Add Symbols to Displays How-to

Insert and reuse graphical symbols on displays.

How-to Pillars Operator Displays Symbols | Tutorial | How-To Guide | Reference

---

Overview

This guide walks you through using and creating Symbols - customizable graphical components that combine visual representation with dynamic behavior and real-time data binding. Symbols provide reusable components that update automatically across all displays when modified.

Prerequisites:

- Tags configured in Unified Namespace
- Basic understanding of display creation
- Knowledge of dynamic properties

Understanding Symbol Libraries
Using Existing Symbols
Configuring Symbol Instances
Creating Custom Symbols
Advanced Symbol Features
Customizing Wizard Symbols
Managing Solution Symbols
Best Practices
Common Symbol Patterns
Troubleshooting
Legacy Symbols

Understanding Symbol Libraries

Two-Level Library System

Product Symbol Library:

- Thousands of pre-built symbols installed with software
- Stored in `Library.dbsln` file
- Located in `\Users\Public\Public Documents\<productName>\SystemSolutions`
- Can be customized by opening as a solution

Solution Local Library:

- Symbols copied from Product Library when used
- Stored within solution file
- Ensures solution independence and portability
- Listed under "In Solution" in Components Panel

Using Existing Symbols

Finding and Inserting Symbols

1. Navigate to **Displays Draw**
2. In Components Panel, expand **Symbols** folder:
    - **Library** - Global product symbols
    - **In Solution** - Symbols already in your solution
3. Check "**with Dynamics**" to filter symbols with behavior
4. Insert symbol using either method:
    - **Click** symbol, then **click** in drawing area
    - **Drag and drop** from panel

Symbol Wizards

The Library includes five wizard symbols — **BLOWER**, **MOTOR**, **PUMP**, **TANK**, **VALVE** — that each generate a family of pre-drawn variants for one industrial component. A wizard is not a template; it is a variant picker. When you drag a wizard symbol onto a Display, the Designer copies the chosen variant into your solution under a new short name (the first wizard pump becomes `PUMP1`, the next `PUMP2`) and opens a configuration dialog. The dialog presents radio buttons for the visual choices that apply to that component family — Appearance (Simple, Detailed, or Inst.+Pipe), Orientation, Feet, Style, Output, Level, Hand Wheel, Plug — with a live preview. Once you confirm, the variant is a static symbol owned by your solution; later changes to the wizard library do not regenerate it. Each wizard exposes a fixed set of dynamic inputs through its SymbolLabels — every Pump variant carries `Rotate`, `RPM`, `State`; TANK exposes up to six (`Level`, `Max`, `Min`, `Rotate`, `RPM`, `State`) — bind these to tags to make the symbol live.

Configuring Symbol Instances

Symbol Configuration Dialog

Double-click any symbol to open configuration:

Property	Description
Is Linked With Library	Updates when library symbol changes
Linked Size (Proportional)	Maintains size ratio with library
Internal Dynamics	Lists animations within symbol
Symbol Links	Parameter mappings for this instance
Copy and Paste	Transfer configurations between symbols
Apply	Save changes (auto-applies on close)

Mapping Parameters

Symbol parameters use simple binding:

1. Double-click symbol to open configuration
2. In **Symbol Links** section, find parameters
3. Map each parameter to tags or expressions:
    - Type tag name directly (with IntelliSense)
    - Click **...** button to browse tags
    - Enter expressions or constants

Common wizard symbol parameters:

- **#State**: 0=HPGOff, 1=HPGOn, 2+=HPGDefault colors
- **#Rotate**: 0=static, 1=rotating
- **#RPM**: rotation speed when `#Rotate=1`
- **#Level**: Tank fill level (with #Min/#Max)

Creating Custom Symbols

From Basic Shapes

1. **Draw your symbol:**
    - Use drawing tools (Rectangle, Ellipse, Polygon, Polyline)
    - Combine shapes with toolbar commands
    - Apply colors, gradients, shadows
2. **Add Dynamic Behavior:**
    - Select element
    - Double-click to open properties
    - Add dynamics (FillColor, Visibility, Rotate, etc.)
3. **Create Parameters:** Use hashtag syntax in expressions:

    ```
    #State:Client.SimulationDigital
    #Value:Tag.PlaceHolder.DoubleTag
    #Min:0
    #Max:100
    ```

    - Text before `:` becomes parameter name
    - Text after `:` is default value
4. **Save as Symbol:**
    - Select all elements
    - Right-click "Make a new symbol"
    - Enter Name, Folder, Description
    - Symbol appears in "In Solution" library

Note: A symbol cannot contain another symbol. Nested symbols are not supported: the "Make a new symbol" command is unavailable when the selection includes a symbol, and the Designer blocks inserting a symbol into the symbol currently being edited.

Example: Tank Level Symbol

Create a simple tank with level indicator:

1. Draw rectangle (tank body)
2. Add Bargraph dynamic:
    - Expression: `#TagValue:Client.Second`
    - Min: `#Min:0`
    - Max: `#Max:100`
3. Add TextBox for value display:
    - LinkedValue: `#TagValue:Client.Second`

4. Select both Create Symbol
5. Result: Symbol with three parameters (TagValue, Min, Max)

Advanced Symbol Features

Linking Symbols to Tag Types

Automatically use symbols when dropping tags:

1. Go to **Unified Namespace Tags**
2. Select tag
3. Click **Map To Symbol** button
4. Choose symbol from library
5. Now dragging this tag creates the symbol

Symbols for UserTypes

Create symbols that bind to UserType members:

For UserType "PID" with Setpoint, PV, CV:

```
Expression: #TagValue:(Tag.Loop1).Setpoint
Expression: #TagValue:(Tag.Loop1).PV
Expression: #TagValue:(Tag.Loop1).CV
```

Using parentheses replaces only the tag name, keeping member references.

Default Symbol Mapping

- Name symbol same as UserType for automatic mapping
- Or use **Map to Symbol** button in UserTypes
- Default types (Integer, Digital, etc.) have built-in text I/O

Using PlaceHolder Tags

Best practice for portable symbols:

```
#TagName:Tag.PlaceHolder.IntegerTag
#TagName:(Tag.PlaceHolder.DoubleTag).Quality
```

Benefits:

- Solution-independent symbols
- Clear type expectations
- IntelliSense support

Customizing Wizard Symbols

Editing Wizard Symbols

1. Select symbol on display
2. Click **Edit Selected Symbol** in toolbar
3. Environment title changes to "SYMBOL EDITOR"
4. Make modifications (add labels, values, etc.)
5. Save with toolbar icon or cancel

Note: After customization, wizard no longer applies to that symbol.

Breaking Apart Symbols

To modify without library link:

1. Select symbol
2. Click **Break Apart** button
3. Elements become individual objects
4. Modify as needed
5. Optionally recreate as new symbol

Managing Solution Symbols

Symbol Management Page

Navigate to **Displays Symbols** to:

- View all solution symbols
- Check size and descriptions
- Export symbols
- Delete unused symbols

Operations

Save as Image:

- Creates PNG preview file

Export to Library:

- Saves to global `Library.dbsln`
- Available to all solutions on computer

Delete Symbols:

- Select rows
- Click delete (X) icon

Best Practices

Performance

Reuse symbols from "In Solution" instead of Library

Link to library for consistent updates

Use parameters instead of fixed values

Test symbols with PlaceHolder tags first

Organization

Create folders to organize symbols

Use descriptive names for symbols and parameters

Document complex symbol behavior

Version control important symbols

Design Guidelines

Keep symbols simple - combine for complex displays

Expose key parameters only

Use consistent parameter naming (#State, #Value)

Provide defaults for all parameters

Common Symbol Patterns

Status Indicator

```
Rectangle with:
- FillColor: #State:0 (0=Gray, 1=Green, 2=Red)
- Visibility: #Enable:1
- Action: OpenPopup(#DetailDisplay:"Details")
```

Value Display with Units

```
TextBox group with:
- Value: #Tag:Tag.PlaceHolder.DoubleTag
- Format: #Format:"N2"
- Units: #Units:"°C"
```

Control Button

```
Button with:
- Action: SetValue
- Target: #ControlTag:Tag.Command
- Value: #CommandValue:1
- Security: #Permission:"Operator"
```

Troubleshooting

Symbol not updating in displays

- Check "Is Linked With Library" setting
- Save symbol in library after changes
- Refresh displays

Parameters not working

- Verify hashtag syntax: `#Name:Default`
- Check for typos in parameter names
- Ensure consistent parameter use

Symbol appears broken

- Verify all internal tag references exist
- Check dynamic expressions for errors
- Test with known good tags

Can't edit wizard symbol

- Editing removes wizard capability
- Create copy first if preserving original
- Use "Break Apart" for one-time edit

Legacy Symbols

Some symbols created in older solutions may show a Symbol Conflict dialog even without being copied from another solution. This happens because the symbol contains canvas elements with tag versions or properties that are no longer current.

This is expected behavior. The dialog is working correctly, as it detects that the symbol's internal data does not match the current canvas tag definitions. To stop the prompt from appearing for a given symbol, update it so its elements use the current tag definitions.

This does not affect newly created symbols, as they are always saved with up-to-date tag definitions.

This guide covered creating, configuring, and managing Symbols in your solution, from using pre-built library symbols to creating custom reusable components with dynamic behavior and parameter-based configuration.

In this section...
