Dynamic Behavior Control Reference

Add Dynamic Behavior to the graphical elements.

Dynamic Behavior Controls (Reference), or Dynamics for short, enable to apply various dynamic behaviors on UI controls.

This guide addresses appearance settings, text input/output, hyperlinks, and security configurations. These features allow objects to change in appearance, position, or behavior in response to user actions, events, or changing conditions.

The Dynamic Properties feature allows connecting the properties of the graphical UI elements, such as color, position, and others, to the values of tags and other real-time values from the runtime namespaces. It also enables defining custom actions for mouse and keyboard interactions, input and output text, and executing scripts when the screen's UI element is enabled.

This flexible functionality enables you to create user interactions with the display that match your specifications. It's also used to create custom symbols, extending the built-in library.

Working the Dynamics and UI Elements
Using Expressions
Dynamics Types
- Action
- CodeBehind
- Shine
- Hyperlink
- Security
- Bargraph
- TextColor
- LineColor
- FillColor
- Visibility
- MoveDrag
- Scale
- Rotate
- Skew

Get, Apply, and Replace
Using Assets in Dynamics
Four-Family Rule for *Link Properties
- Composite-template form (text-display elements only)

Working the Dynamics and UI Elements

Dynamics allow you to configure real-time changes for an object's appearance, position, size, colors, the value the object reflects, user actions when a user clicks, and more. This dynamic behavior is configured by creating links between the object properties, tags, or other runtime properties for the solution.

In some systems, "animation" is used to refer to these dynamic runtime changes. In the platform, we have elected to use the term "Dynamics" to provide a clear distinction from the WPF animation features. WPF animations refer to changes that occur in a graphic's object properties when running the displays. This happens when timers and other object statuses are used to guide the animation, rather than real-time database values.

When you double-click an object, click the Dynamics button — or select it from the right-click context menu — to show a list of compatible dynamics that can be applied to the selected object.

Using Expressions

Expressions can be used in various places on Dynamics using VB.NET syntax to determine values or set conditions. They are compiled using the standard VB.Net compiler and accept everything that would be accepted in the code editor. The syntax is similar to C#, but does not require a semicolon at the end. Both VB.Net and C# programmers can use expression fields without any issues. In expressions, it is not necessary to prefix tag names with "@" since there are no local .NET variables, and the solution object is used directly. Arithmetic operands use standard .NET operands.

To allow single-line evaluations, the .NET language has the IIF command, which currently is used only with VB.Net. The IIF command has three parameters.

Example

```
IIF (A, B, C)
```

The first parameter is a condition. This method will return B if condition A is true, and it returns C if the condition is false.

Example

`IIF (tag.A = 1, "True", "False")` will return the string "True" if the value of tag.A is 1, or "False" if tag.A has a different value.

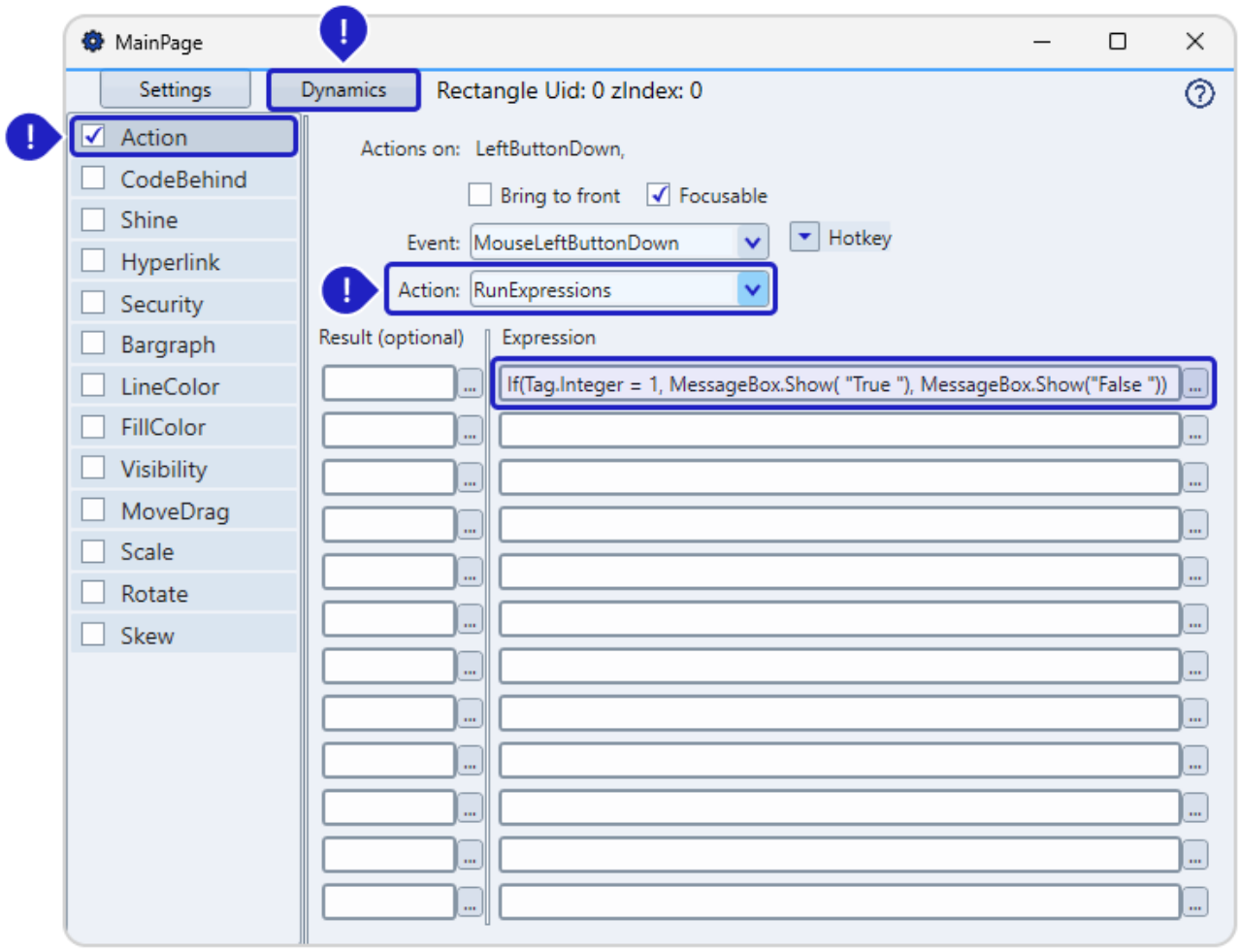
In this .NET method, all three parameters are evaluated independent of the condition. For instance, if you have `IIF (tag.A = 1, script.class.client.Func1(), script.class.client.Func2())`, both Func1 and Func2 will always be executed. Only the return value will be different based on the value of tag.A.

Run Expression Example IF:

In the example below, we have used an IF statement for its implementation. In this case, when the Tag equals 1, a Messagebox containing the word "True" will appear on the screen. Otherwise, if the Tag contains any other value, when the button is pressed, an action will trigger the event and a second Messagebox will appear on the screen containing the word "False".

Expression example:

```
If(Tag.Integer = 1, MessageBox.Show( "True "), MessageBox.Show("False "))
```



For more information, see Scripts Expressions.

Dynamics Types

Action

Dynamic Action allows to define actions that are triggered by user interactions event, such as clicks or keyboard inputs.

Action Dynamic Configuration	
Event	Choose events for mouse's actions. More than one event can be selected for each action. For example, you can set one action for MouseLeftButtonDown and another action for MouseLeftButtonUp. **MouseLeftButtonDownRepeat**: Runs its action repeatedly while the left mouse button is held down on the element. It does not run on the press itself: the first repeat runs about 100 ms after the press or, when a MouseLeftButtonDown action is configured, about 100 ms after that action finishes. The action then repeats about every 100 ms until the button is released. The 100 ms interval is fixed and cannot be configured. *Applies to FrameworX 10.1.6 and later.* On the Windows desktop clients (Rich Client and Smart Client), a repeat runs only while the pointer is over the element. On the Web (HTML5) client, the action also keeps repeating while the pointer is off the element. Release the button over the element to stop the repeat. *Applies to FrameworX 10.1.6 and later.*

Action	Select an action for the given event: **None**: No action. **SetValue**: Sets the value of the object. Object: The object that will receive the value. The object can be a tag or a writable property of a runtime object, for example an alarm area's acknowledge-all command, `@Alarm.Area.Production.AckAll`. *Applies to FrameworX 10.1.6 and later.* Value: The value that will be passed to the object. Format per the four-family rule for `*Link` / expression properties (see section below): numeric literal (bare: `"0"`, `"1"`, `"100"`), boolean literal (bare: `"true"`, `"false"`), tag/expression reference (@ prefix: `"@Tag.X"` or `"@Tag.A == 1"`), or string literal (escape-wrapped: `"\"Dark\""`). A bare alphabetic identifier without @ or quote-wrap parses as a tag reference, fails resolution, and the action silently no-ops with no error logged. **Toggle**: Toggles the object value. If the object's current value is zero, the value will be 1. If the object's current value is different from zero, the value will be 0. Object: The object that will be toggled. The object can be a tag or a writable property of a runtime object. *Applies to FrameworX 10.1.6 and later.* Use Toggle for a change-triggered command such as `@Alarm.Area.Production.AckAll`. AckAll acknowledges every unacknowledged alarm in the area and its child areas when its value changes, and the product does not reset it afterwards. Writing the same value again does nothing, so a SetValue that always writes the same constant, such as 1, acknowledges once and then does nothing on later clicks. Toggle changes the value on every click, so every click acknowledges. Any write of a value that differs from the current one also acknowledges. See Alarms Areas Reference. *Applies to FrameworX 10.1.6 and later.* **OpenDisplay**: Opens a display. Display: The name of the display that will be opened. **CloseDisplay**: Closes a display. Display: The name of the display that will be closed. **OpenLayout**: Opens a layout. Layout: The name of the layout that will be opened. **RunScript**: Runs a script that is placed in the display CodeBehind tab. Do one of the following: Enter the new method name and click New. Select one of the existing methods in the comboBox. **RunExpressions**: Runs the given expression. Expression: Enter the expression. For example: Tag.a + 1, or Tag.a + Tag.b, or Math.Cos(Tag.angle) * Math.PI. Result (optional): Enter the tag or the property that will receive the expression value.

Hotkeys

Note: You can configure hotkeys on Portable displays, but they are only available in Windows Clients due to browser limitations.

CodeBehind

The CodeBehind dynamic runs an action on a Display lifecycle event — when the Display opens, while it stays open, or when it closes. It does not react to the mouse: for an action triggered by a click, use the Action dynamic above.

CodeBehind Dynamic Configuration	
Event	Each of the three lifecycle events carries its own action. Configure any combination of them. **OnDisplayOpening**: Runs once, when the Display opens. Use it for initialization. **OnDisplayIsOpen**: Runs repeatedly while the Display stays open — on every tick of the Display's IsOpen timer, whose period is the Display's `IsOpenInterval` setting, in milliseconds. Keep actions on this event light: at a short interval they run several times per second. **OnDisplayClosing**: Runs once, when the Display closes. Use it for cleanup.

Action	Select an action for the given event: **None**: No action. **SetValue**: Sets the value of the object. Object: The object that will receive the value. Value: The value that will be passed to the object. Format per the four-family rule for `*Link` / expression properties (see section below): numeric literal (bare: `"0"`, `"1"`, `"100"`), boolean literal (bare: `"true"`, `"false"`), tag/expression reference (@ prefix: `"@Tag.X"` or `"@Tag.A == 1"`), or string literal (escape-wrapped: `"\"Dark\""`). A bare alphabetic identifier without @ or quote-wrap parses as a tag reference, fails resolution, and the action silently no-ops with no error logged. **ToggleValue**: Toggles the object value. If the object's current value is zero, the value will be 1. If the object's current value is different from zero, the value will be 0. Object: The object that will be toggled. **RunExpressions**: Runs the given expression. Expression: Enter the expression. For example: Tag.a + 1, or Tag.a + Tag.b, or Math.Cos(Tag.angle) * Math.PI. Result (optional): Enter the tag or the property that will receive the expression value. **ChatRequest**, **ChatClear**, **ChatCompress**: The Local AI chat actions — see ChatRequest Action Reference, ChatClear Action Reference and ChatCompress Action Reference. `ChatCopyToTag` is available on the Action dynamic only. **Known limitation.** In a CodeBehind dynamic, a chat action executes but its reply does not return: the Return Value tag is not written and the Expressions rows do not evaluate. Use the Action dynamic when a chat action needs a Return Value or Expressions. No target release is committed. **ServerRequest**: Calls a server-side Script method. Object: The Script method to call. **HTTPRequest**: Runs a Report WebData request. Object: The WebData object to run. HTTP method: GET or POST.

Shine

Shine Dynamic Configuration enables dynamic object appearance changes with various features such as mouse-over and selection effects, scale adjustments, and reference position settings.

Shine Dynamic Configuration	
IsMouseOver	Enter a tag that will receive the OverValue or the NotOverValue.
OverValue	The IsMouseOver value when the mouse is over the object.
NotOverValue	The IsMouseOver value when the mouse is not over the object.
Mouse Over Appearance	The object appearance when the mouse is over it. **Opacity**: The object opacity (0 = transparent, 1 = opaque). **Scale**: The object size (0.5 = half, 1 = the same size, 1.5 = one and a half, 2 = double size). **OuterGlow**: Defines the OuterGlow color, the check box enables or disables it. **TextColor**: Defines the text color, the check box enables or disables it.
Mouse Not Over Appearance	The object appearance when the mouse is Not over it: **Opacity**: The object opacity (0 = transparent, 1 = opaque). **Scale**: The object size (0.5 = half, 1 = the same size, 1.5 = one and a half, 2 = double size).

Is Selected Appearance	The object appearance when it is selected: **IsSelected**: Defines whether the object is selected. **Opacity**: The object opacity (0 = transparent, 1 = opaque). **Scale**: The object size (0.5 = half, 1 = the same size, 1.5 = one and a half, 2 = double size).
Scale Reference	Center Left Up Right Down
ShowUid	Set to show the Uid of the object on hover
Tooltip	Set to show as a tooltip on hover

Hyperlink

The Hyperlink dynamic configuration enables opening links of various types, while the Security configuration defines runtime permissions, including Esign requirements, confirmation messages, and access restrictions for specific permission groups.

Hyperlink Dynamic Configuration	
HyperLinkType	Selects the hyperlink type: • http • ftp • file • mailto • telnet
Url	Sets the URL to open.
Hotkey	Ctrl Shift Alt OS Key Keyboard buttons

Security

Dynamic Security configuration is a feature that allows you to set object runtime permissions to execute actions. This configuration includes a Disable setting, where you can enter a tag, a property, or an expression that returns a value. If the resulting value is greater than zero, the object will be disabled. The Verify Permissions setting allows you to select specific permission groups that can access the object, providing an additional layer of security. The Confirm Message setting enables you to show a dialog box with a message before taking any action, and the ESign Required setting det ermines whether an electronic signature is required before the action is executed. This feature enhances the security of your solution and ensures that actions are executed only by authorized users.

Security Dynamic Configuration	
Disable	Enters a tag, a property, or an expression returning a value. If the resulting value is: **Zero**: Object will be enabled. **Greater than zero**: Object will be disabled.
Verify Permissions	When selected, only the chosen permission groups can access the object.
Confirm Message	Shows a confirmation dialog before taking some action: **textBox**: Enter the message that will appear in the dialog. **checkBox**: Enables or Disables the Confirm Message.
ESign Required	When enabled, the software opens a dialog requiring authentication before executing the action linked to that object.

Bargraph

The Bargraph Dynamic Configuration creates dynamic bar graphs by defining an expression for the value, setting the fill percentage range, color, and orientation. It is useful for displaying data in a visually appealing and easy-to-understand way.

Bargraph Dynamic Configuration	
Expression	The value used for the bargraph dynamic.
Value Range	The minimum and maximum values that will correspond to the minimum and maximum fill percentage.
Fill (%)	The minimum and maximum bargraph fill percentage.
Bar Color	The bar graph color.
Orientation	The bar graph orientation: to up horizontal center to down to right vertical center to left

TextColor

- **TextColor -** Changes the object text color dynamically.

A dynamic configuration that changes an object's text color and can be customized in real-time using expressions and valid color names/hexadecimal codes, providing a flexible way to adjust the appearance of objects' text.

LineColor

- **LineColor -** Changes the object line color dynamically.

A dynamic configuration that changes an object's line color and can be customized in real-time using expressions and valid color names/hexadecimal codes, providing a flexible way to adjust the appearance of lines in objects.

FillColor

- **FillColor -** Changes the object fill color dynamically.

A dynamic configuration that changes an object's fill color and can be customized in real-time using expressions and valid color names/hexadecimal codes, providing a flexible way to adjust the appearance of objects' fill.

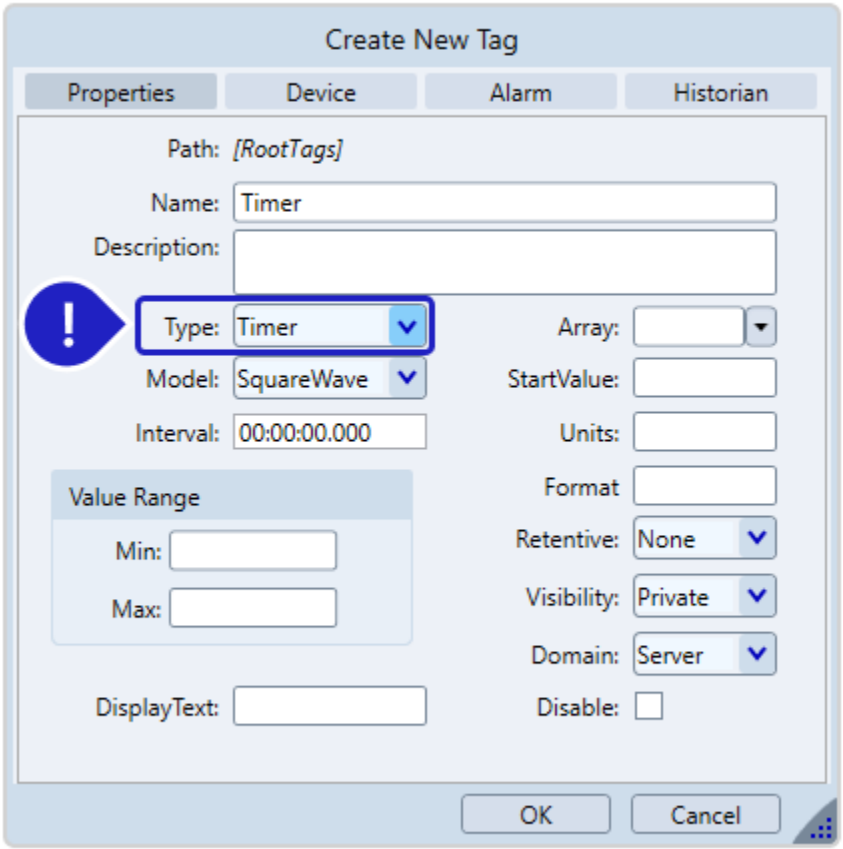
Text, Line and Fill Common Properties	
Expression	The value used for the dynamic.
ChangeColor	**UsingLimits**: The resulting color is determined when the value is equal or higher than one of the limits. From FrameworX 10.1.6, when you build displays in JSON or with the AI tools, a limit color accepts a named theme color, such as `StateAlarm` or `StateGreen`, in place of a fixed color. The color then follows the active theme, so machine state colors change with the rest of the display when the theme changes. Set the `LimitColorTheme` field of the limit, for example `"LimitColorTheme": "StateAlarm"`, or write the name in `LimitColor`, plain or with the `theme:` prefix. A fixed color value keeps working as before. For the theme color names, see Displays Themes Reference. **AbsoluteValue**: The color will be the expression value. The value must be a valid color name or an eight-digit hexadecimal color. For example: "White" or "#FFFFFFFF". In a symbol, write a parameter in the Expression, such as `#Color:"Red"`, so each instance sets its own color; see Color as a Parameter. **ImageWebUrl**:
BadQuality	Selects color when the related tag quality is bad.

Undefined Quality	Selects color when the related tag quality is undefined.

FillColor

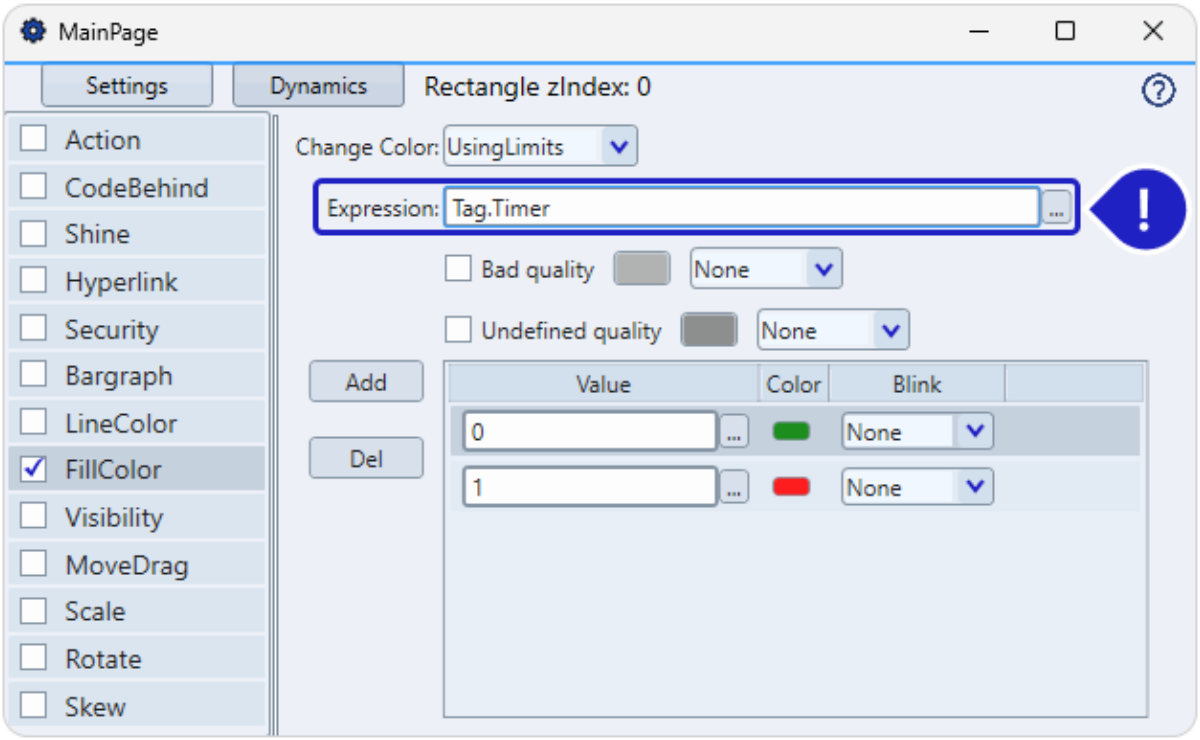
It is possible to configure a solution so that an object blinks only for a pre-determined period, without using scripts or expressions.

To do this, the "Expression" field in FillColor needs to be filled with a Timer type Tag, which contains the following parameters:



Model: DelayOff Interval: <desired time for the object to blink>

In this way, the object configuration will be as follows:



Following the steps above, when the value of Tag.Timer is equal to 1, the object will blink during the defined interval, and at the end of this time, the Tag returns to 0 and the object stops blinking.

Visibility

The Visibility Dynamic Configuration changes an object's visibility and opacity dynamically by hiding or showing objects based on tags, properties, or expressions, setting opacity based on a range of values, and providing a security option for hiding components for certain users. It is useful for creating interactive and dynamic user interfaces.

Visibility Dynamic Configuration	
Visible	Enter a tag, a property, or an expression returning a value. If the resulting value is: **Zero**: Object will be hidden. **Greater than zero**: Object will be visible.
Hide when security is enabled	Hides the component when security is enabled for the current user.
Opacity	**Expression**: The value used to set the opacity. **Range**: The minimum and maximum values that will correspond to the minimum and maximum opacity. **Opacity**: The minimum and maximum opacity (0 - invisible, 0.5 - a bit transparent, 1 - opaque).

MoveDrag

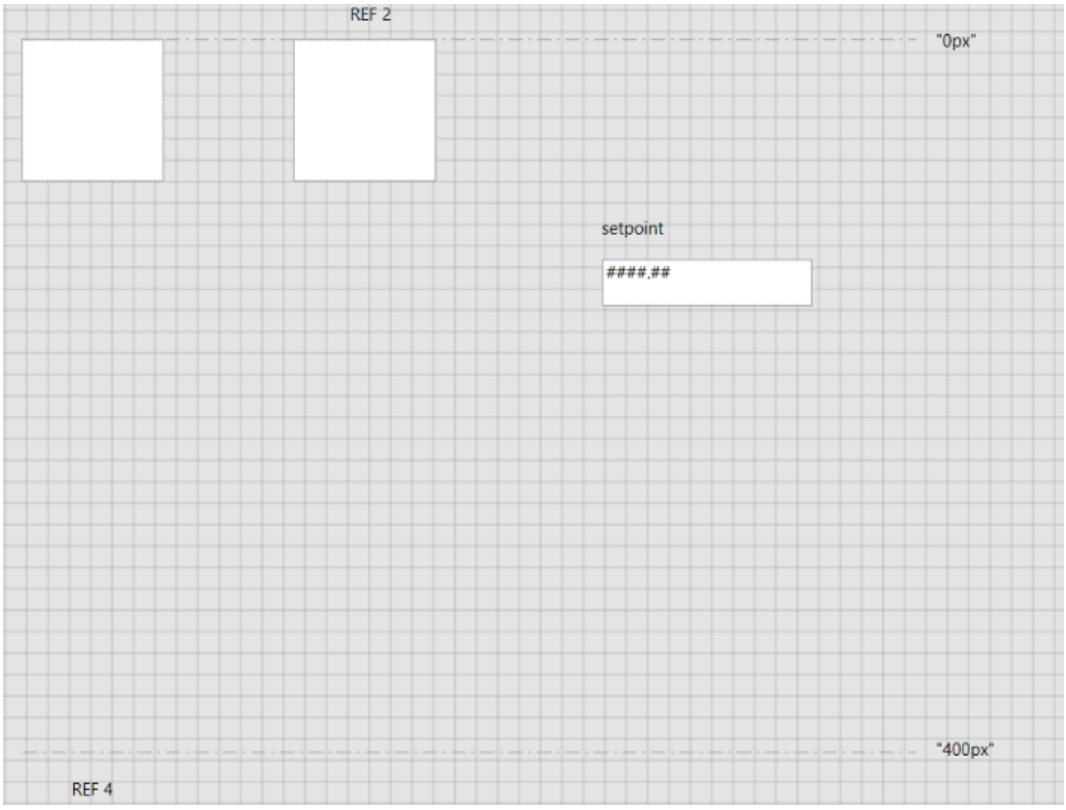
MoveDrag Dynamics is a feature for dynamically moving objects in a display, commonly used in interactive applications and user interfaces. It can be configured with different modes, expressions, and settings, and even use previous objects as a reference for position. This tool is powerful for creating dynamic and interactive solutions.

MoveDrag Dynamic Configuration

BindingMode	**TwoWay**: Input moving and output moving. **InputOnly**: Input moving only. The object does not move when the object value changes. **OutputOnly**: Output moving only. The object does not move with user interaction.
Horizontal Move	Expression with the horizontal move reference. **Range**: The minimum and maximum values that will correspond to the minimum and maximum horizontal position. **Position**: The minimum and maximum horizontal position.
Vertical Move	Expression with the vertical move range. **Range**: The minimum and maximum values that will correspond to the minimum and maximum vertical position. **Position**: The minimum and maximum vertical position.
Use previous object for position reference	Reference the current object's position based on the previous object

When using vertical movement dynamics, ensure that the **minRange** property is properly set. This configuration defines the minimum movement range for the control and may need to be adjusted based on specific requirements.

Moving Objects



Use the code below to move objects on the screen:

Update

```
public void Update (double SPMedidal)
{
        Ref2 = this.CurrentDisplay.GetControl("Ref2") as Rectangle;
        Ref4 = this.CurrentDisplay.GetControl("Ref4") as Rectangle;
        Thickness myThickness = new System.Windows.Thickness();

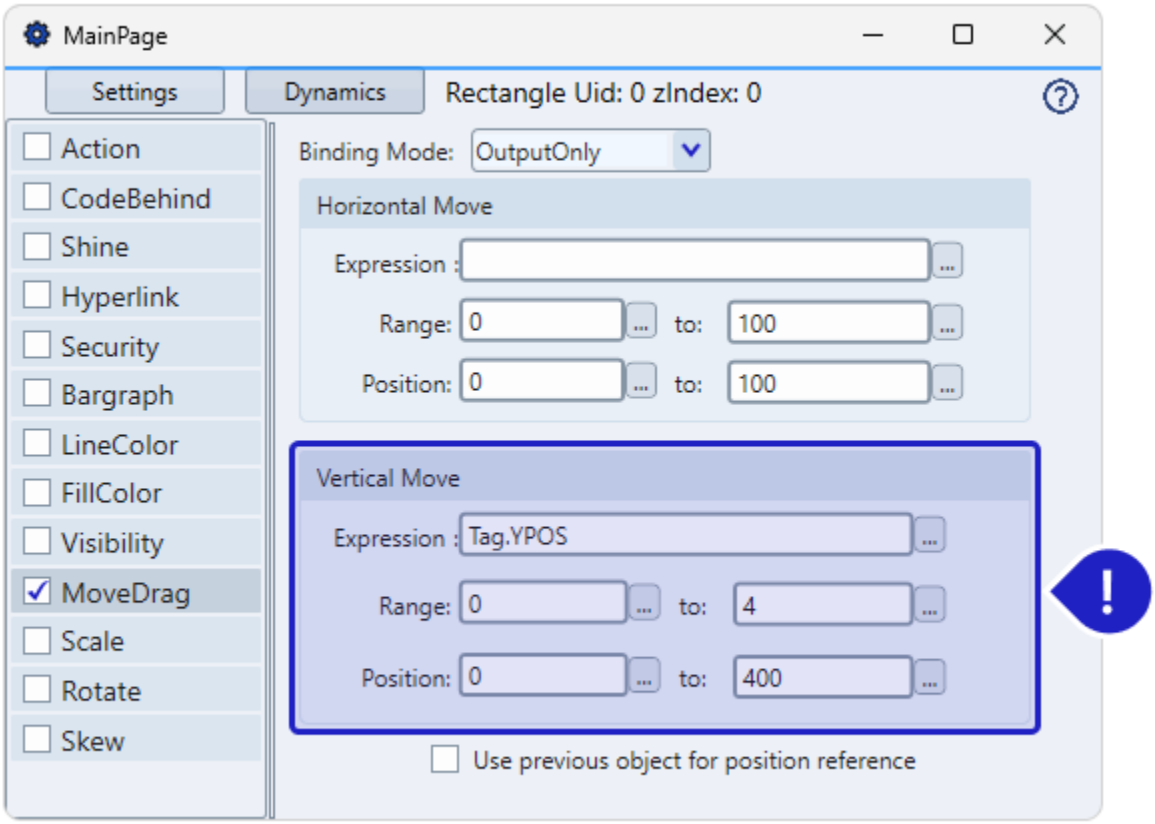
        myThickness.Top = SPMedidal;
        Ref2.Margin = myThickness;
        @Info.Trace("Position UP-DOWN-Ref2: "+Ref2.Margin.ToString());

        myThickness.Top = -SPMedidal+400;
        Ref4.Margin = myThickness;
        @Info.Trace("Position DOWN-UP-Ref4: "+Ref4.Margin.ToString());
}
```

We emphasize that the MoveDrag property is made for this type of application and can be used if desired.

Below is an example of how the MoveDrag property works.

The image below illustrates the Range and Position fields of the Vertical Move section:



The YPOS Tag added to the expression is set in the TextBox present on the screen.

The Position field represents the range of pixels in which the object can move on the screen.

The Range field represents the range of values that the Tag can assume.

Therefore, in the example shown in the image, the user can type a number from 0 to 4, which will be converted to the range of pixels between 0 and 400.

That is, if the user types 1 for the Tag value, the object will move 100px down. If they type 2, it will move 200px, 3 - 300px, and 4 - 400px.

This is just an example, you could leave both the Range and Position fields with equal values, such as 0 and 400 for both.

Then, the user would be allowed to type values between 0 and 400, being displaced exactly the value that was typed by the user, following a 1:1 ratio in this case while in the previous case a 1:100 ratio was being followed.

Scale

Scale Dynamic configuration enables dynamic resizing of an object with Width and Height Scale settings that can be specified as a percentage of minimum and maximum values. Scale Reference allows choosing the reference point for scaling. This configuration provides flexibility in designing and displaying visual elements by adjusting the object size based on changing values or conditions.

Scale Dynamic Configuration	
Width Scale	**Object**: the value used for the width scaling. **Range**: The minimum and maximum values that will correspond to the minimum and maximum width scale percentage. **Scale** (%): The minimum and maximum width scaling percentage.
Height Scale	**Object**: the value used for the height scaling. **Range**: The minimum and maximum values that will correspond to the minimum and maximum height scale percentage. **Scale** (%): The minimum and maximum height scaling percentage.
Scale Reference:	Center Left Up Right Down

Rotate

Rotate Dynamic configuration enables dynamic rotation of an object with user-defined settings, such as angle, speed, center reference, and offset values. An expression can be used to set the rotation value, and a value range can be defined for the minimum and maximum angle. The feature also includes an on/off property. It is useful for creating dynamic and interactive visual displays or animating objects in a graphical user interface.

Rotate Dynamic Configuration	
Expression	The value used for the rotation.
Value Range	The minimum and maximum values that will correspond to the minimum and maximum angle. For example: 0 to 100.
Angle	The minimum and maximum rotation angle. For example: 0 to 360.
RPM	Number of rotations per minute. The value can be defined on Expression.
Center Reference	**Clockwise**: Check if you want Clockwise. Center Left Up Right Down **Offset X**: Defines the distance of the center of the Rotation in X axis. **Offset Y**: Defines the distance of the center of the Rotation in Y axis.
ON/OFF	Enter with the object property that will turn ON/OFF the Rotation Dynamic

Skew

Skew Dynamic configuration enables dynamic skewing of an object by distorting its shape along the x or y-axis. The setting includes values for X-axis and Y-axis skewing, ranges, and angles. The Skew Reference setting allows choosing the center of the skewing. This feature adds interactivity to your design by creating dynamic visual effects.

Skew Dynamic Configuration

X-axis Skew	**ObjectValue**: the value used for the X-axis skewing. **Range**: The minimum and maximum values that will correspond to the minimum and maximum X-axis skewing angle. For example: 0 to 100. **Skew** (º): The minimum and maximum X-axis skewing angle. For example: 0 to 180º.
Y axis skew	**ObjectValue**: the value used for the Y-axis skewing. **Range**: The minimum and maximum values that will correspond to the minimum and maximum Y-axis skewing angle. For example: 0 to 100. **Skew** (º): The minimum and maximum Y-axis skewing angle. For example: 0 to 180º.
Skew Reference	Center Left Up Right Down

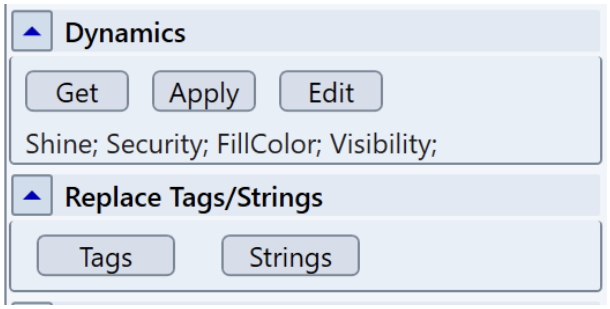
Get, Apply, and Replace

In the left panel, under Drawing Properties, you'll find the Dynamics options, which contain the **Get**, **Apply** and **Edit** buttons. **Edit** will simply open the Dynamics Configuration of the selected object, but **Get** and **Apply** allow you copy and paste dynamics' settings configuration from one object to another.

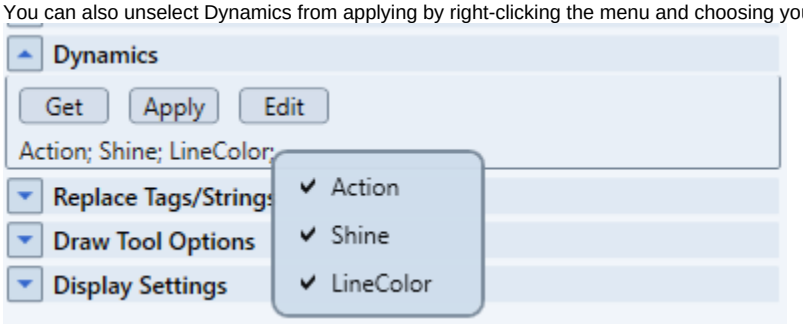
Select the object with the dynamics you want to use, and click **Get**. It'll show all the dynamics you have configured below the buttons and that are ready to be applied to other objects. Select one or more target objects and click **Apply** and all the configured dynamics will be copied.

Below the Dynamics, in the left panel, you'll find the Replace Tags/String options.

You can use the **Replace** button to change the tags in the dynamics of the selected objects. The **String** button is used to change all of the strings for the selected objects.



You can also unselect Dynamics from applying by right-clicking the menu and choosing your preferred Dynamics.

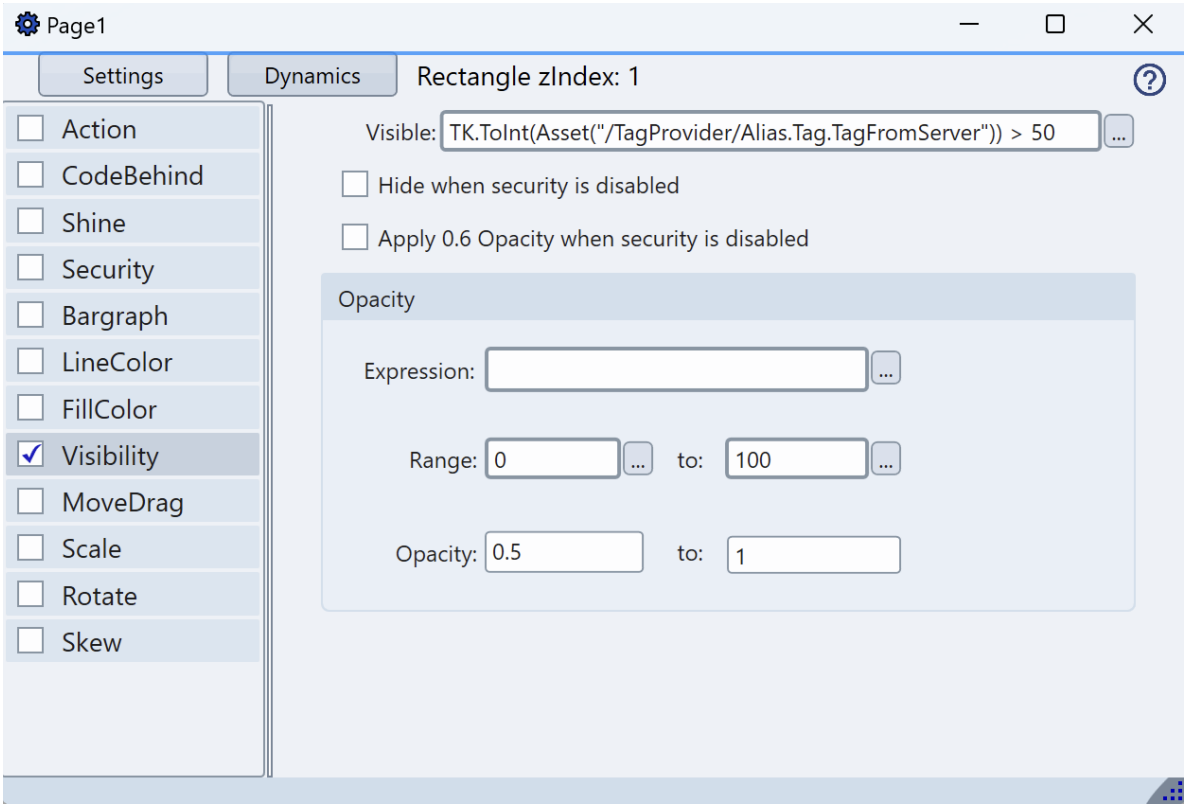


Using Assets in Dynamics

In Portable/HTML5 displays, Assets cannot be treated as dynamic objects due to limitations in NetStandard 2.0. To use them in Dynamics (as values or in expressions), you must first convert them to the required data type (e.g., integer, string, digital, DataTable) using methods such as `TK.ToInt`, `TK.ToString`, etc.

In this example, we converted the Asset to an integer using TK.ToInT to determine if the object will be visible.

In WPF-only displays, which are based on .NET Framework 4.8, Asset calls return values that can be handled dynamically, without the need to explicitly define the data type.



Four-Family Rule for *Link Properties

Every `StringWithExpressions`-typed property — `LinkedValue`, `ObjectValueLink`, `VisibleLink`, `DisableLink`, `LabelLink`, `CaptionLink`, and every other `*Link` field on Action, CodeBehind, Visibility, FillColor, LineColor, TextColor, Rotate, MoveDrag, Scale, Bargraph, Skew, Hyperlink, and Security dynamics — accepts four value families. Pick by intent.

Family	Form	Example	Notes
Numeric literal	Bare digits, no quote-wrap, no `@`	`"1"`, `"100"`, `"3.14"`, `"-1"`	Parser routes to numeric-literal branch.
Boolean literal	Bare keyword	`"true"`, `"false"`	Same parser branch as numeric.
Tag / expression reference	`@` prefix mandatory	`"@Tag.Setpoint"`, `"@Tag.Mode == 1"`	Resolves at runtime against the live object tree. Without `@`, parses as a literal identifier and fails resolution silently.
String literal	Escape-wrapped quotes	`"\"Dark\""` — writes the literal four-character string `Dark`	The escape-wrap is JSON-source. After parse, the in-memory string is `"Dark"` (six characters including the surrounding quotes); the runtime parser sees the inner quotes and recognizes a string-constant token. A bare `"Dark"` (no `@`, no quote-wrap) parses as a tag/object reference, fails resolution, and the SetValue silently no-ops.

Composite-template form (text-display elements only)

On `LinkedValue`, `LabelLink`, and `CaptionLink` for text-display elements (TextBlock, Label, Button) — for labeled-value rendering — the property additionally accepts a composite-template form: `"Temperature: {@Tag.Reactor/Temperature_C} °C"`. Literal text outside braces, `@Tag.X` references inside braces with the `@` prefix MANDATORY inside the braces. A bare `{Tag.X}` without `@` renders as literal text, not as a binding.

For TextBlock specifically: emit the runtime binding on the `LinkedValue` property, NOT on the `Text` property. `Text` is design-time fallback only and does NOT interpolate `{...}` or live-bind `@Tag.X`.

In this section...
