SIEMENS

Ingenuity for life

Using dynamic SVGs with SIMATIC WinCC

SIMATIC WinCC

https://support.industry.siemens.com/cs/ww/en/view/109782045

Siemens Industry Online Support

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit **Fehler! Linkreferenz ungültig.**.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: **Fehler! Linkreferenz ungültig.**.

Table of contents

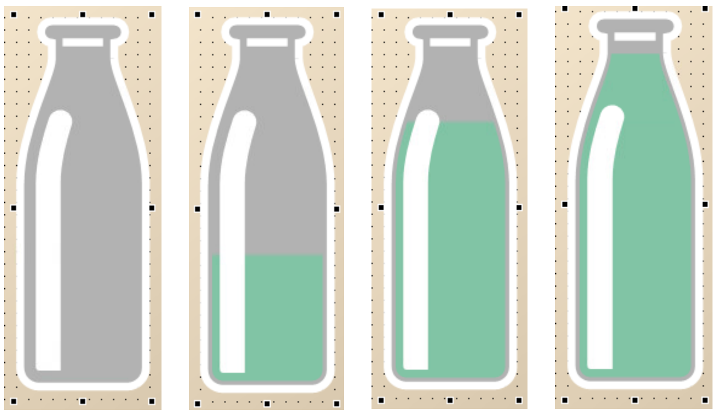
1 Introduction

1.1 Overview

SIMATIC WinCC supports SVG graphics for creating your operator screens. SVG is a descriptive graphics format. This means that a code describes how an image looks.

In conjunction with WinCC and the SVG code, you can alter the graphics during the runtime, for example simple color changes or movement such as rotation.

Figure 1-1



This application example will show you how to create dynamic SVGs for use in SIMATIC WinCC.

Structure of this documentation:

- Basics
- Structure and use in WinCC, special considerations and limitations
- Dynamization and code examples

1.2 Components used

Note

This application example pertains generally to SIMATIC WinCC without regard to version.

Before starting configuration, please check the approved and supported scope of your WinCC version.

The supported scope of SVG functions varies with different WinCC versions; it can be found in the corresponding documentation.

This application example consists of the following components:

Table 1-1

Components	File name	Note
Documentation	109782045_WinCC_and_SVG_DOC_en.pdf	This document

2 Useful information

2.1 Basics

This application example essentially distinguishes between two graphics formats.

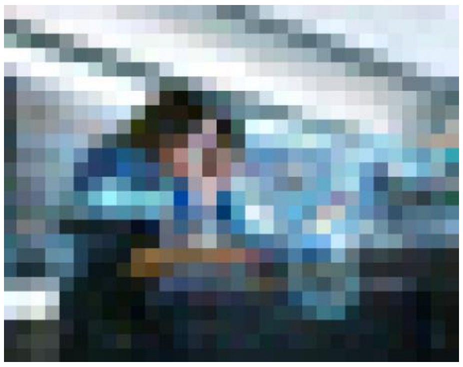
- Pixel-based graphics formats
- Vector-based graphics formats

Pixel-based graphics formats

These graphics formats represent the image data in a grid or coordinate system. Each point in the image in represented with its position and color information.

This information can be represented well in a picture or image whose resolution has been heavily reduced. The table below depicts the same image. The first one is at reduced resolution, where the grid and individual color data are easy to see. The second image is at a high resolution, where the grid and color information are not apparent.

Table 2-1

Heavily reduced resolution	High resolution
Position and color information easy to recognize.	The depiction follows the same principle, but the resolution is high. The eye can recognize the individual positions and color data as a "complete image".
	

These graphics formats are well suited for static images.

Vector-based graphics formats

These graphics formats represent image information descriptively. Analogously: "Draw a circle with radius x in the center of the picture and fill the circle with color."

The image information described in this manner present the following advantages:

- Scalability
- Editability

Scalability of vector-based graphics formats

Thanks to the descriptive information, the graphic is independent of any resolution it may be displayed at later.

It will not matter whether it appears on a high-resolution screen or a low-resolution screen. There will always be a circle in the middle of the picture.

Editability of vector-based graphics formats

The descriptive information permits a simple modification which, for example, only fills half the circle with color. This can be used to display a fill level, for instance.

This characteristic in particular is the subject of this application example.

SVG as a vector-based graphics format

SVG is a descriptive, XML-based file format. SVG is easily accessible for editing/later modification.

SVG graphics can, for example, be created directly with a text editor; or they are produced in graphics/design programs such as Inkscape, Adobe illustrator, and others.

The recommended file extension is .svg. The MIME type is "image/svg+xml".

2.2 Structure

As an XML document, an SVG is built in a tree structure from different elements and the attributes assigned to those elements.

An SVG file can begin with the XML declaration, as is common in XML-based languages. This can be followed by processing instructions, such as instructions to reference external style templates.

As with all XML documents, this header is followed by the root element, which has the name "svg" for SVG documents. In order to uniquely assign the element and its contents to the SVG namespace and thereby give it a defined meaning relating to the SVG recommendations, the namespace is annotated in the root element with the XML attribute construction xmlns.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.1" baseProfile="full"
    width="800" height="600"
    viewBox="-400 -300 800 600">
    <title>Titel der Datei</title>
    <desc>Beschreibung/Textalternative zum Inhalt.</desc>

<!--Inhalt der Datei -->

</svg>
```

The attributes `width` and `height` of the SVG element define the size of the graphic. Set both values to ensure the image is displayed properly in WinCC.

The `viewBox` attribute can be used to specify which area of the draw plane in the display area the image should appear in. In some cases, this area may be transformed to the display area. If the attribute is not specified, the area will be covered by the display area; no transformation will occur.

Note Do not use the `viewBox` attribute with dynamic SVGs. This would cause the line thicknesses to scale by a factor of two in graphics that are doubled in scale.

2.3 Coordinate system, coordinate specifications

The initial coordinate system has its origin in the upper left corner of the draw area. This is an internal, dimensionless coordinate system in which the X-axis points to the right and the Y-axis points downward. This coordinate system is dimensioned for output with the attributes `width` and `height`.

If using the `viewBox` attribute (and with most descendants of the root element), as well as when using the `transform` attribute, the resultant existing coordinate system will differ from the initial coordinate system.

All relative and absolute length numbers within the graphic will be referenced to this current coordinate system.

Unlike whole number pixel coordinates in raster graphics (e.g. JPEG, PNG, BMP), SVG coordinates are floating point numbers.

2.4 Requirements for use in SIMATIC WinCC

In order to use .svg-type vector graphics in SIMATIC WinCC as dynamic graphics, it is important to remember a few things. These features and particularities are described in this chapter.

2.4.1 SVG structure for SIMATIC WinCC

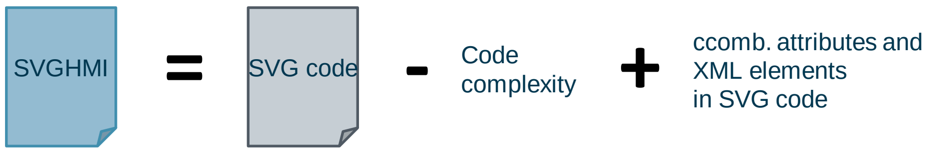
WinCC needs additional information to dynamize an SVG graphic.

Example

A rectangle must be filled with a color to display the fill level of a tank. WinCC needs the information that the graphic can be dynamized with a property as well as how this property is interpreted.

WinCC utilizes a specific "SVGHMI" format which contains this information.

Figure 2-1



Properties of SVGHMI

In order to be recognized by SIMATIC WinCC as a dynamic SVG, the file ending ".SVGHMI" must always be used:

The MIME type is always identical to standard SVGs: image/svg+xml.

Unlike HTML, SVG is case-sensitive.

While SVG itself does not prescribe any dedicated character encoding, the underlying XML standard allows any arbitrary encoding.

This means that SVGs can be very complex. The complexity has been reduced with the specifically designed SVGHMI type. SVGHMI files can only be encoded in UTF-8.

Within the document itself, an SVGHMI object defines its own XML document type declaration in order to allow the distinction from standard SVG profiles and to identify the concrete version of the specification for the rendering template.

".SVGHMI files" can no longer be opened with SVG graphics editors.

The following declaration must be placed over every SVGHMI object document.

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
```

These have been supplemented with extensions so that a dynamization of the SVGHMI object is possible.

These extensions include, among other things, a binding mechanism which makes it possible to access properties of the Screen Object Model, as well as to react to the control flow for the rendering and user interactions.

Every SVGHMI file must specify the SVG namespace as defined by W3C as an attribute for the root SVG element. In other words,

```
<svg xmlns="http://www.w3.org/2000/svg"
```

in order to maintains the function and conformity with the SVG specification, all additional XML elements and attributes are preceded by the following namespaces:

- hmi: This namespace contains user-defined elements and attributes of this specification.
  They typically allow for logic to be introduced, while the visual elements lie within the namespace mentioned below.
- hmi-bind: This namespace is used for adding expressions and defining data connections. The concrete attribute names used after the prefix are not defined.
  Instead, any valid SVG attribute can be placed.

- hmi:self: This namespace is used for defining the interface between dynamic SVG and TIA Portal.

  hmi:localdef: This namespace is used to define temporary tags within the dynamic SVG.

Important: Normally defined in XML namespaces. This means that the URL in the value says what it is; you can choose the namespace yourself.

Example: xmlns:banane="http://www.w3.org/1999/xlink"

The example is not valid: "banane" must be used in place of "xlink".

However: The HMI namespaces defined in WinCC behave exactly the opposite.

The value of the URL is irrelevant, but the names of the namespaces (e.g. hmi, hmi-bind, etc.) are fixed and cannot be changed.

The complete document header for an SVGHMI object must always appear and be structured as follows. In order to define an initial aspect ratio, you still have to adjust the size of the viewbox to suit the object being displayed. If you do not wish to define a fixed aspect ratio, it is sufficient to define the size of the graphic by specifying the width and height parameters. In this example we will use one viewbox for the sake of simplicity:

Note In operation, the runtime will thus scale the display area of the SVG object.

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<!-- SIMATIC WINCC Copyright (C) Siemens AG 2017. All Rights
Reserved. -->

<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:hmi-bind--xlink="http://svg.siemens.com/hmi/bind/xlink/"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     xmlns:hmi-element="http://svg.siemens.com/hmi/element/"
     xmlns:hmi-feature="http://svg.siemens.com/hmi/feature/"
     xmlns:hmi-event="http://svg.siemens.com/hmi/event/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
<hmi:self type="widget" displayName="DefaultName"
name="extended.DefaultName"
</svg>
```

2.4.2 General structure shown with an example

The structure of a dynamic SVG is composed of the following order:

Document type declaration

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
```

Definition of the dynamic SVG

In order to uniquely assign the element and its contents to the SVG namespace and thereby give it a defined meaning, the namespace is annotated in the root element with the XML attribute construction "xmlns".
If you create a dynamic SVG, it must contain the code shown here.
Only the "viewBox" can be modified on the respective SVG. "preserveAspectRatio" must be "none".

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

Definition of the interface between dynamic SVG and WinCC

Type="widget" means that the graphic is permitted to be hosted in a HmiCustomWidgetContainer. This is a rule for dynamic SVGs.

"displayName" and "Name" must be given a unique name that describes the dynamic SVG. For "name", an "extended" precedes the name. (must be set).

The version of the dynamic SVG can be specified in "version".

In the "hmi:paramDef" area, an interface of the dynamic SVG is defined for WinCC.

"name" specifies the name of the interface. It can be selected freely. However, it is recommended to use recurring names. Example names can be found in chapter 4 Listing of example properties.

You will also find the possible types of interface listed in this chapter.

"default" specified the value of the interface that is described when the interface is not interconnected on the HMI side.

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1" >
     <hmi:paramDef name="BasicColor"              type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"           type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"                type="number"
    default="0"  />
</hmi:self>
```

Definition of local tags (temporary tags)

"name" and "type" of local tags behave the same as in the interface.

"hmi-bind:value" specifies how the value of the local tag should be calculated.

Chapter 3.11 Converter commands describes which converter can be used for this purpose.

```
<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>
```

Description of the graphics elements

Code for the individual graphics elements is located here. If a property of an element must be dynamized, then write "hmi-bind:" before the property. The quotation marks then contain how the value of the property shall be determined. Chapter 3.11 Converter commands can be used for this purpose.

```
<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50"
hmi-bind:transform=" translate({{0 +
LocalProps.NormalizedValue*100}})" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
```

End of the SVG file

```
</svg>
```

2.4.3 Elements supported in SIMATIC WinCC

Only the following SVG elements are permitted for the SVGHMI objects.

Table 2-2

Element	Description
`circle`	Form element for a circle.
`circle-path`	Element that bounds the area that color can be applied to.
`defs`	Element used as a section for elements that will be reused (e.g. due to the usage element).
`ellipse`	Form element for an ellipse.
`filter`	Container element to which some SVG elements refer for atom filter operations.
`fecolormatrix`	Filter primitive for modifying colors on the basis of a matrix transformation.
`fecomposite`	Filter primitive for pixelwise combination of two input images.
`fegaussianblur`	Filter primitive for blurring an input image.

Element	Description
`feoffset`	Filter primitive for offsetting the complete input image.
`g`	Structure element for grouping multiple elements. Transformations applied to this element will be performed for all contained child elements. This element is also used for creating a decorated plane.
`image`	Element for integrating images into the visualization. Supported formats are PNG, JPEG and SVG files that do not exceed the limits of this document.
`line`	Form element for a simple line.
`lienargradient`	Definition of linear gradients to be applied to forms or graphical elements.
`mask`	Element that uses an arbitrary form, a graphical element or a group element as an alpha mask for compositing.
`marker`	Element that defines a graphic that will be used as a marking on path, line, polyline and polygon.
`path`	Generic form element for creating base forms.
`pattern`	Element that defines a graphic and serves to fill or paint a form or graphical element with an object which can be replicated.
`polygon`	Form element for a polygon.
`polyline`	Form element for a polyline.
`radialgradient`	Definition of radial gradients to be applied to forms or graphical elements.
`rect`	Form element for a rectangle.
`svg`	Container element for nesting independent fragments, with its own view window and coordinate system.
`symbol`	Undrawn grouping element to be used as a template to which usage elements refer.
`text`	Graphics element consisting of text.
`tref`	A reference object that makes it possible to refer to a text within a defs section.
`tspan`	Structure element that is used within a text element to modify properties of text, script and position.
`use`	Element that references SVG notes from the same document and duplicates them.

2.4.4 Functions not supported by SIMATIC WinCC

The following functions within SVGHMI objects are not supported.

Styling

Elements in an SVG document can be formatted with CSS. Most visual features and some aspects of element geometry are controlled by CSS properties.

This function is not supported by SIMATIC WinCC!

The properties from the style must be defined at the attributes of the respective places of use.

Does not work:

```
<rect id="ViereckUnten" style="x:125; y:125; fill:#EB695F; stroke:
#000000; stroke-width:0.75; width:50; height:50;" />
```

Works:

```
<rect id="ViereckUnten" x="125" y="125" fill="#EB695F"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
```

Interactivity

SVG objects can execute event-triggered actions (animations), for example changes in form, color etc. when the mouse is clicked.

This functionality is powered by SVG-internal animations.

This function is not supported by SIMATIC WinCC!

In order to achieve comparable effects in SIMATIC WinCC, use the binding options and script/events within the WinCC runtime.

Examples of this can be found below in chapter 3 Engineering: Dynamization of SVGHMI objects

Scripting

A 'script' element corresponds to the 'script' element in HTML and is thus the place for scripts (e.g. ECMAScript). All functions that are defined within a "script" element have a "global" validity to the complete present document.

This function is not supported by SIMATIC WinCC!

Animations

According to the SVG specification, animations are divided into two categories:

- Animation element
  An animation element is an element that can be used to animate the attribute or property value of a different element. The following elements are animation elements: animate, animateMotion, animateTransform, discard and set.
- Animation event attribute
  An animation event attribute is an even attribute that specifies a script which should be run for a specific animation-related event.
  The animation event attributes are `onbegin`, `onend` and `onrepeat`.

This function is not supported by SIMATIC WinCC!

Note In order to achieve comparable effects in SIMATIC WinCC, use the binding options and scripts/events within the WinCC runtime.

Examples of this can be found below in chapter 4, "Dynamization of SVGHMI objects".

Expandability

SVG is designed in such a way that it is compatible with other XML languages for describing and representing other types of content.

The 'foreignObject' element makes it possible to integrate elements in a non-SVG namespace and which can be rendered within a region of the SVG graphic with the help of other user agent processes.

This function is not supported by SIMATIC WinCC!

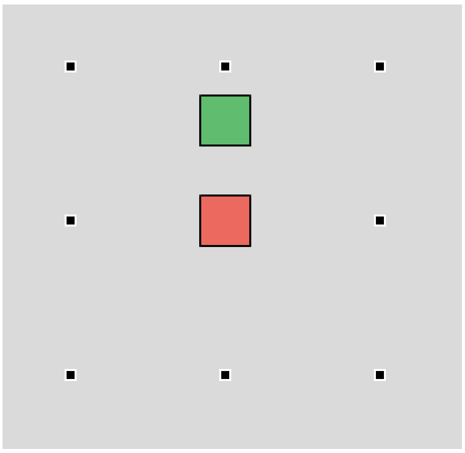
3 Engineering: Dynamization of SVGHMI objects

3.1 Starting point

Example

A graphic with two equally sized squares will be used as an example image.

Figure 3-1



Associated code

The XML code for this appears as follows:

The code, needed in every dynamic SVG, is already implemented.

Note Apart from the graphics elements, the code can be used and/or copied as a basis for a new dynamic SVG.

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">

<rect id="VierecKUnten" x="125" y="125" fill="#EB695F"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
<rect id="VierecKOben" x="125" y="25" fill="#60BC6E"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

Refer to the following chapters for examples of how graphics elements can be dynamized.

3.2 Linking colors

Now we wish to link the color of the squares to properties. Additional definitions will be made between the header and the graphic in order to achieve this.

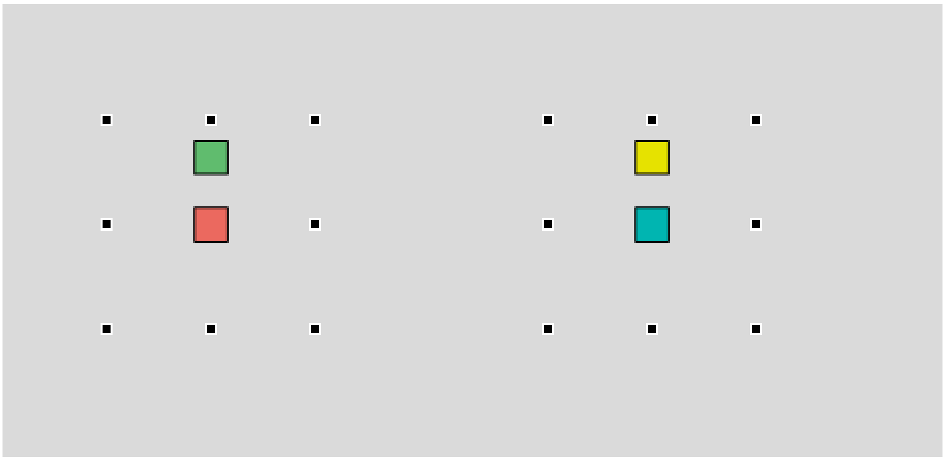
There are essentially two different types of these definitions:

- localProps: applies and is visible only within the document
- paramProps: these form an interface, meaning that they can be populated or interconnected externally (for example, from the Engineering System)

In order to interconnect properties from outside of the graphic, create the so-called `paramDefs` in the graphic.

As an example, the `BasicColor` will be linked to the "ViereckUnten" ["SquareLower"]:

Figure 3-2



In the graphic, `fill="#EB695F"` is now replaced by `hmi-bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"`.

This is also possible with lines. Here, `hmi-bind:stroke="{{Converter.RGBA(ParamProps.BasicColor)}}"` would be used.

The `ContrastColor` is now linked to the "ViereckOben" ["SquareUpper"]:

Here, `fill="#60BC6E"` is replaced by `hmi-bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"`. The original colors are set as default values behind the `paramDef`.

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
    <hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1" >
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
   </hmi:self>


<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />


</svg>
```

3.3 Linking a gradient

Color changes in dynamic SVGs are realized with gradients. If the gradient must be dynamized, this can be done with the same approach as before. In order to change the color, select a "stop-color" and dynamize it.

Without dynamization

```
<linearGradient id="Farbverlauf_ViereckUnten"
gradientUnits="userSpaceOnUse" x1="125" y1="125" x2="125" y2="175">
      <stop offset="0" stop-color="#EB695F" />
      <stop offset="1" stop-color="#FFFFFF" />
</linearGradient>
```

With dynamization

```
<linearGradient id="Farbverlauf_ViereckUnten"
gradientUnits="userSpaceOnUse" x1="125" y1="125" x2="125" y2="175">
      <stop offset="0" hmi-bind:stop-
color="{{Converter.RGBA(Converter.Darker(ParamProps.BasicColor,
0.000))}}" />
      <stop offset="1" hmi-bind:stop-
color="{{Converter.RGBA(Converter.Darker(ParamProps.BasicColor,
0.516))}}" />
```

3.4 Movement/translation of an object

Movements are called translations in an SVG environment.

In a translations, the following transform condition is inserted to the object that will be moved:

```
hmi-bind:transform="translate({{0   +
LocalProps.NormalizedValue*100}})   "
```

Starting point + LocalProps.NormalizedValue * length of the displacement

The starting point is the distance to the point at which the object is currently located. If the value is 0, the displacement starts at the original point of the object and not at the origin of the coordinate system.

3.4.1 Translation in x-direction

The "Position" must now be inserted into the ParamDefs. The "number" type is used here.

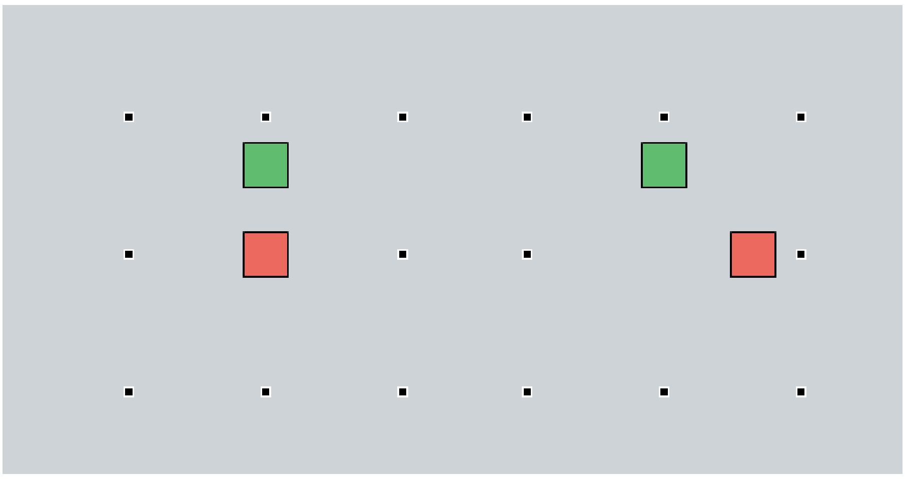
```
<hmi:paramDef name="Position"             type="number"
default="0"  />
```

Additionally, a local tag will be used here and in the other examples below. These are defined in the "defs":

```
<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>
```

The validity of the tag is defined here. In this case, it is between 0 and 1. The input tag is also divided by 100 beforehand. The "NormalizedValue" tag is used in this and the following examples to represent a displacement between 0 and 100 percent.

Figure 3-3



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1" >
     <hmi:paramDef name="BasicColor"              type="HmiColor"
   default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"           type="HmiColor"
   default="0xFF60BC6E" />
     <hmi:paramDef name="Position"                type="number"
   default="0"  />
</hmi:self>

<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>

<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50"
hmi-bind:transform=" translate({{0 +
LocalProps.NormalizedValue*100}})" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

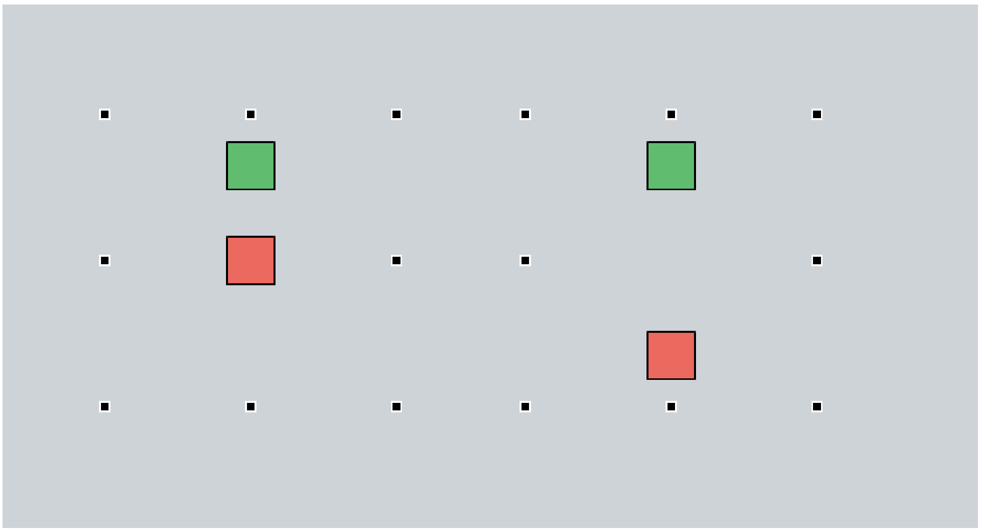
3.4.2 Translation in y-direction

If the translation will be done in the y-direction, the condition must appear as follows:

```
hmi-bind:transform="translate(0, {{0   +
LocalProps.NormalizedValue*100}})"
```

Displacement in x-direction, starting point + LocalProps.NormalizedValue * length of the displacement

Figure 3-4



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">

<hmi:self type="widget" displayName="BeispielViereck"
name="extended.BeispielViereck" version="1.0.1" >
     <hmi:paramDef name="BasicColor"            type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"         type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"              type="number"
    default="0"  />
</hmi:self>
```

```
<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>
```

```
<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50"
hmi-bind:transform=" translate(0,{{ 0 +
LocalProps.NormalizedValue*100}})" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.4.3 Translation in x- and y-direction

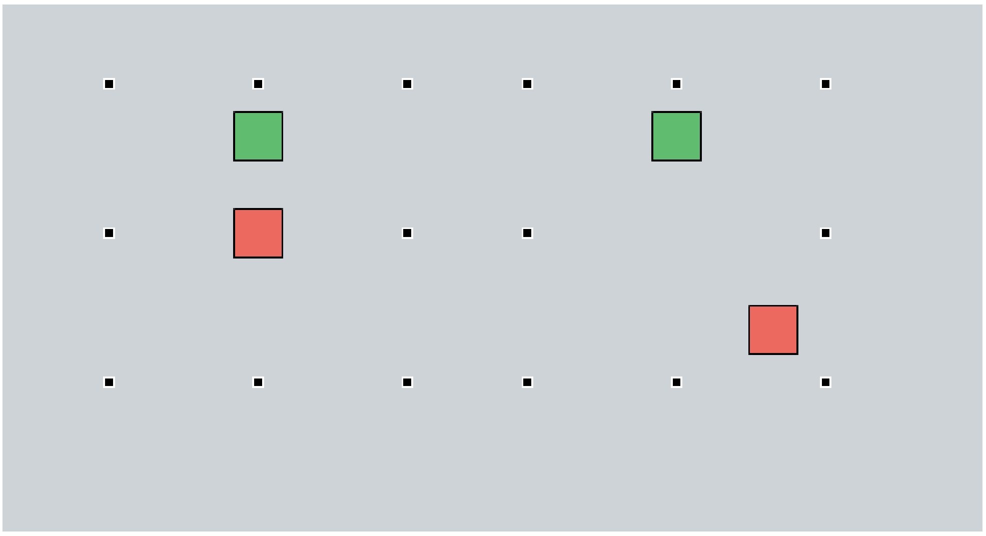
If the translation will be done in the x- and y-direction, the condition must appear as follows:

```
hmi-bind:transform=" translate({{0 +
LocalProps.NormalizedValue*100}}, {{0 +
LocalProps.NormalizedValue*100}})"
```

Starting point + LocalProps.NormalizedValue * length of the displacement, starting point + LocalProps.NormalizedValue * length of the displacement

Note Different local tags or interface parameters may also be used for x and y for dynamization.

Figure 3-5



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"

     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"               type="number"
    default="0"  />
</hmi:self>


<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>
```

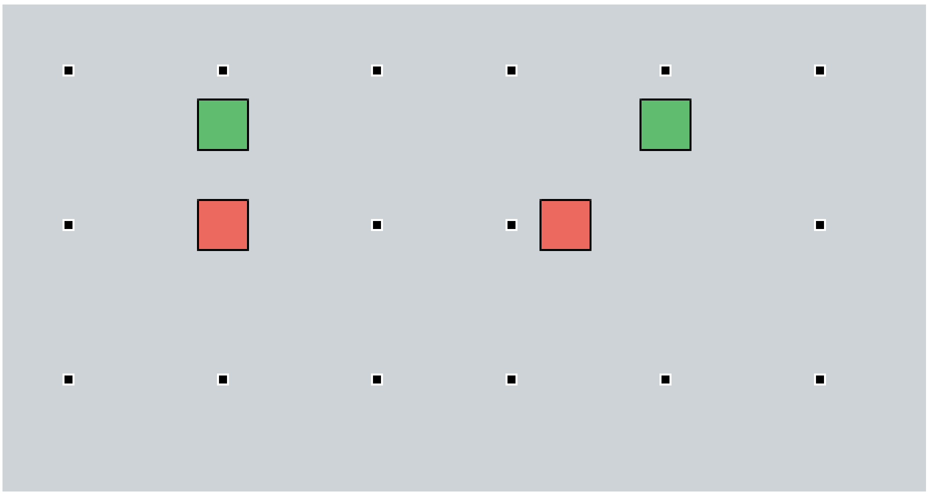
```
<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50"
hmi-bind:transform=" translate({{0 +
LocalProps.NormalizedValue*100}},{{ 0 +
LocalProps.NormalizedValue*100}})" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.4.4 Translation in negative x- and y-direction

If the translation must be done in the negative x- or y-direction, simply change the sign.

```
hmi-bind:transform=" translate({{0 -
LocalProps.NormalizedValue*100}})"
```

Figure 3-6

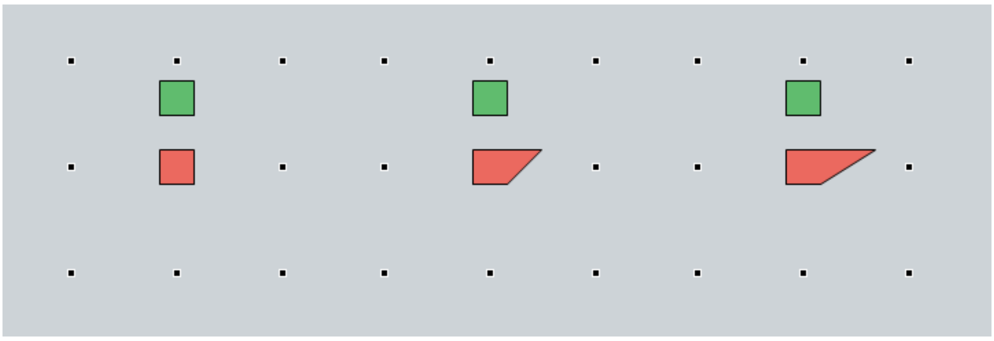


3.5 Movement/translation of a path

If the object to be translated consists of a path, then individual points may also be translated. Here, the tag is placed directly in the path instead of a coordinate.

```
hmi-bind:d="M125,125
L{{`(175+(100*LocalProps.NormalizedValue))`}},125 L175,175 L125,175
L125,125z"
```

Figure 3-7



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"               type="number"
    default="0"  />
</hmi:self>


<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>


<path id="ViereckUnten" hmi-bind:d="M125,125
L{{(175+(100*LocalProps.NormalizedValue))}},125 L175,175 L125,175
L125,125z" hmi-bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.6 Rotation

Rotational movement is known as rotation in an SVG environment.

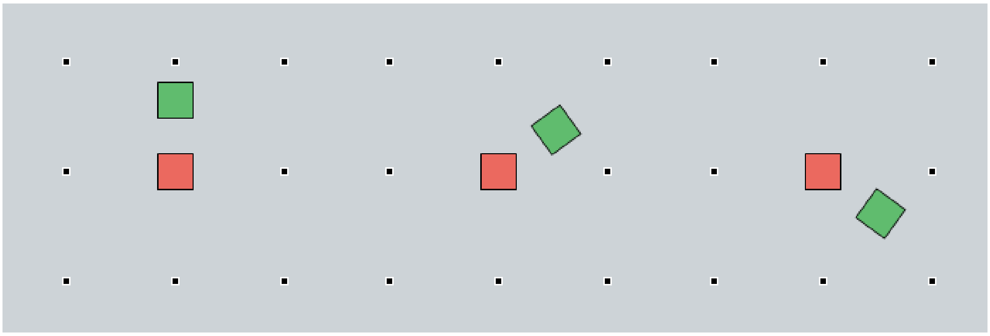
3.6.1 Clockwise rotation

If the object must be rotated, add the following condition:

```
hmi-bind:transform="rotate ({{0 + LocalProps.NormalizedValue*360}},
150, 150)"
```

Start point + LocalProps.NormalizedValue * angle of maximum rotation, x-coordinate of center of rotation, y-coordinate of center of rotation

Figure 3-8



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">

<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1" >
     <hmi:paramDef name="BasicColor"            type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"         type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"              type="number"
    default="0"  />
</hmi:self>

<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>
```

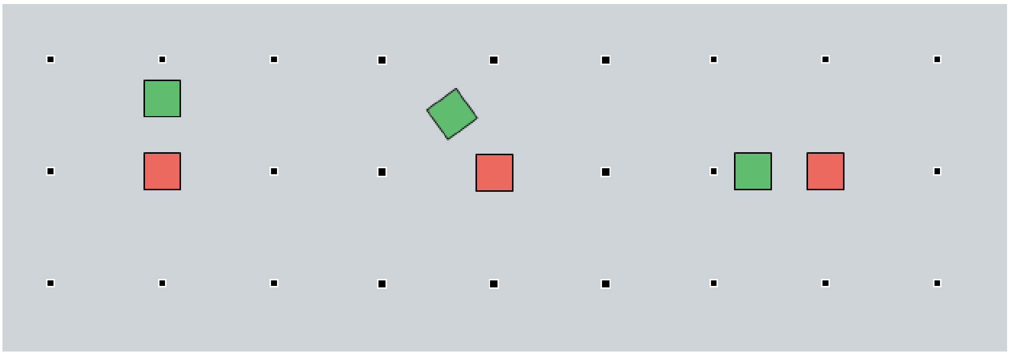
```
<rect id="VierecKUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
<rect id="VierecKOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:transform="rotate ({{0 + LocalProps.NormalizedValue*360}}, 150,
150)" />
</svg>
```

3.6.2 Counterclockwise rotation

If the rotation should occur counterclockwise, simply change the sign.

```
hmi-bind:transform="rotate ({{0 - LocalProps.NormalizedValue*360}},
150, 150)"
```

Figure 3-9



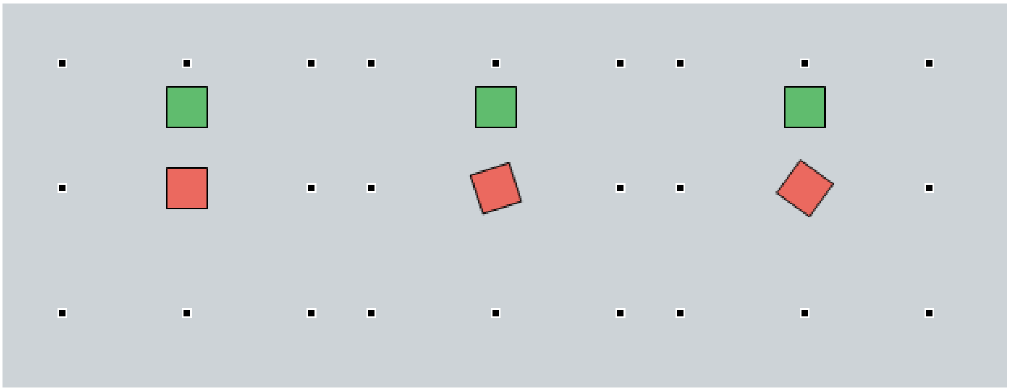
3.6.3 Rotation around internal axis

If the rotation should occur around an internal axis, add the following condition:

```
hmi-bind:transform="rotate ({{0 + LocalProps.NormalizedValue*360}},
150, 150)"
```

Start point + LocalProps.NormalizedValue * angle of maximum rotation, x-coordinate of center of rotation, y-coordinate of center of rotation

Figure 3-10



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"               type="number"
    default="0"  />
</hmi:self>


<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>


<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:transform="rotate ({{0 + LocalProps.NormalizedValue*360}}, 150,
150)"/>
```

```
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.7 Scaling (x-direction)

The following condition must be added in case of positive scaling in the x-direction and translation in x-direction:

```
hmi-bind:transform="translate({{125 –
(1+LocalProps.NormalizedValue)*125}}) scale({{1 +
LocalProps.NormalizedValue}},1)"
```

translate(starting point – (1+LocalProps.NormalizedValue) * starting point)

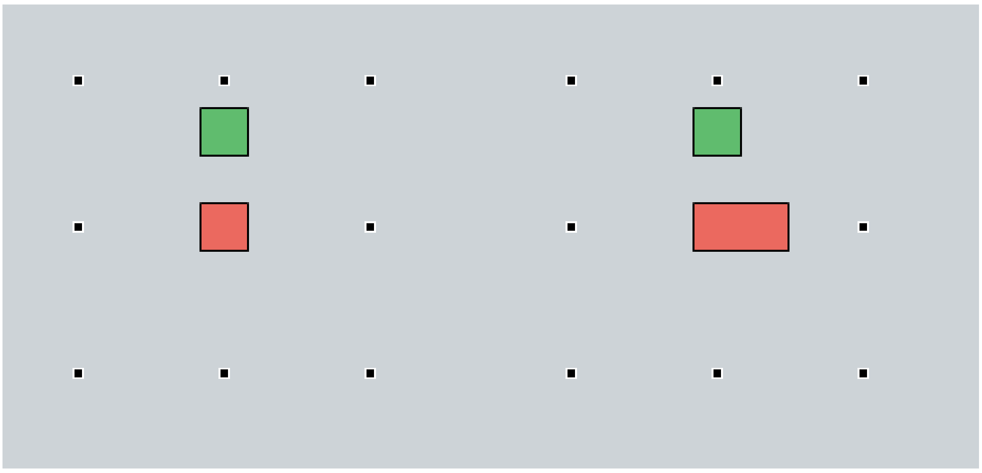
"translate" displaces the element. If the translation should occur as a function of the current position, the new position can be scaled with the following formula.

scale(1 + LocalProps.NormalizedValue, scaling in y-direction)

"scale" shrinks or enlarges the element. The element will be shrunk if the value is less than 1. If it is smaller than 0, the element will be mirrored and scaled. In this example, we wish to enlarge the element. Therefore, the value must be greater than 1.

The ones added on to the LocalProps.NormalizedValue cause the positive scaling.

Figure 3-11



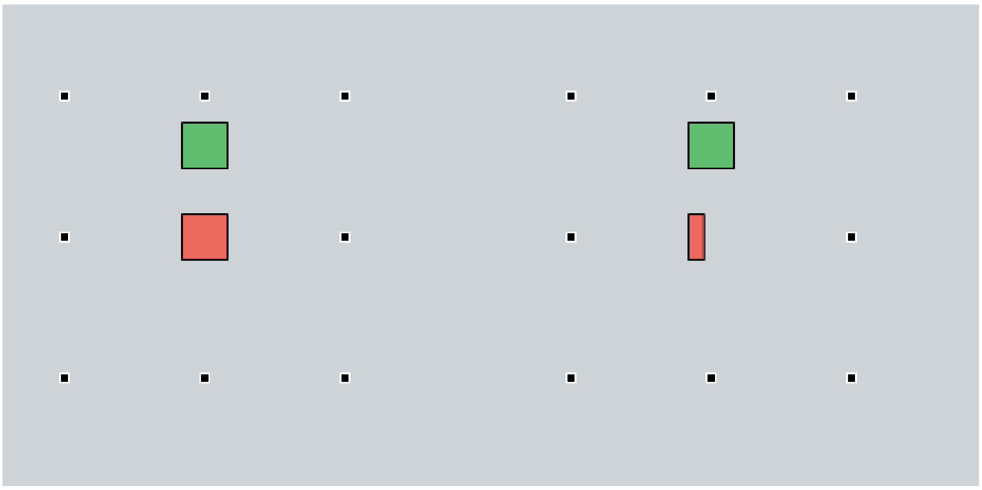
```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">


<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Position"               type="number"
    default="0"  />
</hmi:self>


<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>


<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:transform="translate({{125 -
(1+LocalProps.NormalizedValue)*125}}) scale({{1 +
LocalProps.NormalizedValue}},1)" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

The following condition must be added in case of negative scaling in the x-direction:

```
hmi-bind:transform="translate({{125 -
LocalProps.NormalizedValue*125}})
scale({{LocalProps.NormalizedValue}},1)"
```

Figure 3-12



3.8 Display/hide objects

3.8.1 Showing/hiding objects based on a Bool value

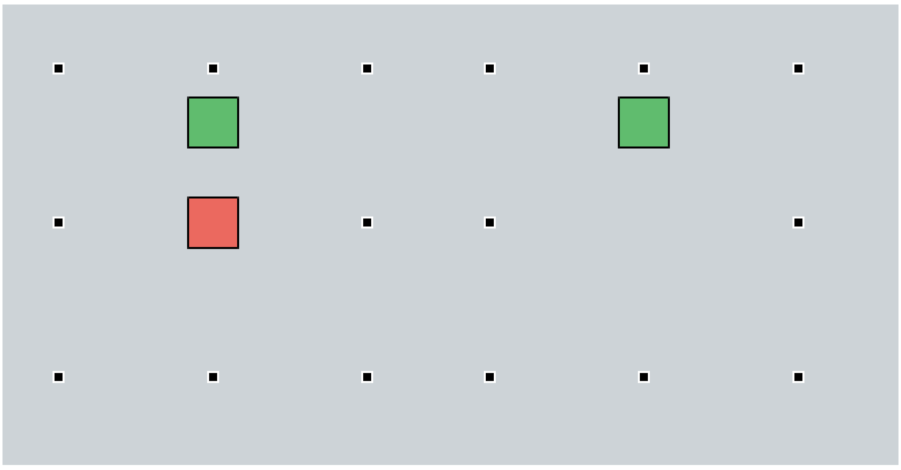
Objects can be shown and hidden with the condition:

```
hmi-bind:display="{{ ParamProps.Display ? 'inline' : 'none' }}"
```

And the following must be added in the paramDefs:

```
<hmi:paramDef name="Display"              type="boolean"
default="true"/>
```

Figure 3-13



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Display"                type="boolean"
    default="true"/>
</hmi:self>
<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:display="{{ ParamProps.Display ? 'inline' : 'none' }}"/>
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.8.2 Showing/hiding objects based on a number

Objects can also be displayed dependent on a number. This condition is used for the red square:

```
hmi-bind:display="{{ le( ParamProps.Display, 1) ? 'inline' :
'none'}}"
```

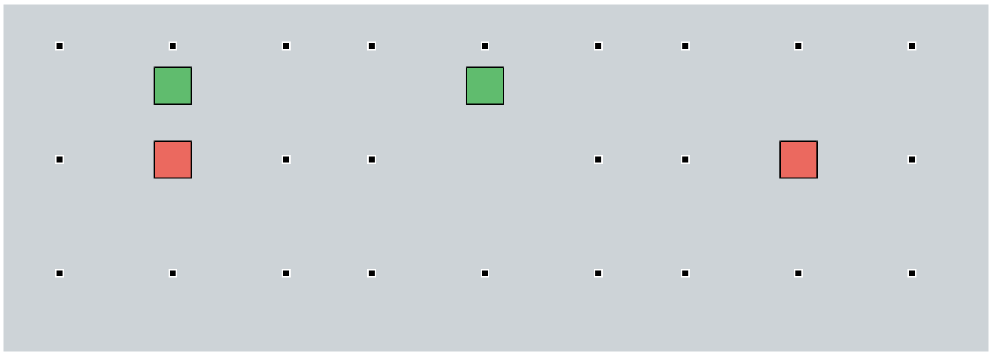
The red square will be displayed here if the value is less than or equal to 1.

This condition is used for the green square:

```
hmi-bind:display="{{ and(le( ParamProps.Display
,2),gt(ParamProps.Display,1)) ? 'inline' : 'none' }}"
```

The green triangle will be displayed if the value is greater than 1 and less than or equal to 2.

Figure 3-14



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielViereck"
name="extended.BeispielViereck" version="1.0.1">
     <hmi:paramDef name="BasicColor"           type="HmiColor"
    default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"        type="HmiColor"
    default="0xFF60BC6E" />
     <hmi:paramDef name="Display"         type="number"
default="1"/>
</hmi:self>
<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:display="{{ le(ParamProps.Display,1) ? 'inline' : 'none' }}" />
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
```

```
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:display="{{ and(le( ParamProps.Display
,2),gt(ParamProps.Display,1)) ? 'inline' : 'none' }}" />
</svg>
```

3.8.3 Showing/hiding objects based on a bit set in a tag

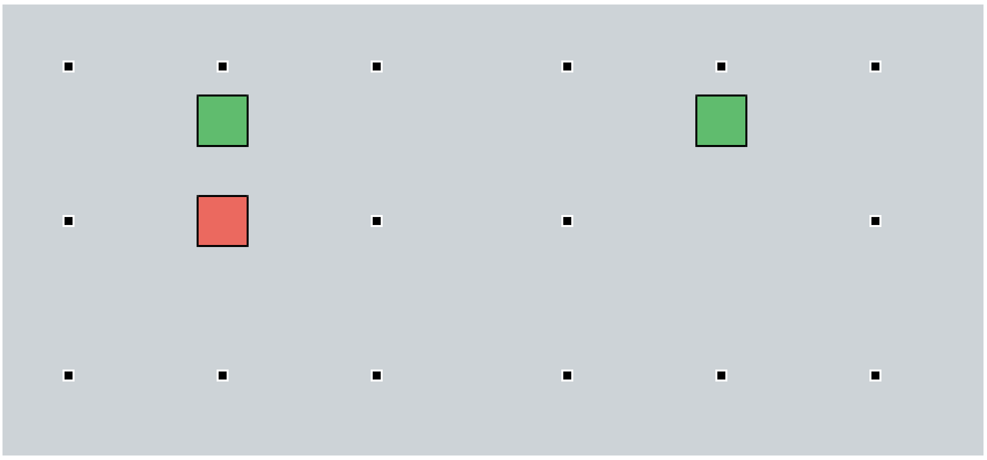
Objects can also be displayed as a function of a certain bit. The following condition is set for the red square:

```
hmi-bind:display="{{has(ParamProps.overwrite,
ParamProps.overwriteBit)? 'inline' : 'none'}}"
```

Overwrite is a Word or DWord and OverwriteBit is the bit number that determines what bit should be checked for.

The rectangle will be displayed if OverwriteBit is set for the DWord.

Figure 3-15



Overwrite = 85 → 0101 0101
OverwriteBit = 4 → 0001 0000
➔ Square is visible

Overwrite = 35 → 0010 0010
OverwriteBit = 4 → 0001 0000
→ Square is not visible

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">


<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
    <hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
        <hmi:paramDef name="Position"          type="number"
default="50"/>
```

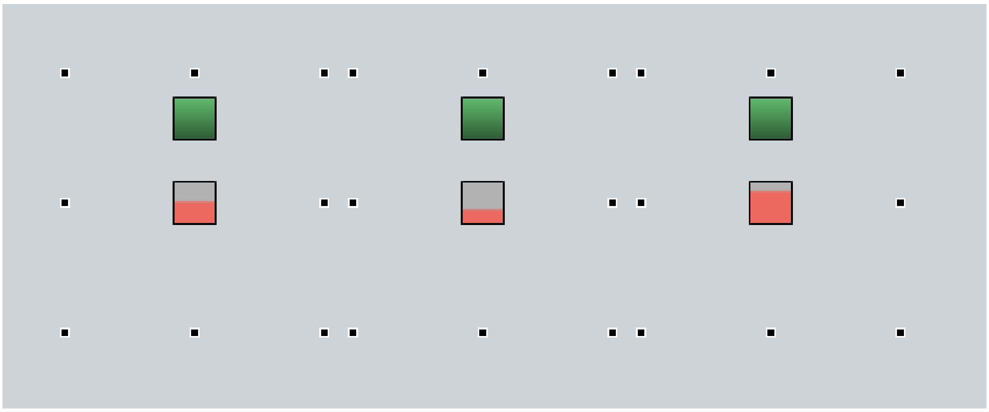
```
        <hmi:paramDef name="BasicColor"         type="HmiColor"
default="0xFFEB695F"   />
        <hmi:paramDef name="ContrastColor"      type="HmiColor"
default="0xFF60BC6E"   />
        <hmi:paramDef name="overwrite"          type="number"
default="16"/>
        <hmi:paramDef name="overwriteBit"       type="number"
default="4"/>


    </hmi:self>


     <defs>
       <hmi:localDef name="NormalizedValue"     type="number"   hmi-
bind:value="{{Converter.Bounds( ParamProps.Position/100, 0.0, 1.0)
}}" />
     </defs>


<rect id="ViereckUnten" x="125" y="125" hmi-
bind:fill="{{Converter.RGBA(ParamProps.BasicColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" hmi-
bind:display= =" {{has(ParamProps.overwrite,
ParamProps.overwriteBit)? 'inline' : 'none'}}}"/>
<rect id="ViereckOben" x="125" y="25" hmi-
bind:fill="{{Converter.RGBA(ParamProps.ContrastColor)}}"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
</svg>
```

3.9 Translating a gradient

The "linear-gradient" may be used to show a fill level, as we have already seen from dynamizing a gradient. These can be used to move the fill level.

This works by placing 2 "stop" points with the different colors at the same position.

Figure 3-16



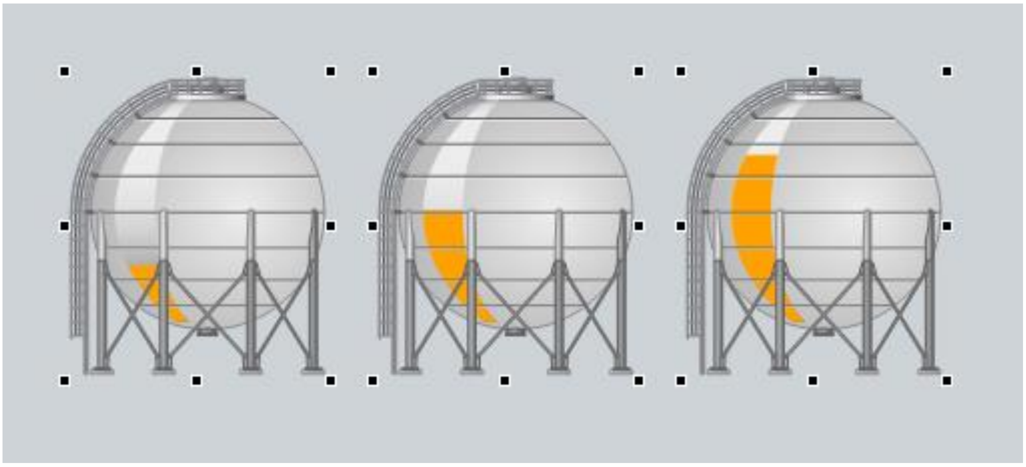
```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">
```

```
<hmi:self type="widget" displayName="BeispielVierecke"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="BasicColor"             type="HmiColor"
   default="0xFFEB695F" />
     <hmi:paramDef name="ContrastColor"          type="HmiColor"
   default="0xFF60BC6E" />
     <hmi:paramDef name="Position"               type="number"
   default="50"  />
</hmi:self>

<defs>
     <hmi:localDef name="NormalizedValue"   type="number"
       hmi-bind:value="{{Converter.Bounds( ParamProps.Position /
100, 0.0, 1.0) }}" />
</defs>


<linearGradient id="Farbverlauf_VierecktUnten"
gradientUnits="userSpaceOnUse" x1="125" y1="175" x2="125" y2="125">
```

```
      <stop offset="0"                                hmi-
bind:stop-color="{{Converter.RGBA(ParamProps.BasicColor)}}" />
      <stop hmi-bind:offset="{{LocalProps.NormalizedValue}}"  hmi-
bind:stop-color="{{Converter.RGBA(ParamProps.BasicColor)}}" />
      <stop hmi-bind:offset="{{LocalProps.NormalizedValue}}"  stop-
color="#B2B2B2" />
      <stop offset="1"                                stop-
color="#B2B2B2" />
</linearGradient>
<rect id="ViereckUnten" x="125" y="125"
fill="url(#Farbverlauf_ViereckUnten)" stroke="#000000" stroke-
width="0.75" width="50" height="50" />
<linearGradient id="Farbverlauf_ViereckOben"
gradientUnits="userSpaceOnUse" x1="125" y1="25" x2="125" y2="75">
      <stop offset="0" hmi-bind:stop-
color="{{Converter.RGBA(Converter.Darker(ParamProps.ContrastColor,
0.000))}}" />
      <stop offset="1" hmi-bind:stop-
color="{{Converter.RGBA(Converter.Darker(ParamProps.ContrastColor,
0.516))}}" /> </linearGradient>
<rect id="ViereckOben" x="125" y="25"
fill="url(#Farbverlauf_ViereckOben)" stroke="#000000" stroke-
width="0.75" width="50" height="50" />
</svg>
```

This allows even objects with a non-quadrilateral form to be filled, for example.

Figure 3-17

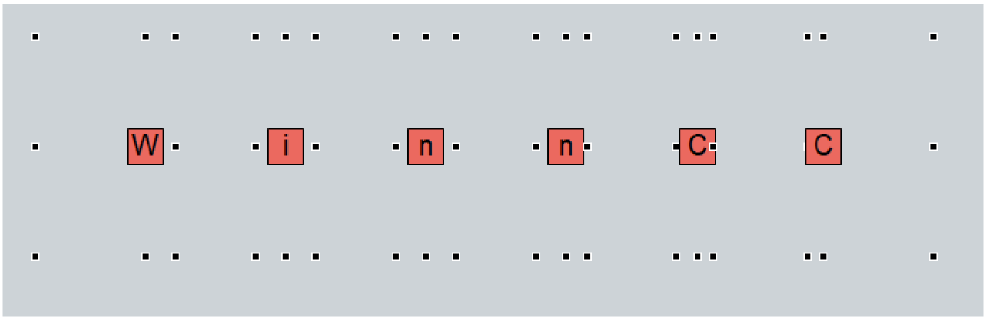


3.10 Dynamic text

Note the following syntax for dynamizing text. ParamProps.Text contains the dynamic text.

```
    <text x ="150" y="165" fill="#000000" font-family="Arial" font-
size="40" letter-spacing="3" text-anchor="middle">
        <hmi:text hmi-bind:value="{{ParamProps.Text}}" />
    </text>
```

Figure 3-18



```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">

    <hmi:self type="widget" displayName="Textbeispiel"
name="extended.BeispielVierecke" version="1.0.1">
     <hmi:paramDef name="Text"          type="string"
default="T" />
    </hmi:self>
```

```
    <rect id="ViereckUnten" x="125" y="125" fill="#EB695F"
stroke="#000000" stroke-width="0.75" width="50" height="50" />
    <text x ="150" y="165" fill="#000000" font-family="Arial" font-
size="40" letter-spacing="3" text-anchor="middle">
        <hmi:text hmi-bind:value="{{ParamProps.Text}}" />
    </text>
</svg>
```

3.11 Converter commands

Table 3-1

Converter	Description
IsString(any)	Returns true if the argument is handled as a string.
IsNumber(any)	Returns true if the argument is handled as a number (including NaN).
IsBoolean(any)	Returns true if the argument is handled as a Boolean.
CountItems(Array)	Returns the count of items within an array or collection.
RGB(HmiColor)	Extracts the RGB value from an ARGB-defined HmiColor (ignoring Alpha).
RGBA(HmiColor)	Returns the RGBA value for SVG from an ARGB-defined HmiColor.
Alpha(HmiColor)	Extracts the alpha value from an ARGB-defined HmiColor.
Illuminate(HmiColor, Number [=deviation], HmiColor [=low, def. #FFFFFF], HmiColor [=high, def. #000000])	Allows creating lower and higher illuminations of an input color, according to the given deviation. Deviation is defined as percentage with a range from -1.0 to 1.0. low and high define the direction, whereas low is usually the background color and high the foreground color. The direction is defined through the light-factor of both color's HSL representation. If those two parameters not given or have the same L-factor, the background is assumed to be white and the foreground to be black (which is often called a "light theme"). Note: The alpha value of low and high is ignored. The output color will retain the same alpha as the input color.
Darker(HmiColor, Number [=deviation])	Shortcut for Illuminate(HmiColor, Number, #FFFFFF, #000000) Deviation must always be a positive number from 0.0 to 1.0
Lighter(HmiColor, Number [=deviation])	Shortcut for Illuminate(HmiColor, (-1)*Number, #FFFFFF, #000000) Deviation must always be a positive number from 0.0 to 1.0
Bounds(Number, Number [=min],Number [=max])	Bounds a given number into an upper (max) and lower (min) limit.
Min(Number, Number)	Returns the minimum of the two given numbers
Max(Number, Number)	Returns the maximum of the two given numbers
FormatPattern(Value, String)	Formats a raw value (this is usually a numerical value such as the process value) according to a given string that holds the format pattern to apply on. The return value is always a string.
TextDecoration(HmiFontPart)	Returns the text decoration for the font referenced by the given HmiFontPart.
TextHeight(String, HmiFontPart	Returns the text height for the given text according to the given HmiFontPart.
TextWidth(String, HmiFontPart)	Returns the text width for the given text according to the given HmiFontPart.
TextEllipsis(String, Number [=width/length], HmiFontPart)	Returns the truncated text with the character "…" on the right side if the given text is longer than the provided width for the given font. Note: In case of a missing font, the passed number is interpreted as maximum text length.

3.12 Expressions

The following functions can be used in binding:

Table 3-2

Type	Symbols/methods	Example
Operators	+ - * / %	hmi-bind:x="{{(HmiProps.Width + LocalProps.Margin) * 0.5}}"
Comparators	== !=	hmi-bind:display="{{HmiProps.ProcessValue != 0.0 ? 'inline' : 'none'}}" Hint: Should be used for String compares
Conditional expression	bool ? x : y	hmi-bind:display="{{HmiProps.IsActive?'inline' :'none'}}"

Logical function calls	and(bool, bool)
	or(bool, bool) not(bool)
		hmi-bind:display="{{and(HmiProps.Enabled, not(HmiProps.Toggled)) ? 'inline' : 'none'}}"

Comparator function calls	gt(num, num) ge(num, num)
	lt(num, num)
	le(num, num) eq(num, num) ne(num, num)
	has(num, num)
		hmi-bind:display="{{ge(HmiProps.ProcessValue, 100) ? 'inline' : 'none'}}"
		Note:
		The "has" function is a bitwise AND operator (checks whether the second parameter is contained in the first).

Mathematical function calls	abs(num) round(num)
	floor(num)
	ceil(num)
	sin(num)
	asin(num)
	cos(num)
	acos(num)
	tan(num)
	atan(num)
	atan2(num,num)
	rad2deg(num)
	deg2rad(num)
	log(num)
	log10(num)
	sqrt(num)
	pow(num,num)
		hmi-bind:y="{{sin(HmiProps.Width)}}"

4 Listing of example properties

```
<!DOCTYPE svg PUBLIC "-//SIEMENS//DTD SVG 1.0 TIA-HMI//EN"
"http://tia.siemens.com/graphics/svg/1.8/dtd/svg18-hmi.dtd">



<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:hmi-bind--xlink="http://svg.siemens.com/hmi/bind/xlink/"
     xmlns:hmi="http://svg.siemens.com/hmi/"
     xmlns:hmi-bind="http://svg.siemens.com/hmi/bind/"
     xmlns:hmi-element="http://svg.siemens.com/hmi/element/"
     xmlns:hmi-feature="http://svg.siemens.com/hmi/feature/"
     xmlns:hmi-event="http://svg.siemens.com/hmi/event/"
     viewBox="0 0 300 300"
     preserveAspectRatio="none">

<hmi:self type="widget" displayName="18WheelerTruck"
name="extended.18WheelerTruck" version="1.0.1">
   <hmi:paramDef name="BasicColor"                type="HmiColor"
    default="0xFFEB695F" />
          <hmi:paramDef name="ContrastColor"
   type="HmiColor"      default="0xFF60BC6E" />
   <hmi:paramDef name="FillLevel"                 type="number"
default="1" />
   <hmi:paramDef name="PipeColor"                 type="HmiColor"
    default="0xFFEB695F" />
   <hmi:paramDef name="FluidColorLow"      type="HmiColor"
default="0xFFEB695F"  />
   <hmi:paramDef name="FluidColorHigh"     type="HmiColor"
default="0xFFEB695F" />
   <hmi:paramDef name="FireSize"                  type="number"
    default="1" />
   <hmi:paramDef name="Cutaway"                   type="boolean"
    default="true"/>
   <hmi:paramDef name="Position"                  type="number"
    default="1" />
   <hmi:paramDef name="Text"                      type="string" />

    </hmi:self>

     <defs>
       <hmi:localDef name="Color" type="HmiColor" hmi-
bind:value="{{ParamProps.ContrastColor}}" />
     </defs>
```

5 Appendix

5.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.

Please send queries to Technical Support via Web form:

support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

siemens.com/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

support.industry.siemens.com/cs/sc

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for iOS and Android:

support.industry.siemens.com/cs/ww/en/sc/2067

5.2 Links and literature

Table 5-1

No.	Subject
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the article page of the application example (currently unpublished) https://support.industry.siemens.com/cs/ww/en/view/109782045

5.3 Change documentation

Table 5-2

Version	Date	Change
V1.0	03/2021	First version