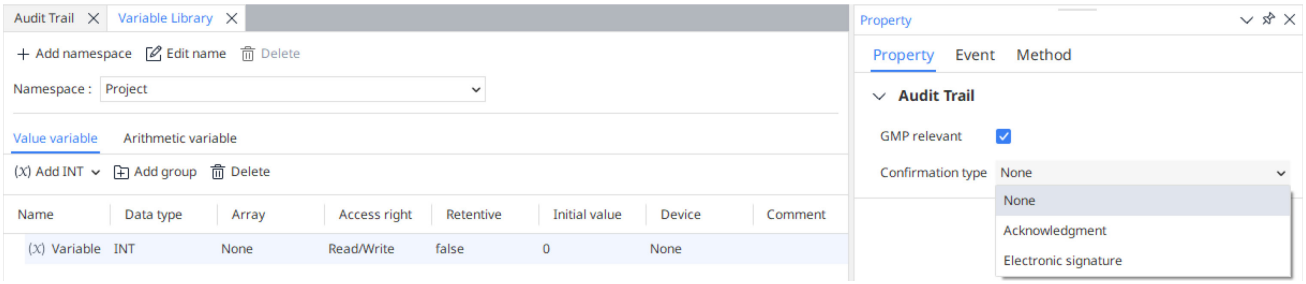
Audit Trail methods are available as global object methods and can be used wherever [Call object method] is supported, including event-triggered actions and widget tasks. For example, when configuring the general properties of a Button widget, select [Task] » [Add] » [Call object method]. Then, under [Category], select [Audit Trail] to choose the corresponding object and method.

Method	Description	Parameters
Start logging	Start Audit Trail logging.	Result (UDINT)
Stop logging	Stop Audit Trail logging.	Result (UDINT)
Log operator action	Manually record an operator action.	Object Name (STRING): Target object name. Description (STRING): Subtitle shown in the confirmation dialog. Confirmation Type (DINT): 0=None 1=Confirmation 2=Electronic Signature Require Comment (BOOL) Description Parameters (STRING) Result (UDINT)

Clear buffer	Discard the remaining data in the buffer before it is flushed to the log file.	None
Flush	Flush data to the storage location.	None

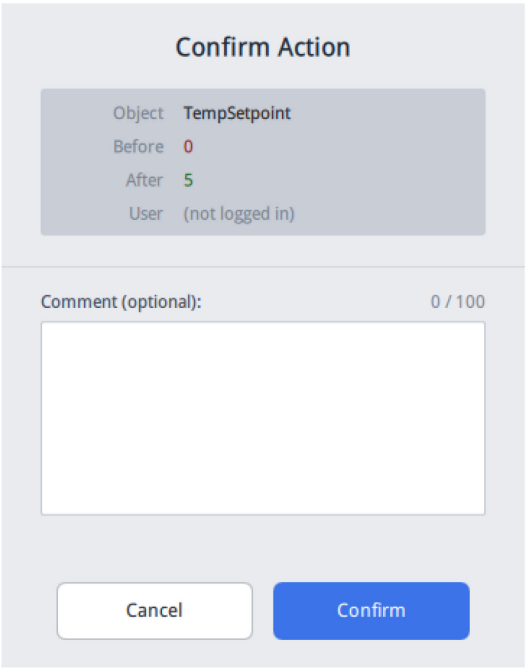
20.3. Usage

20.3.1. Variable Audit Trail

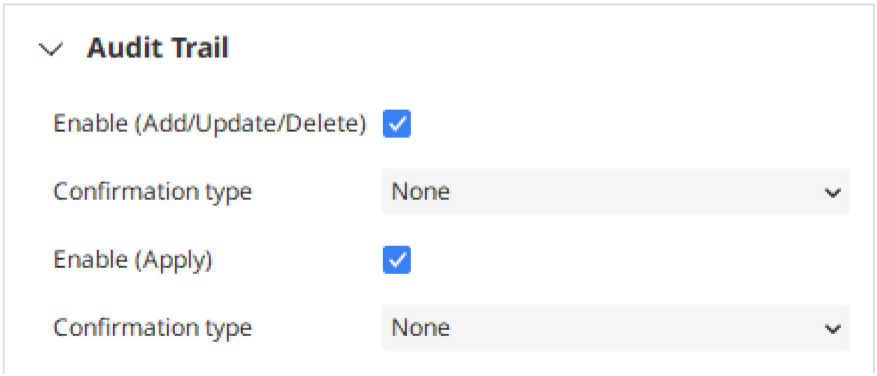


Property	Description
Audit Trail	
GMP relevant	Set whether Audit Trail is enabled for this variable.
Confirmation type	**None:** No confirmation is required when the variable changes. The change is recorded in Audit Trail after the operation. **Acknowledgement:** Confirmation is required when the variable changes. The change is recorded in Audit Trail after confirmation. **Electronic signature:** Confirmation and UAC login information are required when the variable changes. The change is recorded in Audit Trail after confirmation.
Comment required	**Unchecked:** A comment is not required during confirmation before saving to Audit Trail. **Checked:** A comment must be entered during confirmation before saving to Audit Trail.

Variable Audit Trail Confirmation Dialog (Confirmation + Comment Not Required)

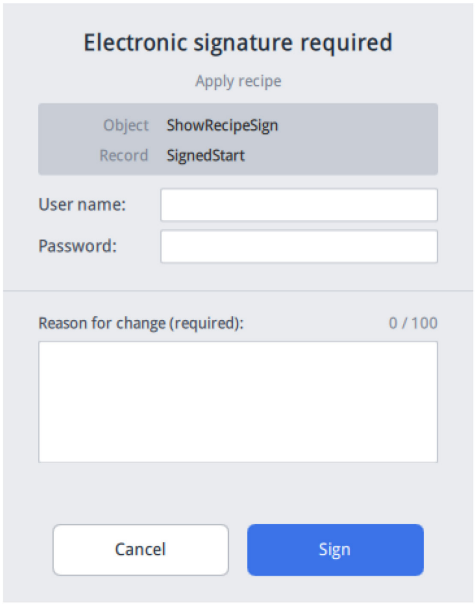


20.3.2. Recipe Database Audit Trail



Property	Description
Audit Trail	
Enable (Add/Update/Delete)	Set whether Audit Trail is enabled when this recipe is added, updated, or deleted.
Enable (Apply)	Set whether Audit Trail is enabled when this recipe is applied to the device.
Confirmation Type	Same as Variable Audit Trail.
Comment required	Same as Variable Audit Trail.

Recipe Database Confirmation Dialog (Electronic Signature + Comment Required)



20.3.3. Audit Trail Usage Limitations

- Audit Trail files can only be exported using the Backup object.
- When a background object attempts to modify a variable for which Audit Trail is enabled and confirmation or an electronic signature is required, the confirmation dialog cannot be displayed. Therefore, the variable value is not changed.
  For example, if a macro (background object) uses the SetVariable function to write data to a variable that requires confirmation or an electronic signature, the write operation is not executed and no Audit Trail record is generated.

20.4. Audit Trail Viewer

The Audit Trail Viewer widget is used to display Audit Trail records in table form on the HMI screen.

20.4.1. Audit Trail Viewer Properties

Item	Description
General	
Name	Set Widget name.
Comment	Set Widget description.
Display	
Show header	Show or hide the column header.
Show caption	Show or hide the caption.
Columns	Set the displayed fields and their order.

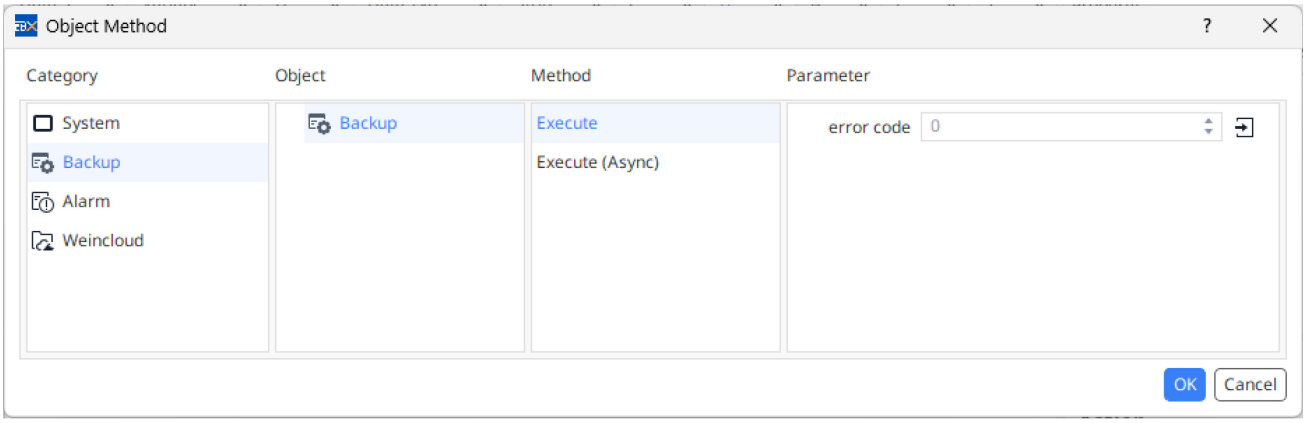
Record ID format	Specify the data type of the ID.
Date/time format	Set the display format for the date and time fields.
Sort order	Sort data in ascending or descending time order.
Appearance	
Style	Set the overall style of the widget. If Default is selected, the colors for fields, text, background, and grid lines must be configured manually.
Style color	Set the style color.
Header text color	Set the text color of the column header.
Horizontal padding	Set spacing between text and column edge.
Stretch last column	Stretch last column to fit widget size.
Toolbar	
Show search bar	Set whether the search bar is displayed. Enter filter text to search for specific records.
Search placeholder	Set the default placeholder text displayed in the search bar when no text is entered.
Show filter button	Set whether the filter button is displayed. Records can be filtered by individual fields.
Background	
Fill	Set background color and pattern.
Radius	Set the corner radius.
Border	Set the border style, width, and color.
Outline	
Picture	Set a background image for the Widget.
Picture color	Some system pictures allow color selection.
Horizontal flip	Flip the image horizontally.
Vertical flip	Flip the image vertically.
Label	
Font	Set text font.
Font size	Set text size.
Bold	Enable bold text.
Italic	Enable italic text.
Color	Set text color.
Availability	
Visible	Set whether the Widget is visible.
Enabled	Set whether the Widget allows control.

20.5. Backup and Verification

Audit Trail files can be exported to a USB drive by using the Backup object to convert them into CSV files. The file integrity can then be verified with ebxtool.exe to ensure that the Audit Trail data has not been tampered with. The following sections explain the procedure.

20.5.1. Backing Up Audit Trail Files

1. In [Data/History], locate the [Maintenance] section, then click [Backup] to open the Backup page.
2. In the Backup properties, set Source Type to [Audit Trail], then configure the backup time range and output settings.
3. Call the method that executes the backup in a task to back up the Audit Trail file on the HMI to the specified path.



20.5.2. Verifying Audit Trail Files

Use ebxtool.exe to offline-check whether an Audit Trail CSV backup file exported from EasyBuilder X remains intact. It can help determine whether the file has been affected by:

- damage during transfer or storage
- content changes made with a spreadsheet or text editor
- inserted, deleted, reordered, or truncated data rows

It is recommended to use this tool in the following situations:

- before delivering, archiving, or submitting an Audit Trail CSV for audit purposes
- after copying a CSV file from the HMI, USB drive, network drive, or another device
- when the file may have been opened manually, edited, or transferred incompletely
- when receiving an Audit Trail CSV backup file from another person and integrity must be confirmed first

Use PowerShell or Command Prompt to run ebxtool.exe. Do not launch it by double-clicking with the mouse.

1. Place ebxtool.exe and the Audit Trail CSV backup file to be verified in an easy-to-find location, for example:
   D:\EBXV101\ebxtool.exe
   D:\EBXV101\AuditTrailLog.csv

2. Open PowerShell or Command Prompt in that folder. First, verify that the ebxtool version and SHA256 information are correct.
   PowerShell:
   ```
   .\ebxtool.exe --version
   Get-FileHash -Algorithm SHA256 .\ebxtool.exe
   ```

   Command Prompt:
   ```
   ebxtool.exe –version
   certutil -hashfile ebxtool.exe SHA256
   ```

3. Verify the Audit Trail CSV backup file.
   PowerShell:
   ```
   .\ebxtool.exe audit verify "D:\EBXV101\AuditTrailLog.csv"
   ```
   Command Prompt:
   ```
   ebxtool.exe audit verify "D:\EBXV101\AuditTrailLog.csv"
   ```

4. After verification is complete, check the exit code.
   PowerShell:
   ```
   $LASTEXITCODE
   ```
   Command Prompt:
   ```
   echo %ERRORLEVEL%
   ```

Exit Code	Result	Recommended Action
0	File is intact	Continue with delivery, archiving, or further processing.
1	File content or structure has changed	Keep the original file, stop using it, re-export it, or notify the administrator.
2	Path, format, or input is incorrect	Check whether the file exists and whether it is a protected Audit

		Trail CSV.
3	Internal tool error	Keep the full output and contact technical support.

5. View the verification results.

Verification passed

```
OK: D:\EBXV101\AuditTrailLog.csv: check code validated across 5 data row(s)
```

If OK is displayed and the exit code is 0, the file has passed the integrity check.

File content changed

```
TAMPERED: D:\EBXV101\AuditTrailLog.csv
  first modified row: data row 1 (CSV line 3)
```

If TAMPERED: is displayed and the exit code is 1, the tool has detected the first location where the content or structure changed.

This does not necessarily mean the file was modified maliciously. Manual editing, file corruption, missing rows, or incomplete transfer may produce the same result.

Keep the original file, do not overwrite it, and obtain or export the file again.

File is not a protected Audit Trail export

```
ebxtool audit: D:\EBXV101\DataSampling.csv: no protected metadata row -- not a protected backup export
```

This means the file is not a currently supported protected Audit Trail CSV. The exit code is 2, and this does not mean the file was tampered with.

File cannot be opened

- whether the file name and path are correct
- whether the USB drive or network drive is still connected
- whether the current account has permission to read the file
- whether the full path, if it contains spaces or Chinese characters, is enclosed in double quotation marks

21. System Object

21.1. Overview 21-2
21.2. General 21-2
21.3. Network 21-3
21.4. Backlight 21-4
21.5. Remote 21-5

21.1. Overview

The System Object integrates functions related to HMI system settings. By binding variables, the required setting parameters can be updated on the HMI in real time.

21.2. General

21.2.1. General Properties

The General properties manage the core settings of the HMI system, including HMI time and language.

Property	Data Type	Description
General		
Hardware Key	STRING	Display the HMI hardware key
Time		
Local time	DateTime	Display the current HMI RTC date and time.
Uptime (ns)	ULINT	Display the elapsed HMI operating time.
Idle duration (sec)	UDINT	Display the elapsed HMI idle time. The idle duration is reset when the HMI is touched.
Cursor		
Visible	BOOL	Set mouse cursor visibility on the HMI screen.
Locale		
Language	USINT	Set the initial language displayed after project download and startup.
Language code	STRING	Display the language code corresponding to the current language setting.
Speaker		
Available	BOOL	Indicate whether the HMI supports a speaker.
Volume	USINT	Set the speaker volume level.
Buzzer		
Available	BOOL	Indicate whether the HMI supports a buzzer.

21.2.2. General Events

The general events are triggered when the system language changes, allowing follow-up actions to be configured.

Language Changed

An event triggered when the system language changes.

Parameter	Language Index (USINT)
current language	Output of the language index after the change.
previous language	Output of the language index before the change.

21.2.3. General Methods

Method	Description	Parameter
Set Time	Set local date and time for system.	Time (DateTime)

21.3. Network

21.3.1. Network Properties

Manage the HMI network configuration.

Property	Data Type	Description
LAN1		
Available	BOOL	Display the LAN1 status.
MAC address	STRING	Display the MAC address.
DHCP mode	UINT	Indicate whether DHCP is enabled to automatically assign an IP address.
IP address	IPv4	Display the IP address.
Subnet mask	IPv4	Display the subnet mask.
Gateway	IPv4	Display the gateway address.
DNS	IPv4	Display the DNS address.
LAN2		
Available	BOOL	Display the LAN2 status.
MAC address	STRING	Display the MAC address.
DHCP mode	UINT	Indicate whether DHCP is enabled to automatically assign an IP address.
IP address	IPv4	Display the IP address.
Subnet mask	IPv4	Display the subnet mask.

21.3.2. Network Methods

The network configuration can be adjusted by calling widget methods.

Method	Description	Parameter
Apply auto addressing to network	Obtain the IP address automatically through DHCP.	interface (STRING)
Apply static addressing to network	Assign the IP address manually.	interface (STRING) IP address (IPv4) subnet mask (IPv4) gateway (IPv4) DNS (IPv4)

21.4. Backlight

21.4.1. Backlight Properties

The Backlight properties manage the HMI backlight saving settings.

Property	Data Type	Description
General		
State	BOOL	Display the backlight status.
Brightness	USINT	Set the backlight brightness. Range: 0 ~ 100.
Saver		
Enabled	BOOL	Enable or disable backlight saving.
Timeout (sec)	UINT	Set the timeout value. When the HMI screen is not touched for the specified duration, the backlight turns off. Range: 60~7200. If this property is bound to a variable and the value is lower than 60, 60 is still used as the minimum value.

21.4.2. Backlight Event

This event is triggered when the backlight state changes, allowing follow-up actions to be configured.

Status Changed
An event triggered when the backlight state changes.

21.5. Remote

21.5.1. Remote Properties

Property	Data Type	Description
Inspector		
Enabled	BOOL	Set whether to enable the Inspector connection.

22. Variant Scope

22.1. Overview of Variant and Scope 22-2
22.2. Differences Between Value variables and Arithmetic Variables 22-3
22.3. Compatibility Rules for Variant Binding and Configuration..... 22-3
22.4. Dynamic Resolution of Arithmetic Variables and Examples..... 22-4
22.5. Common Errors and Debugging Suggestions 22-5

22.1. Overview of Variant and Scope

In EasyBuilder X, many types of data change dynamically during runtime, such as widget properties, value variables, arithmetic variables, and system status values. In this chapter, all such runtime-changing data is collectively referred to as variants.

Understanding variant scope is essential for correct project configuration. Scope refers to the runtime range in which a variant remains valid; in other words, when and under what conditions its existence is guaranteed.

The key to scope resolution is not whether a name is visible, but whether the referenced source is guaranteed to remain valid at the same time as the referencing object during runtime. If the source may become invalid before the referencing object, the reference is invalid and compilation will fail.

For clarity, this chapter uses two terms consistently: the object that issues a reference is called the referencing object (for example, a widget), and the object being read is called the source (for example, a variant). There is only one rule: the referencing object must not depend on a source whose lifetime is shorter than its own.

In EasyBuilder X, scope can be divided from broad to narrow into the following three levels:

Level	Description	Lifetime	Common Examples
Background	The HMI core system layer. Global in scope and not tied to any specific screen.	Same as the runtime of the entire project; the longest and most stable.	Value variables, alarm evaluation, network IP addresses, communication device status
Frontend	The Viewer layer, such as the local HMI screen or an independent client like cMT Remote.	Exists for the duration of a single Viewer connection and can be shared by all screens in that Viewer.	System language, current user permissions
Specific Screen	Inside a specific screen or window.	Exists only while that screen is open, and ceases to exist when the screen is closed.	The Value property of a Numeric Input widget on a specific screen

22.2. Differences Between Value variables and Arithmetic Variables

During project development, variables are declared in the Variable Library for data transfer or calculation. These variables mainly fall into two types: value variables and arithmetic variables.

Variable Type	Default Level	Characteristics	Typical Usage
Value Variable	Background (global)	Equivalent to a global variable. Its value is retained regardless of screen switching.	Suitable for cross-screen data transfer or intermediate data storage
Arithmetic Variable	Dynamic	Its scope is not fixed. It is automatically narrowed according to the objects referenced in its expression. If the expression references a widget on a specific screen, it is valid only while that screen is open.	Suitable for real-time calculation based on multiple data sources

22.3. Compatibility Rules for Variant Binding and Configuration

When configuring widget properties or conditions, always follow this compatibility rule: the referencing object must not depend on a source whose lifetime is shorter than its own.

- **A shorter-lived object may reference a longer-lived source only**

If a widget (the referencing object) references a variant (the source), the source must have a lifetime longer than or equal to that of the referencing object. Conversely, a longer-lived object must not depend on a shorter-lived source, because the shorter-lived source may disappear at any time.

Correct example: A widget on a specific screen reads a background value variable. The background lifetime is longer than that of the screen.

Incorrect example: A background-level object is bound to the system language (a frontend variant). The background core cannot guarantee when the frontend Viewer may disconnect. Once it disconnects, the source no longer exists.

- **Different screens have incompatible scopes**

Each screen is independent. Screen A cannot guarantee that Screen B is always open. Therefore, variables from different screens belong to incompatible scopes, even if their names or types are identical.

Incorrect example: A widget on Screen A directly reads the switch status on Screen B. This is not allowed.
Likewise, an arithmetic variable cannot reference variables from two different screens in the same expression at the same time (see Scenario 2 in 22.5).

- **Use value variables as a bridge**

If data must be transferred across screens, or if a longer-lived object needs to access data from a shorter-lived source, create a background-level value variable as an intermediary.

Correct approach: First write the state of Screen B into a value variable, and then let a widget on Screen A read that value variable. In this way, both sides depend only on a long-lived background variant instead of depending on each other directly.

22.4. Dynamic Resolution of Arithmetic Variables and Examples

The special characteristic of arithmetic variables is that their scope is not fixed. Instead, it dynamically converges from broad to narrow according to the objects referenced in the expression:

1. The initial state is the broadest, background-level scope, when only background variants or constant values are referenced.
2. If the expression references a frontend variant, its scope converges to the frontend level.
3. If the expression references a variable from a specific screen, it further converges to that screen level.
4. If the expression references multiple sources, the final scope is determined by the narrowest compatible level. For example, if it references both a frontend variant and a variable from Screen A, the result legally converges to Screen A, because Screen A is narrower than the frontend and the two are compatible. However, if it references variables from two different screens, compilation will fail because those scopes are incompatible.

Example: Assume TempC and Offset are both background-level value variables. The following expression references only background sources, so the arithmetic variable remains at the background level and can be used safely throughout the project:

${TempC} * 9 / 5 + 32 + ${Offset}

However, if the expression is changed to reference the value of a Numeric widget on Screen A, then the arithmetic variable is narrowed to the Screen A level and is valid only while Screen A is open.
The table below summarizes the resolution results of common combinations:

Expression Sources	Final Scope	Description
Value Variable (Background) + Value Variable (Background)	Background	Both exist globally, so the arithmetic variable remains at the background level.
System Language (Frontend) + Value Variable (Background)	Frontend	Because it depends on the frontend language, the arithmetic variable is narrowed to the frontend level.
Value of a Numeric Widget on Screen A (Specific Screen) + Value Variable (Background)	Screen A	Because it depends on Screen A, the arithmetic variable is valid only while Screen A is open.
System Language (Frontend) + Value of a Numeric Widget on Screen A (Specific Screen)	Screen A	Since the two are compatible, the result legally converges to the narrowest compatible level, which is Screen A.

22.5. Common Errors and Debugging Suggestions

To prevent data loss or runtime exceptions, EasyBuilder X performs strict checks during compilation. If you encounter scope-related errors during compilation, the cause is usually a violation of the compatibility rule.

Scenario 1: Incompatible scope

Cause: A longer-lived object depends on a shorter-lived source. For example, a property of a background-level object is bound to a frontend or specific-screen variant.
Solution: Check the binding source and ensure that the source has a lifetime at least as long as that of the referencing object. Alternatively, use a background-level value variable as an intermediary.
Special note: If the bound source is itself an arithmetic variable, its actual level depends on the scope it converges to. For example, if an arithmetic variable converges to the frontend level because it references a frontend variant, then binding it to a background-level object will still be incompatible and cause a compilation error.

Scenario 2: An arithmetic variable references two different screens

Cause: The expression of the arithmetic variable includes both a value from Screen A and a state from Screen B. The system cannot guarantee that these two screens are always open at the same time, so it rejects this unsafe calculation.
Solution: Do not access values across screens directly in the expression. First write the state of Screen B into a background-level value variable, and then let the arithmetic variable read that value variable.

To make the project more stable and easier to maintain, follow these design practices:

1. Keep arithmetic variables at a higher, broader level whenever possible. Unless necessary, avoid directly referencing widgets on a specific screen in an expression, so that the scope is not narrowed unnecessarily.
2. Treat widget properties as local states of that screen. If a value needs to be shared across screens or globally, do not read another screen's widget directly. Instead, write it into a background-level variant first and then share it.
3. Avoid direct cross-screen access. Whenever data needs to be shared across screens or levels, use a background-level value variable as an intermediary.

By following these principles, the project can maintain clearer lifetime boundaries during runtime, as well as more stable and predictable behavior.

Finally, note that these scope rules apply not only to arithmetic variables and property binding. Any case that requires reading another variable during runtime, including conditional expressions, variants inserted into dynamic strings, and parameter references in actions or events, must also follow the same compatibility rules.

23. cMT Inspector

23.1. Overview 23-2
23.2. cMT Inspector Interface 23-3

23.1. Overview

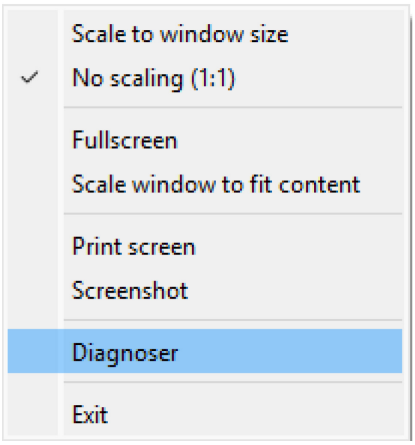
cMT Inspector can be used to monitor widget address states and values, browse variables in a project, and view the real-time connection status of devices.

23.1.1. Opening cMT Inspector

cMT Inspector can be used during simulation or with an HMI. The two scenarios are described below.

Simulation

1. During Online Simulation or Offline Simulation, right-click the simulation screen in cMT Remote and select [Diagnoser] to open cMT Inspector.



HMI

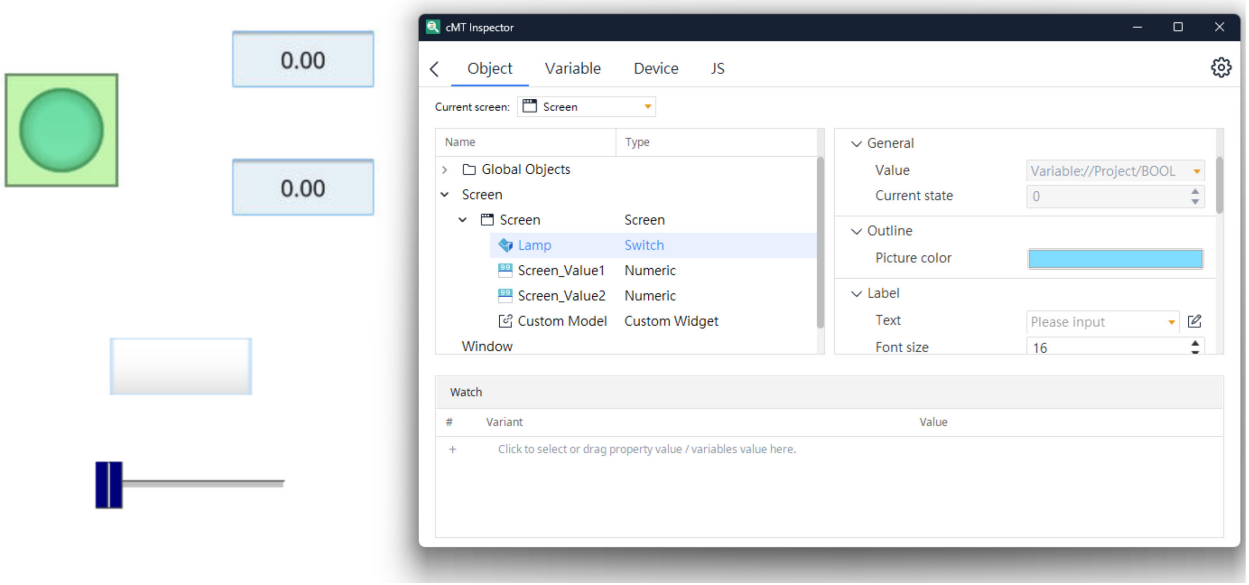
1. To allow cMT Inspector to connect to an HMI, go to [System Object] » [Remote] in the project and enable Inspector.
2. Run cMTInspector.exe from the EasyBuilder X installation folder to open the standalone cMT Inspector application. The Search tab is displayed by default.
3. Enter the HMI name or IP address in the search field. A list of available HMIs on the local network is displayed below.
4. Select [Show EBX HMIs only] to display only HMIs running an EBX project. This makes it easier to locate the target HMI in an environment containing both EasyBuilder Pro and EasyBuilder X HMIs.
5. Select the HMI to monitor, connect to it, and open the cMT Inspector interface.

23.2. cMT Inspector Interface

23.2.1. Object

This tab displays the object tree of the current project, including global objects and widgets under each screen and window.

1. Select an item in the object tree to display its properties in the panel on the right.
2. When a widget is selected in the object tree, the corresponding widget is highlighted on the HMI screen for easy identification.



3. Variables and widget properties can be added to the Watch table at the bottom for monitoring. A property can also be dragged directly to Watch to display its current value in real time.
4. If the screen containing the widget is not currently open on the HMI, the Watch value field displays “View not open on the HMI” instead of the actual value.

23.2.2. Variable

This tab displays all variables in the project. Add a variable to Watch to monitor its value changes in real time.

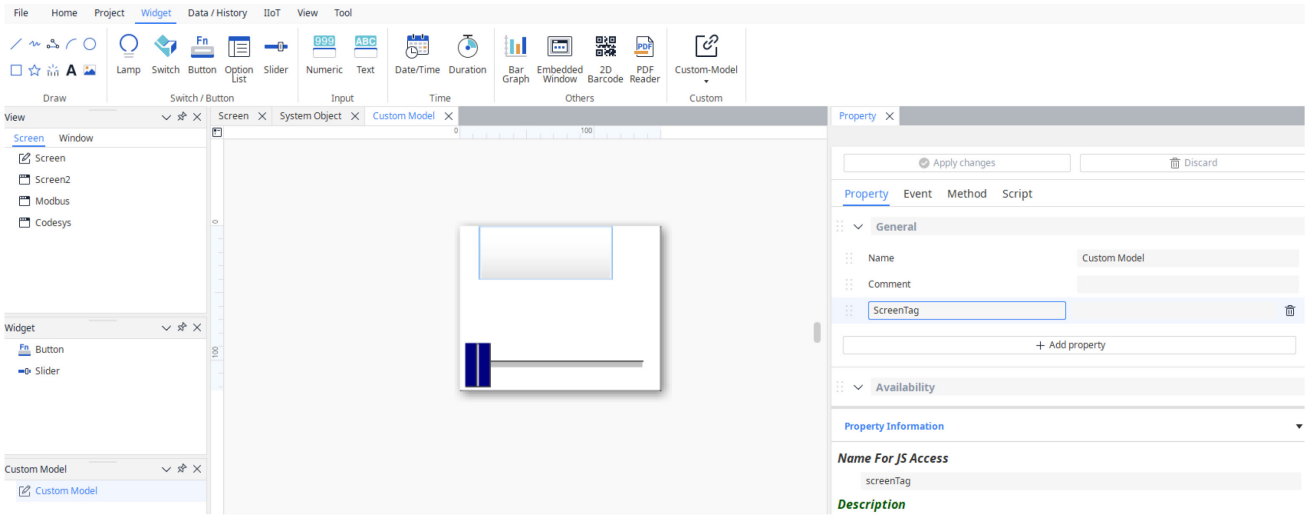
23.2.3. Device

This tab displays real-time connection information and status for all devices in the project, making it easy to verify whether each device is communicating properly.

23.2.4. JS

This tab displays console messages generated by JS scripts in Custom Widgets on the current screen. The following example illustrates how this works:

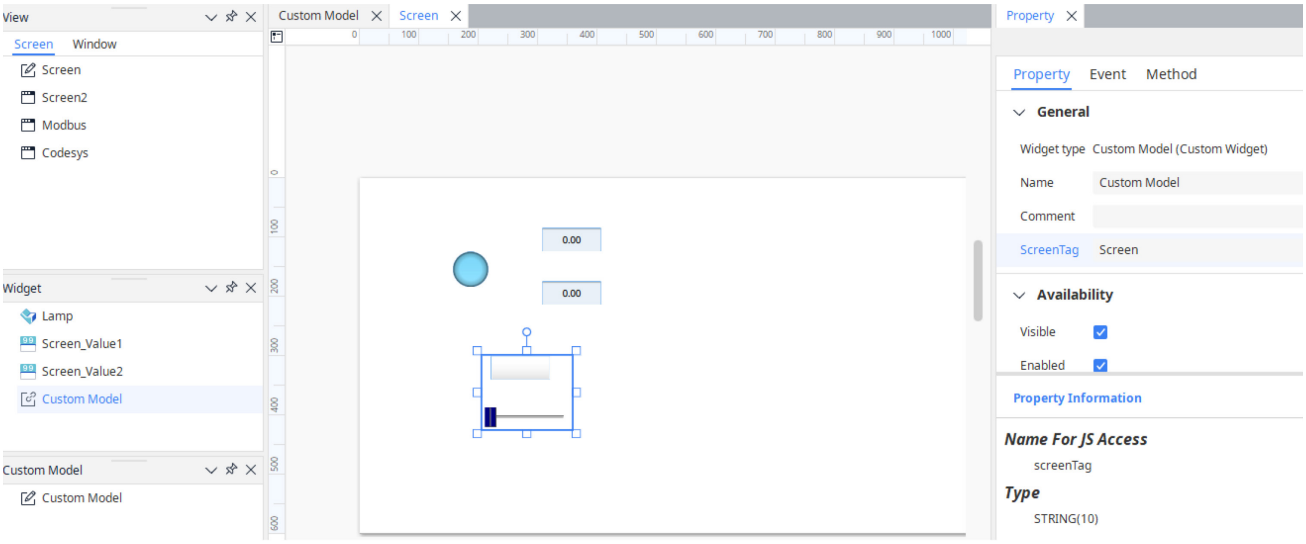
1. Add a String property to a Custom Model, for example ScreenTag, and set its JS access name to screenTag.



2. In the Script tab of the Custom Model, enter the following script. The script uses “this.property.screenTag” to obtain the current Screen Tag value and output it to the console.

```
const tag = String(this.property.screenTag || 'unknown');
console.log('[screen-probe] enter: ' + tag);
setInterval(() => {
console.log('[screen-probe] on screen: ' + tag);
}, 1000);
```

3. Place the Custom Model on the screen to be monitored and assign a value to its ScreenTag property, for example the screen name “Screen”.



4. As long as the Custom Widget exists on the screen currently displayed in cMT Inspector, the JS tab continuously displays the corresponding console messages.

