Symbols Control Reference

Insert and configure reusable symbols.

Symbols are customizable graphical components that can be used on screens. The symbol has the option to be managed in a centralized way, so when changing a Symbol in the solution library, it changes automatically in all displays.

Symbols can have only a static drawing, or they can also have a dynamic behavior and binding to real-time Tags. When using or creating symbols with dynamic properties and tags binding, a simple configuration dialog allows to map the dynamic values to the desired tags, when applying multiple instances of the symbol.

Additionally to allowing full customization, some of symbols frequently used in SCADA displays, like Motors, Blowers, Pumps, Valves and Tasks have a Wizard tool, allowing a quicker customization of its appearance.

Product Library and Local Solution Library
Inserting a Symbol and Customizing Parameters
- Symbol Configuration Dialog
- Symbols and Dynamics

Creating New Symbols
- Static Drawing
- Node UID Naming for Connector Snap Targets
- Dynamic Properties and Parameters
- Color as a Parameter
- Clicks Inside a Symbol

Automatically Linking Symbols And Tags
- Linking Symbols and Tags
- Dropping Tags on Displays
- Linking Symbols To Tag Templates (UserTypes)

Managing the Solution Symbols
Configuration

Product Library and Local Solution Library

Product Symbol Library

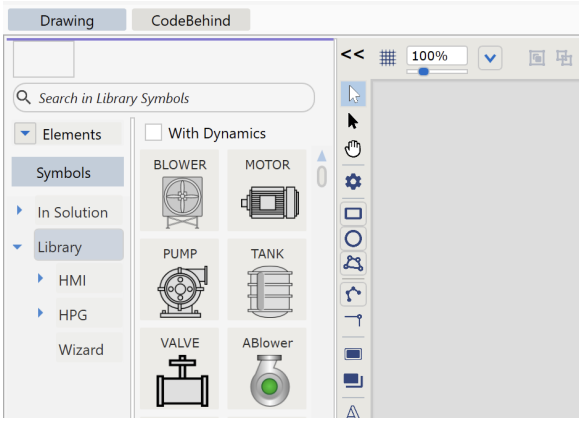
When installing the software, a set with a few thousands of global symbols is made available to the solutions. Those pre-defined symbols are stored in the file Library.dbsln, copied to the SystemSolutions folder in the Public Documents product folder. That file is a standard built-in solution file, therefore, if you want to customize the built-in library, you just need to open that solution with the Designer and modify the symbols, or other other elements in that global Library.

Solution Local Library

When a Symbol from the global library is used in a Solution, a copy of that symbol is saved in the Solution file, this is important and useful, so it keeps the concept that a solution file is fully autonomous and independent. In that way, when copied to a production computer for deployment, or archived, it has within itself everything the solution needs to run without the need of external dependencies.

In the Drawing Tool, on the Symbols section, the Tree under "In Solution" are the symbols that are part of the solution file, those symbols are also listed in the solution explorer, navigating to Displays -> Symbols.

The tree under Library, are the symbols in the global library file.



The "with Dynamics" checkBox filters to show only the symbols with dynamic properties.

When you select a Symbol from the Library sub-folders, a copy of that Symbol is created in the "In Solution" folders to be used on the displays of that solution.

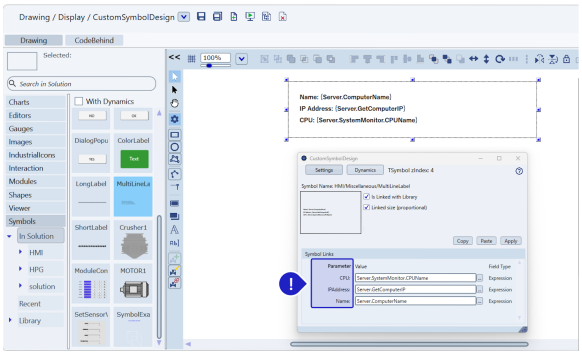
When you select a Symbol from the "In Solution" folder, you are re-using a Symbols that is already part of the solution.

The first five symbols showed on the list, Blower, Motor, Pump, Tank and Valve, are the Symbol Wizards, which have an additional features: a customization dialog that allows you to easily select the appearance of the elements, guiding the selection across the typical visuals for that type of symbol, for instance if the Task is Horizontal or Vertical, and various other customization options. See Symbol Wizards for more information.

Inserting a Symbol and Customizing Parameters

To Insert a Symbol.

1. Select a Symbol from the graphical components. It can be from the "Library" sub-folders (external Library file), or from the "In Solution" folder to pickup symbols ready in the solution.
2. You can Click on the Symbol in the ListBox to select it, then Click again in the drawing area to insert. Or, if you prefer, you



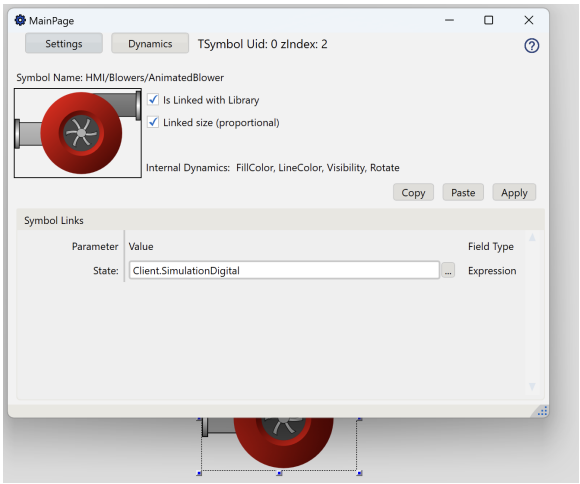
can perform a Drag-Drop operation. Both mouse actions are accepted.

3. When you double-click on the Symbol, or press the Properties icon in the toolbar, the Symbol Configuration dialog is shown.
4. if the Symbol has dynamic bindings with tags, the Symbol Links will show the **Parameters** defined for that symbol, to which tag or expression it is currently connected. Just use the edit the solution to bind that symbol instance to another variable.

Symbol Configuration Dialog

Symbol Configuration Table

Property	Description
Is Linked With Library	If checked, the symbol instance appearance will update when saving the symbol to Solution Library. if unchecked, the symbols is now specific the displays, disconnected from the library.
Linked Size (Proportional)	When linked to the Solution Library, if the symbol is resized in the Library, it resized on the display. **10.1.5 change:** editing Width or Height in the Properties panel now automatically clears `LinkedSize` so the new dimensions take effect immediately. In earlier releases, manual size edits were silently lost while `LinkedSize` remained true.
LockAspectRatio	*New in 10.1.5.* Decouples aspect-ratio locking from `LinkedSize`. Default `false`. When set to `true`, manual Width or Height edits in the Properties panel preserve the symbol's aspect ratio — changing one dimension scales the other proportionally.
Has Legend	*New in 10.1.5.* When checked, a `TLegend` label child is automatically created and follows the symbol on the display. Can be set from the Symbol Configuration panel and from the Make Symbol dialog. See Display Legends for the legend control reference.
Internal Dynamics	The list of dynamic animations used inside the symbol. Further explanation on next section, Symbols and Dynamics
Copy and Paste	Button to copy the Symbol Links to the clipboard, or paste from it.
Apply	Apply the edited expressions to the Symbol. This action is optional, when closing the configuration dialog, or selecting another object the expression changes are automatically applied.
Symbol Links	Table with the Parameters defined to that Symbol. Parameters are an easy and quick way to customize each instance of the Symbol to specific tags when using multiple copies of the Symbol. When creating the symbol, none, one, or more, parameters can be defined to change specific properties of the underlying symbol dynamic behavior and tags binding.



Symbols and Dynamics

When you insert a symbol from the Symbol Library, it may have customization parameters, for instance, the previous image, has a parameter STATE, which should receive the Expression that will control if the blower will show ON or OFF when running the solution.

Those parameters will affect the behavior of Dynamic properties embedded in the symbol creation. Those internal settings are not visible, unless you EDIT or BREAK the symbol, as described in later sections. The Configuration dialog has a a string, INTERNAL DYNAMICS, which just list the types of animations the symbols has, ir other to assist the understanding of its behavior in runtime.

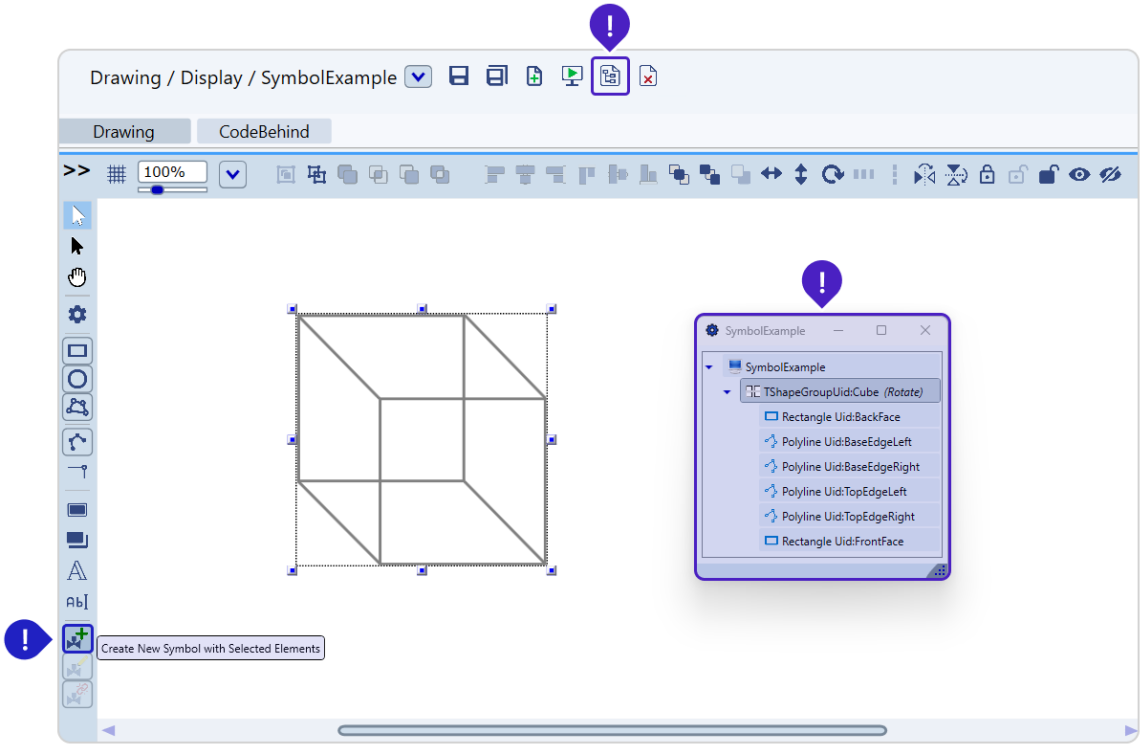
The **DYNAMICS BUTTON** in the configuration dialog isn't related to those internal dynamics, but it's intended to add additional dynamic behavior that will apply to the entire symbol, in the same way you can do with any graphical element in the screen.

For instance, you can apply an Visibility-Opacity property, to change the appearance of the entire symbol when in maintenance state, or you can add a ACTION dynamics to perform some operation when the operator clicks on the Symbol. For actions on elements inside the symbol, see Clicks Inside a Symbol below.

Creating New Symbols

Static Drawing

- You can use the drawing tools and the COMBINE commands in the horizontal toolbar to draw your own symbols.
- You can also use the symbols from the Symbol Factory and click the UNGROUP command in the horizontal toolbar to edit the imported symbols.
- You can use the Direct Selection cursor to edit internal elements without needing to ungroup.
- If you want to save a new symbol to the Local Symbols Library, select the object or objects that you want to be part of the symbol. Then, right-click and select Make a new symbol.
- Enter the Name, Folder, and the description of the Symbol.
- You can open the Symbol Library and choose the symbol to use in your displays.



Node UID Naming for Connector Snap Targets

For a symbol to be a valid snap target for the Connector tool, the symbol author must assign a **Node UID** to each element intended as a connection point. The Connector tool only recognises elements whose UID matches the `Node<N>` pattern; elements without a Node UID are invisible to connector snapping.

Naming convention: use `Node1`, `Node2`, `Node3`, … one identifier per connection point. The numbering is sequential and starts at 1. Names are case-sensitive.

How to assign: while editing the symbol, select the element to expose as a connector target (typically a small Ellipse, Rectangle, or invisible shape at the desired snap location), open the Properties panel, and set the UID field to `Node1` (or the next available `Node<N>`). Repeat for each additional snap point. When the symbol is used in a display, the Connector tool will snap to every `Node<N>` element on hover.

Example: a Valve symbol with one inlet and one outlet exposes two snap targets — the inlet circle is named `Node1` and the outlet circle is named `Node2`. A pipe drawn with the Connector tool snaps to either node and stays anchored when the symbol is moved.

Dynamic Properties and Parameters

After creating a new drawing, or modifying an existing one, you can optionally add Dynamic Properties to the symbol.

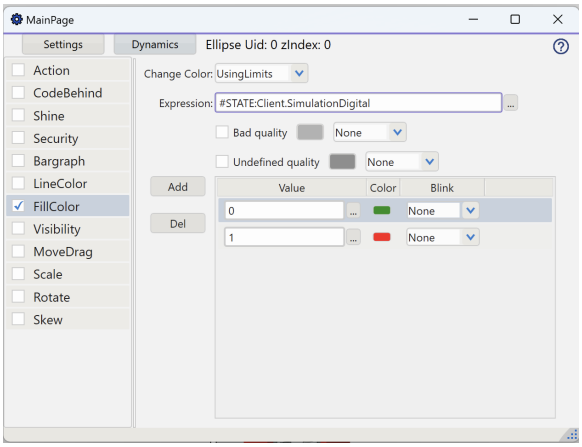
See the page Dynamic Behavior Control Reference to learn about the dynamic animations available.

When applying those dynamics, you can make the configuration so the dynamic behavior is fixed, embedded inside the symbol, or expose some properties or Parameters for customization each time a new instance of the Symbol is used in a display.

For instance, when creating a symbol for a VALVE that has the dynamic behavior to change its color to Green or Red, according to its ON or OFF state, it is desirable that the Tag driving that On/Off state can be customized to each instance the symbol is used.

This is accomplished by using a special syntax on the Value field (or other fields you want to later bind), which uses the Hashtag Symbol (#) to add a Parameter Key to that symbol.

Example, Expression: #STATE:Client.SimulationDigital



In this example, when applying the text **#STATE:** you are creating a Parameter, a new key to the dictionary of parameters this symbol can have. That is done by using the hashtag (#) followed by the Parameter.

The STATE Parameter will allow to later customize which tag shall be used to control the value of ColorFill dynamic.

The value after the colon char, in this example "Client.SimulationDigital", is the default value for that property.

The same parameter can be used in various animations of the Symbol, for instance, if adding also a LineColor dynamic, you can use the same STATE parameter key, so when biding a tag to the STATE parameters, both the Fill and the Line dynamic properties will be connected with the tag.

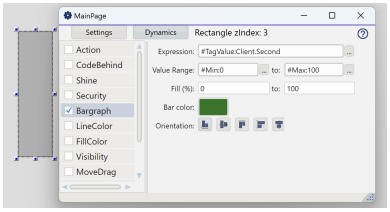
When creating Symbols, it is useful to add simulated values, or the Tag. PlaceHolder variable, so your symbol is independent of you solution data, and can be easily be exported to other solutions, it also brings the befit of having a preview of the symbol behavior before binding it to the production data.

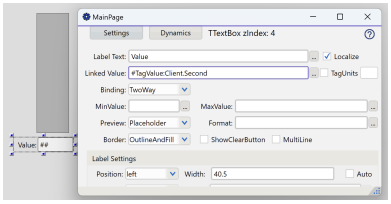
In this other example, a Bargraph dynamic property was added to the object, with 3 Parameters keys being created. The TagValue, the Min, and Max parameters. The default values for Min and Max were set to 0 and 100, the default value for the TagValue was the internal variable Client.Second.

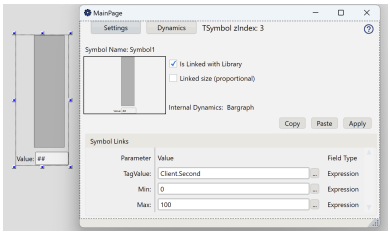
Then a TextBox object was added, using the same TagValue Parameter key to its value, so when selecting binding a new tag to the TagValue paramet ers, both the level of Bargraph and the TextBox message will show the value of the selected tag.

When those two graphical elements (the Bargraph and the TextBox) are selected, and applying the command GROUP to SYMBOL (or Save to Symbol if ending an existing one), the Symbol Configuration Dialog will present 3 Parameters to be customized to the symbol, on each instance of Symbol.

- **TagValue**: this parameter will set which tag will drive the level of the barograph and the value on the TextBox
- **Min**: this parameters will set the Tag, or Constant, or Value, that will driver the minimum value for the bargraph level.
- **Max**: this parameters will set the Tag, or Constant, or Value, that will driver the maximum value for the bargraph level.







Color as a Parameter

A symbol takes its color from each instance this way: add a FillColor, LineColor or TextColor dynamic to the element, set **Change Color** to **AbsoluteV alue**, and write a parameter with a default color in the **Expression** field:

```
#Color:"Red"
```

The symbol then shows a **Color** parameter in Symbol Links, and each instance sets its own value:

Value in Symbol Links	Result
`"Green"`	Named color.
`"#FF00A0E0"`	Hexadecimal color with eight digits, alpha first (ARGB).
A Text tag, such as `Tag.LineColor`	The color follows the tag at runtime. The tag holds a color name or an eight-digit hexadecimal color.
A numeric tag	The value is read as a 32-bit ARGB color.

A value that is not a valid color paints the element black, and so does an empty value in HTML5 displays. Give every instance a valid color.

In the **UsingLimits** mode, each limit value takes a parameter, but the colors of the limits list stay fixed in the symbol, as do the Bad Quality and Undefined Quality colors. To vary those colors per instance, use AbsoluteValue with an expression that returns the color.

Clicks Inside a Symbol

Actions live on two levels:

- **On the symbol instance**, added with the Dynamics button in the Symbol Configuration dialog. The action runs when the operator clicks anywhere on the symbol.
- **On an element inside the symbol**, such as a button, added while editing the symbol. The action runs when the operator clicks that element.

An action inside the symbol uses the symbol parameters in its fields, for example `#Target:Tag.PlaceHolder.IntegerTag` in the object field of a SetValue action. Each instance then writes to its own tag. The object, value, expression and result fields of the Action dynamic all accept parameters.

Put each click action on one level: on the symbol instance, or on the elements inside it.

Automatically Linking Symbols And Tags

Linking Symbols and Tags

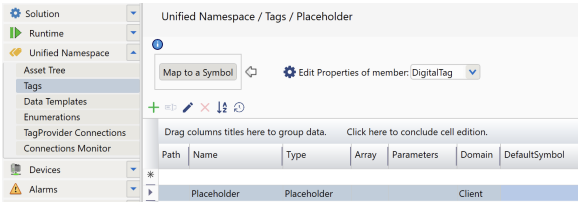
A symbol can be configured to be the default representation of a tag type.

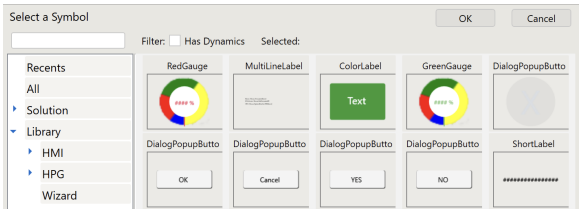
Whenever you paste a tag into the Drawing, a symbol will be automatically created and mapped to the inserted tag.

The Digital, Integer, Double, Decimal, Text, DateTime, and TimeSpan tag types have the tag name and an input/output text box as their default visualization. The UserTypes and Arrays, will use a DataGrid by default.

You can customize that behavior at Unified Namespace Tags, using selecting the Tag and using the Map To Symbol button. (The result of that configuration is saved on the DefaultSymbol column)

> Check out our Link Tags to Symbol to see this feature in action.





Dropping Tags on Displays

The traditional way to setup HMI and SCADA screens was to drop graphical elements on the displays, and then go the configuration to bind Tags to those elements.

Our platform presents a newer, fasters and simpler way to do that, which is by directly dropping tags into the displays, and automatically the proper representation of tag is selected, using the Symbol to Tag configuration. No binding is required, and the symbol will be created on the display mapped to the selected tag.

There are various ways to start that operations:

1. At Unified Namespace Tags, select one or more rows with tag names, click COPY on toolbar (or Ctrl-C), go a displays and PASTE the tags on the display.
2. You can also do that copy and pasting from Excel, when doing from Excel, you can add the columns Top and Left to customize the position the tag/symbol will be inserted.
3. In the Drawing tool, right-click on the display to open the context menu, and use the Insert Tag command
4. In the Drawing area, select Tags on the Tree on the Right side and drag-drop the tags to the displays.

For advanced solutions, or large projects composed with many SCADA displays, it is possible to fully automate this process, using a .NET programming API (EngWrapper) that creates the displays automatically based on external files, or custom programming.

Linking Symbols To Tag Templates (UserTypes)

When using UserTypes together with Symbol Links, symbols can be designed to bind automatically to all template properties by receiving only the parent tag as a parameter. This eliminates manual binding and enforces a strict contract between the data model and the visualization.The symbol must reference the UserType through the extended HashTag syntax, explicitly isolating the template-based tag.

Symbol Link Mnemonic Pattern:

All internal bindings inside the symbol sould follow this pattern:

```
#TagValue:(TagNameWithTemplate).PropertyName
```

Where:

- TagValue is the only exposed symbol parameter
- TagNameWithTemplate is the parent tag implementing the UserType
- PropertyName is a member of the UserType

The parentheses define the replacement boundary, ensuring that only the parent tag reference is substituted when the symbol is instantiated.

No direct or fixed tag paths should exist inside the symbol.

Example

For a UserType named PID with properties Setpoint, PV, and CV, a symbol can internally reference:

```
#TagValue:(PID).Setpoint
#TagValue:(PID).PV
#TagValue:(PID).CV
```

The symbol exposes a single parameter:

- TagValue Tag of type PID

When instantiated, TagValue receives the parent tag (for example, Tag.Loop1), and all internal bindings resolve automatically.

Runtime Symbol Linking Behavior

Symbol linking is resolved by name matching between the Tag DataType and the Symbol:

- If a Symbol exists with the same name as the UserType, it is considered the default visualization.
- When a tag whose DataType matches the symbol name is dragged or pasted onto a display:
    - The system automatically instantiates the symbol.
    - The TagValue parameter is populated with the pasted tag.
    - All internal expressions are resolved without manual parameter mapping.

This mechanism establishes a one-to-one relationship between UserType (data contract) and Symbol (visual contract), ensuring reuse, consistency, and maintainability across the solution.

Default Symbol Mapping

If multiple symbols exist for the same UserType, a default symbol can be explicitly mapped in:

- Unified Namespace UserTypes Map to Symbol

Some internal UserTypes do not support custom symbol mapping.

Using the PlaceHolder Tag

When creating Symbols intended to use across multiple solutions, it is a better practice to use the Tag.PlaceHolder when creating the symbol.

The PlaceHolder Tag is a UserType with digital, integer, double and other Tag types, and it is defined by default in all solutions.

In this way you can create the symbol with the expressions like: #TagValue:Tag.PlaceHolder.IntegerTag, or #TagValue:(Tag.PlaceHolder.DoubleTag).Quality

This approach has various benefits:

- The symbol isn't dependent of any tag in the solution
- The PlaceHolder selected informs which tag type is expected to that symbol
- You can access tag properties, with the intelisense when creating more advanced symbols.

Linking Symbols with Tag Properties

When defining Symbol Parameters, in a similar way used on UserTypes, the parenthesis can be used to define various dynamics properties, using fields (properties) of a Tag, and having only the main tag name as parameter key.

For example, if you want to show the Tag Value, Quality and Timestamp, of a Tag, instead of definition 3 parameters, you can define one parameter to the TagValue, and access the fields of that main tag.

- Expression: #TagValue:(Tag.Tag1).Value
- Expression: #TagValue:(Tag.Tag1).Quality
- Expression: #TagValue:(Tag.Tag1).Timestamp

When creating the symbol, the parameter key #TagValue will be created, allowing the instances of the Symbol to replace Tag.Tag1 to another tag in the application.

Mapping Symbols to UserTypes

When mapping Symbols to UserTypes (formerly DataTemplates), the binding is performed at the template level, not at the level of individual members. Faceplate symbols are associated with tags of that UserType (for example, *PID*), not with specific properties such as *Setpoint*, *PV*, or *CV*.

If a symbol needs to bind to a specific member instead of the template tag, use the standard tag syntax without parentheses:

- #PV:Tag.Loop1.PV
- #CV:Tag.Loop1.CV
- #SetPoint:Tag.Loop1.Setpoint

Without parentheses, TagValue is expected to receive the final attribute or tag value, and the symbol no longer participates in template-based automatic symbol linking.

Managing the Solution Symbols

The symbols included in the Solution, are listed navigating to Displays Symbols.

At this page, you can export the symbol, delete, or verify Symbol Information, including Size and Description.

Save as an Image File

This command, creates a image (PNG) file with the Symbol Preview

Export to the Library

This command saves the symbols to the file Library.dbsln, located at the folder \Users\Public\Public Documents\<productName>\SystemSolutions.

This file is shared to be used by the Designer tool, when editing any solution in the computer.

Delete Unused Symbols

In order to delete a symbol, select the Row (or multiple rows), and use the delete (X) icon on the toolbar in top of the table.

The Symbol Control page allows the customization of Smart Symbols

Configuration

10.1.5 Authoring Changes

- **LockAspectRatio** — new property that decouples aspect-ratio locking from `LinkedSize`.
- **Has Legend** — new property that auto-creates a `TLegend` child following the symbol; available in the Symbol Configuration panel and the Make Symbol dialog.
- **LinkedSize auto-clear** — editing Width or Height in the Properties panel now automatically clears `LinkedSize` so manual size edits take effect immediately (previously they were silently lost).

Symbol Settings	
Field	**Description**
Is Linked With Library	When saving the Symbol it is saved to a repository shared by all displays in the solution. Meaning changing one time will affect the appearance and behavior or all displays using that symbol.
LinkedSize Proportional	When linked with the Solution Library, the fields specifies if the size of that Symbol instanced is independent or should change proportionally if the Symbol in the library has a new size. **10.1.5 change:** editing Width or Height in the Properties panel now automatically clears `LinkedSize` so the new dimensions take effect immediately. Previously, manual size edits were silently lost while `LinkedSize` remained true.
LockAspectRatio	*New in 10.1.5.* Decouples aspect-ratio locking from `LinkedSize`. Default `false`. When set to `true`, manual Width or Height edits in the Properties panel preserve the symbol's aspect ratio — changing one dimension scales the other proportionally.
Has Legend	*New in 10.1.5.* When checked, a `TLegend` label child is automatically created and follows the symbol on the display. Can be set from the Symbol Configuration panel and from the Make Symbol dialog. See Display Legends for the legend control reference.
Internal Dynamics	List of Dynamics properties used internal by the Symbol This is not to the list of Dynamics dialogs, that appears on the right side of the Settings dialog. The Dynamics configuration dialog are dynamics that are applied to the Symbol itself. the "Symbol Dynamics Message" show the internal dynamics used when the symbol was created.
Symbol Links	Maps the exposed properties of the Symbol to Tags or Expressions.
Copy and Paste	Button to copy the Symbol Links to the clipboard, or paste from it.
Apply	Apply the edited expressions to the Symbol. This action is optional, when closing the configuration dialog, or selecting another object the expression changes are automatically applied.
Appearance Wizard	Some symbols have an additional customization tools, see Symbol Wizards for more information.

In this section...