Honeywell

Uniformance® Insight Graphics Scripting User Guide

R110.1
July 2017

Release 110.1

Contact Information

Seeking technical assistance

For technical queries, contact Honeywell Customer Support.

For contact information, visit:

https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx

Reporting a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, follow the instructions at: https://honeywell.com/pages/vulnerabilityreporting.aspx.

Submit the requested information to Honeywell using one of the following methods:

- Send an email to security@honeywell.com.
- Contact your local Honeywell Technical Assistance Center (TAC).

Providing documentation feedback

To provide feedback about this document, or to report errors and omissions, write to us at:

hpsdocs@honeywell.com

Contents

1. Introduction

1.1 About this Document

This document provides information about authoring scripts in HMIWeb Display Builder for customizing the graphics to be used in Uniformance Insight. This document also provides some example scripts for reference.

1.2 About HMIWeb Display Builder

HMIWeb Display Builder is a specialized drawing application that enables you to create your own (custom) graphics (also called as displays). The graphics created using this application can be viewed in Uniformance Insight visualization browser.

1.3 Scripting for Graphics

HMIWeb Display Builder enables you to write scripts to customize displays that you want to view in other applications such as Uniformance Insight.

HMIWeb Display Builder exposes a script editor specifically for VBScript and JScript. JScript is Microsoft's dialect of the ECMAScript standard and is very similar to JavaScript.

2. Display Scripting

2.1 Overview

You use the Script Editor in HMIWeb Display Builder to write scripts for objects and the display itself.

The Script Editor is modeless, which means that your script is saved to memory as soon as you select another object or event. However, your changes are only saved to disk when you save the display.

2.2 Scripting for Uniformance Insight Graphics

Uniformance Insight displays are web pages designed using web technologies like Hypertext Markup Language 5 (HTML), Cascading Style Sheet (CSS), JavaScript, and so on. Display scripts use exactly the same technology as scripts used in other common standard web pages.

When authoring graphics, that are to be used in Uniformance Insight, the script must be written in JavaScript language. Hence, you must ensure that the default scripting language set for HMIWeb Display Builder is JScript (see Setting the default scripting language).

REFERENCE:

The following web sites provide useful documentation and tutorials on JavaScript:

- Mozilla's JavaScript - http://developer.mozilla.org/en-US/docs/Learn/JavaScript
- Microsoft's JavaScript Language Reference - http://msdn.microsoft.com/en-us/library/d1et7k7c(v=vs.94).aspx.

2.3 Object Model

Object Model is made up of the standard Document Object Model as well as Uniformance Insight specific extensions described in the Uniformance Insight Object Model Reference chapter.

2.4 Cascading Style Sheet (CSS)

Uniformance Insight graphics and various features (like Dynamic Visualization and Error Indication) are based on HTML elements and are styled using CSS classes.

Manipulating the style of an element directly using its inline style parameter will override standard Uniformance Insight visual features and cause them to not take effect. This is because CSS rules give higher priority to inline styles than CSS classes.

When writing script that needs to change the visual appearance of elements, such as borders, background, or foreground colors, it is recommended to remove or apply CSS

classes rather than using inline styles. This is especially the case for elements that retrieve data or have dynamic visualization.

ATTENTION:

Scripts that manipulate inline colors or borders of data bound elements may override standard Uniformance Insight functionality.

REFERENCE:

The following web sites provide useful documentation and tutorials on CSS specificity:

- CSS Tricks Specifics on CSS Specificity - http://css-tricks.com/specifics-on-css-specificity/
- Mozilla's Specificity - http://developer.mozilla.org/en-US/docs/Web/CSS/Specificity
- Microsoft's Understanding CSS Selectors - http://msdn.microsoft.com/en-us/library/hh781510(v=vs.85).aspx

2.5 Setting the default scripting language

You must set the default scripting language to JScript before you start writing scripts for Uniformance Insight displays.

To set the default scripting language to JScript:

1. Choose **Tools** > **Options**.
2. On the **General** Tab, set the **Default scripting language** to **JScript**.

3. Uniformance Insight Object Model Reference

3.1 Overview

This section describes the objects that are common within the Uniformance Insight document model and their related properties and methods.

3.2 Objects

Following are the object types within Uniformance Insight model.

- Alphanumeric Object
- Indicator Object
- Shape Object
- TimeControl Object
- Vector Object
- Trend Object
  - Axis Object
  - HairlineCollection Object
    - Hairline Object
  - Legend Object
  - TraceCollection Object
    - Trace Object
  - TraceCanvas Object

ATTENTION:

Scripts that are copied and pasted from this document to the script editor may not work, as the script editor may not recognize it due to the font styles and formats (for example, quotation marks often differ between fonts).

Alphanumeric Object

Properties

The following properties are accessible from an alphanumeric object (for example, document.getElementById('alpha001')

Property	Type	Description
displayValue	String	Gets the value of an alphanumeric as a string after all formatting options, including localization, have been applied.

fillColor	String	Gets or sets the fillColor of the object. The format of this return is browser-dependent.
		Examples
		`alpha001.fillColor = "#f3f3f3";`
		`alpha001.fillColor = "green";`
		`var color = alpha001.fillColor; //green`

lineColor	String	Gets or sets the lineColor of the object. The format of this return is browser-dependent.
		Examples
		`alpha001.lineColor = "#f3f3f3";`
		`alpha001.lineColor = "black";`
		`var lineColor = alpha001.lineColor; //black`

textColor	String	Gets or sets the textColor of the object. The format of this return is browser-dependent.
		Examples
		`alpha001.textColor = "#f3f3f3";`
		`alpha001.textColor = "gray";`
		`var textColor = alpha001.textColor; //gray`

Property	Type	Description

value	Number	Gets or sets the value of an alphanumeric.
		Examples
		`alpha001.value = 50;`
		`if(alpha001.value > 10){`
		`//do something`
		`}`
		`var val = alpha001.value;`
		For a bound object setting the value will not change the underlying data value. The value will be transient on the display only and will be overwritten by the next data update.

Events

- Onupdate Event

Indicator Object

Properties

The following properties are accessible from an indicator object (e.g. document.getElementById('indicator001')

Property	Type	Description

fillColor	String	Gets or sets the fillColor of the object. The format of this return is browser-dependent.
		Examples
		`indicator001.fillColor = "#f3f3f3";`
		`indicator001.fillColor = "green";`
		`var color = indicator001.fillColor; //green`
		For an indicator this sets the background color. To set the level fill color, see levelFillColor.

lineColor	String	Gets or sets the lineColor of the object. The format of this return is browser-dependent.
		Examples
		`indicator001.lineColor = "#f3f3f3";`
		`indicator001.lineColor = "black";`
		`var lineColor = indicator001.lineColor; //black`

levelFillColor	String	Gets or sets the levelFillColor of an indicator.
		This is the color that is filled in the indicator corresponding to the indicator value. The format of this return is browser-dependent.
		Examples
		`indicator001.levelFillColor = "red";`
		`indicator001.levelFillColor = "#c3c377";`
		`var indicatorFillColor = indicator001.levelFillColor; //"#c3c377"`

Property	Type	Description

value	Number	Gets or sets the value of an indicator.
		Example
		`if(indicator001.value > 10){`
		`//do something`
		`}`
		For a bound object setting the value will not change the underlying data value. The value will be transient on the display only and will be overwritten by the next data update.

Events

- Onupdate Event

Shape Object

Methods

Method	Description
getCustomProperty (string type, string name)	Returns the value of the specified custom property as a string: • type - The type to find, as specified for the custom property in display builder. • name - The name of the custom property. *Example* `var property = shape001.getCustomProperty("General", "BilgePump");`
objects (string name)	Returns the shape's child object with the specified name or null if not found. *Example* `var element = shape001.objects("alpha001");`

TimeControl Object

Properties

Property	Type	Description

startTime	Date	Gets or sets the start time of the timecontrol time span.
		Examples
		`timecontrol001.startTime = new Date('1992-04-01');`
		`var start = timecontrol001.startTime;`

endTime	Date	Gets or sets the end time of the timecontrol time span.
		Examples
		`timecontrol001.endTime = new Date('1992-04-03');`
		`var end = timecontrol001.endTime;`

Methods

Method	Description
setTimeWindow (Date startTime, Date endTime)	Sets the timecontrol time span. *Example* `timecontrol001.setTimeWindow( new Date('1992-04-01'), new Date('1992-04-02'));` Setting a new time window will flow on to the workspace time control and take effect for all other tiles in a workspace.

Vector Object

Description

A vector object is a 'line-based' object such as a rectangle or curve.

Properties

Property	Type	Description

fillColor	String	Gets or sets the fillColor of the object. The format of this return is browser-dependent.
		Examples
		`oval001.fillColor = "#f3f3f3";`
		`oval001.fillColor = "green";`
		`var ovalColor = oval001.fillColor; //green`

lineColor	String	Gets or sets the lineColor of the object. The format of this return is browser-dependent.
		Examples
		`oval001.lineColor = "#f3f3f3";`
		`oval001.lineColor = "black";`
		`var ovalStrokeColor = oval001.lineColor; //black`

lineWidth	String	Gets or sets the lineWidth of the object.
		Examples
		`oval001.lineWidth = "2";`
		`var ovalStrokeWidth = oval001.lineWidth; //”2”`

Trend Object

Description

The Trend object represents a trend and contains a number of child objects that provide control over specific parts of the trend.

Properties

Property	Description
hairlines	Returns the HairlineCollection Object.
legend	Returns the Legend Object.
traces	Returns the Trace Object.
traceCanvas	Returns the TraceCanvas Object.
xAxis	Returns the Axis Object representing the x-axis.
yAxis	Returns the Axis Object representing the y-axis.
yAxisMode	Gets or sets the axis mode. Available options are: • 0 – multi-scale • 1 – single scale • 2 – stacked scale

Methods

Method	Description
getVisibleDataAsStrings()	Returns an array of the visible data in the trend in a csv friendly format.

Axis Object

Description

The Axis object is a child object of the Trend object. It contains properties for controlling and manipulating an axis of the trend

Properties

Property	Type	Description
backgroundColor	string	Sets or gets the background color of the axis.
gridLinesVisible	boolean	Sets or gets the visiblility of the grid lines for the axis, for example the x-axis controls the vertical grid lines.
lineColor	string	Sets or gets the line color of the axis.
locked	boolean	Sets or gets if zoom operations are able to change the y-axis scale. Applicable to the y-axis only.
linewidth	number	Sets or gets the width of the axis line.
visible	boolean	Sets or gets the visibility of the axis.

Methods

None

HairlineCollection Object

Description

The HairlineCollection Object is a child object of the Trend object. It contains properties and methods for accessing individual hairline objects

Properties

Property	Type	Description
hairlines	Array	Array of Hairline Object. The hairlines are sorted by date.

Methods

Method	Description
hairline(date)	Hairline Object

Hairline Object

Description

The Hairline Object contains properties and methods for accessing trace values where traces intersect with a hairline

Properties

Property	Type	Description
date	number	Returns the date of the hairline.
values	Array	Returns an Array of objects representing where the hairline intersects with traces in the trend. Each object contains: • traceId • value

Methods

Method	Description
getValue (number traceId)	Returns the value for the given traceId at the point where the trace intersects with the hairline.

Legend Object

Description

The Legend object is a child object of the Trend object. It contains properties to control and manipulate the legend associated with the trend.

Properties

Property	Type	Description
dockPosiiton	string	Gets or sets the docking position of the legend. Available options are: • “float” • “bottom” • “top” • “left” • “right”
minimized	boolean	Gets or sets whether the legend is minimized.
visible	boolean	Gets or sets the visibility of the legend.

TraceCollection Object

Description

The TraceCollection Object is a child object of the Trend object. It contains properties and methods for adding removing or accessing individual trace objects.

Properties

Property	Type	Description
count	number	Gets the number of traces in a Trend Object. *Example* `var numberOfTraces = trend001.traces.count;`

Methods

Method	Description
trace (number traceId)	Returns the trace object associated with the specified traceId or null if no trace exists with the traceId. *Example* `var trace = trend001.traces.trace(0);`
addTrace (string itemName, string dataSource, string aggregate, number sampleInterval, string requestUnits, number traceId)	Adds a trace to the trend with the following parameters passed as arguments: • itemName: The item/tag name to add to the trace (e.g. 11AFC01.PV). • dataSource: The data source of the tag. • aggregate: The aggregate to request from the data source (e.g. "Auto", "Raw", "BestFit"). Note that not all data sources support all aggregates. • sampleInterval: The requested sample interval in seconds. • requestUnits: The unit type to request from the data source. Leave as the empty string ("") to request the default units for the item. • traceId – [Optional] The trace id to assign to the new trace. If left blank, the next available traceId is automatically assigned. Returns the new Trace object. If a traceId was specified and the trace already existed, the parameters passed in will apply to the existing trace.

	Returns null if there was an error adding the trace. *Example* `trend001.traces.addTrace("11AFC01.PV", "DefaultPHD", "Snapshot", 5, "");`
removeTrace (number traceId)	Removes the trace with the provided traceId from the trend. Returns true if the trace was found and removed, returns false if not. *Syntax* `trend001.traces.removeTrace(0);`

Trace Object

Properties

Property	Type	Description
aggregate	string	Gets or sets the aggregate of the trace.
color	string	Gets or sets the color of the trace line.
continuousAutoScale	boolean	Gets or sets if the scale of the trace will be automatically updated to fit the visible data.
description	string	Gets the description of the trace.
id	number	Gets the ID of the trace.
itemName	string	Gets or sets the item/tag name of the trace.
lineWidth	number	Gets or sets the width of the trace line.
visible	boolean	Gets or sets the visibility of the trace.
sampleInterval	number	Gets or sets the resample interval of the trace. The requested resample interval may be automatically adjusted if the request would result in too much data. The resample interval is ignored for RAW aggregates.
selected	boolean	Gets or sets whether the trace is selected. A selected trace is drawn with a two pixel line and if in multi-scale the selected trace's scale appears on the y-axis.
step	boolean	Gets or sets the step mode for the trace. In step mode, the plot line is drawn horizontally to the next sample, where a vertical line then connects to the sample. Step mode makes it easier to see sample values using hairlines or the active cursor. If step mode is not used, values are interpolated.
tagSource	string	Gets or sets the data source for the trace.
units	string	Gets or sets the unit of measure for the trace.

Methods

Method	Description
getYAxisScale	Returns the y-axis scale of the trace as an Array of two numbers where the first number is the low scale and the second number is the high scale.
setYAxisScale ([number minScale, number maxScale])	Sets the y-axis scale of the trace. Setting the y-axis scale will drop a trace out of auto scale if it was previously enabled.

TraceCanvas Object

Description

The TraceCanvas object is a child object of the Trend object. It contains properties for controlling and manipulating the visual aspects of the trace canvas. The trace canvas is the visual area within the trend on which the traces are rendered.

Properties

Property	Type	Description
backgroundColor	string	Gets or sets the background color of the trace canvas.

3.3 Event

Onupdate Event

Applicable to

- Alphanumeric Object
- Indicator Object

Description

Fires when the value of the object is updated from the server.

Remark

The event does not bubble.

4. Scripting Examples

4.1 Overview

This section provides examples on how to script for wide range of realistic tasks.

ATTENTION:

Scripts that are copied and pasted from this document to the script editor may not work, as the script editor may not recognize it due to the font styles and formats (for example, quotation marks often differ between fonts).

4.2 Changing context text based on a numeric value

Scenario

To make displays easier to use by setting a text field based on an object's value.

Solution

Write a script to set a textbox's text property whenever the value you are interested in changes:

1. Add a **textbox** element called **textbox001**.
2. Add an **alphanumeric** element called **alpha001** and configure it to retrieve data.
3. Add a script to the **onupdate** event of the **alpha001** element:

```
function alpha001_onupdate
{
//////// START OF SCRIPT ////////
var equipStateOrdinal = alpha001.value;
var equipState;
switch (equipStateOrdinal)
{
Case 0: equipState = “Stopped”; break;
Case 1: equipState = “Measurement”; break;
Case 2: equipState = “Completed”; break;
Case 3: equipState = “Stabilization”; break;
default: equipState = “Error”; break;
}
textbox001.value = equipState;
//////// END OF SCRIPT ////////
}
```

4.3 Creating a shape sequence

Scenario

You have a pump that has 2 states (OFF and ON), a tag that has values 0 and 1 (for OFF and ON), and want to visually show a different picture of the pump depending on its state.

Solution

Create a shape with the 2 pump states:

1. In **HMIWeb Display Builder**, go to **File** > **New** > **Dynamic Shape**.
2. Add a **Custom Property** with:
   - **Name** of **Tagname**
   - **Type** of **Item**
3. Add vector elements to represent the pump's 1st state, group them together, and call the group **state001**.
4. Add vector elements to represent the pump's 2nd state, group them together, and call the group **state002**.
5. Add an alphanumeric called **shapevalue** to read the value to drive the shape sequence.
6. Configure the alphanumeric:
   - On the **Data** tab, configure **Item** to be **{%Item::Tagname%}**
   - On the **Times** tab, configure **Time Control** to be **Configured on display**
7. Select all (Ctrl+A) the elements in the shape (**state001**, **state002** and **shapevalue**), align them so that they are all on top of each other and then group them together.

Configure the group:

1. On the **General** tab, set the visibility of the entire group to be hidden.
2. Add a script to the shape for the **onupdate** event of the **shapevalue** element:

```
function shapevalue_onupdate
{
//////// START OF SCRIPT ////////
state001.style.visibility = "hidden";
state002.style.visibility = "hidden";
switch (shapevalue.value) {
case 0: state001.style.visibility = "visible"; break;
case 1: state002.style.visibility = "visible"; break;
}
```

```
//////// END OF SCRIPT ////////
}
```

3. Click **File** > **Save** to save the shape
4. Insert the shape into a display:
    1. In **HMIWeb Display Builder**, go to **File** > **New** > **Display**.
    2. Go to **Edit** > **Insert Shape**.
5. Browse to the shape file created above.
6. Configure the shape:
    1. On the **Custom Properties** tab, configure **Time Control** to be the primary time control.
    2. On the **Custom Properties** tab, configure **Item** to be the tag that has the pump value.

4.4 Showing/hiding an object with item values

Scenario

Show or hide the element depending on the values of several tags.

Solution

Write a script that changes the visibility property whenever the values you are interested in change.

This example is for a group of elements called "**well101**" to be shown or hidden when the values of "**alpha001**" and "**alpha002**" change.

1. Add a script to the **onupdate** event of the **alpha001** element:

```
function alpha001_onupdate
{
//////// START OF SCRIPT ////////
updateWellVisibility();
//////// END OF SCRIPT ////////
}
```

2. Add a script to the **onupdate** event of the **alpha002** element:

```
function alpha002_onupdate
{
//////// START OF SCRIPT ////////
updateWellVisibility();
//////// END OF SCRIPT ////////
}
```

3. Add a **general** script to the display:

```
//////// START OF SCRIPT //////// function
updateWellVisibility()
{
var P101 = document.getElementById(“alpha001”);
var F101 = document.getElementById(“alpha002”);
var Well101 = document.getElementById(“well101”);
If (P101.value < 0.5) && (F101.value < 5)) {
Well101.style.visibility = “hidden”;
}
Else {
Well101.style.visibility = “visible”;
}
} //////// END OF SCRIPT ////////
```

4.5 Showing a different picture on mouse click

Scenario

Create a display that needs to show a different picture when an operator requests it.

Solution

Add two images to the display, called **ImageMap** and **ImageSatellite**. Use a button (textbox element with **onclick** event) to toggle the visibility of the images on the display.

This script is attached to a textbox element’s **onclick** event.

1. Add an **image** element called **ImageMap**.
2. Add an **image** element called **ImageSatellite**.
3. Add a **textbox** element called **textbox001**.
4. Add a script to the **onclick** event of the **textbox001** element:

```
function textbox001_onclick
{ //////// START OF SCRIPT ////////
if (ImageSatellite.style.visibility == “visible” ||
ImageSatellite.style.visibility == “”) {
ImageSatellite.style.visibility = “hidden”;
ImageMap.style.visibility = “visible”;
}
Else {
ImageSatellite.style.visibility = “visible”;
ImageMap.style.visibility = “hidden”;
} //////// END OF SCRIPT ////////
}
```

4.6 Stepping through time

Scenario

Step backwards and forwards through time in whole day increments snapped to a day boundaries. The Workspace Time Control allows stepping but only in 1/12 of the slider time span and not snapped to day boundaries.

Solution

Add two clickable images or textboxes to the display, called **Previous** and **Next**. Add script to these elements' **onclick** event to move the time.

1. Add a **textbox** element called **PreviousDay**.
2. Add a **textbox** element called **NextDay**.
3. Add a **general** script to the display:

```
//////// START OF SCRIPT //////// var timeControl;
function getPrimaryTimeControl()
{
var primaryTimeControl = null;
var timeControls = document.getElementsByTagName('hn-
timecontrol');
for (var i = 0; i < timeControls.length; i++) {
var timeControl = timeControls[i];
if (!primaryTimeControl) {
// Make sure we get one even if the primary attribute is not
set
primaryTimeControl = timeControl;
}
else if (timeControl.getAttribute('data-isprimary') === "true")
{
primaryTimeControl = timeControl;
}
}
return primaryTimeControl;
}
//////// END OF SCRIPT ////////
```

4. Add a script to the **onclick** event of the **PreviousDay** element:

```
function PreviousDay_onclick
{
//////// START OF SCRIPT //////// if (timeControl === null ||
typeof timeControl === "undefined") {
timeControl = getPrimaryTimeControl();
}
```

```
// Get the current end time
var end = new Date(timeControl.endTime);
// Calculate midnight on the previous day
end.setDate(end.getDate() - 1);
end.setHours(0, 0, 0, 0);
// Calculate the start date 24hrs earlier
var start = new Date(end);
start.setDate(end.getDate() - 1);
start.setHours(0, 0, 0, 0);
// Set times on the time control
timeControl.setTimeWindow(start, end);
//////// END OF SCRIPT ////////
}
```

5. Add a script to the **onclick** event of the **NextDay** element:

```
function NextDay_onclick
{
//////// START OF SCRIPT //////// if (timeControl === null ||
typeof timeControl === "undefined") {
timeControl = getPrimaryTimeControl();
}
// Get the current end time
var end = new Date(timeControl.endTime);
// Calculate midnight on the next day
end.setDate(end.getDate() + 1);
end.setHours(0, 0, 0, 0);
// Jump back a day if it's a future date
if (Date.now() < end.getTime()) {
end.setDate(end.getDate() - 1);
}
// Calculate the start date 24hrs earlier
var start = new Date(end);
start.setDate(end.getDate() - 1);
start.setHours(0, 0, 0, 0);
// Set times on the time control
timeControl.setTimeWindow(start, end);
//////// END OF SCRIPT ////////
}
```

5. Scripting Performance

5.1 Overview

This section covers some tips for authoring scripts to improve the performance of displays.

5.2 Using general section functions in a display

Sometimes you may need to repeat the same functionality for many elements in a display. It can be easy to use copy and paste to replicate the same script across multiple elements in a display.

However, this is not an ideal solution. It makes the display harder to maintain and larger in size than necessary, making the display slower to call up.

You can improve the maintainability and performance by using general section functions in the display to provide the required common functionality, and calling those functions with scripts from the event scripts for the relevant objects.

The performance improvements are particularly significant if there are multiple instances of the script in a display.

Example

Before

You have 10 alphanumerics that execute standard flow calculations. You copy and paste the script for each alphanumeric so that there are 10 copies of the same script in each alphanumeric's onupdate event.

```
function alpha001_onupdate
{
Do standard flow calculations based on alpha001.value (eg. 10
lines of script)
Update UI of alpha001 based on the result
}
…
function alpha010_onupdate
{
Do standard flow calculations based on alpha010.value (eg. 10
lines of script)
Update UI of alpha010 based on the result
}
```

After

To improve the performance of the display, you write the following general section function called "standardFlowCalculations".

```
function standardFlowCalculations(inputvalue) {
```

```
Do standard flow calculations based on inputValue (eg. 10 lines of
script)
Return result
}
```

You then update the script in each alphanumeric's **onupdate** event to call the general section function.

```
function alpha001_onupdate
{
var result = standardFlowCalculations(alpha001.value); // only 1
line
Update UI of alpha001 based on the result
}
…
function alpha010_onupdate
{
var result = standardFlowCalculations(alpha010.value); // only 1
line
Update UI of alpha010 based on the result
}
```

5.3 Using script holder shapes

Shapes provide the ability to create a "custom object" that can be easily re-used within a display and across displays. When a shape is inserted into a display, the visual elements as well as the scripts are added to the parent display.

If a particular shape is inserted multiple times to a display, this will result in the same scripts being added to the display multiple times. This makes the display larger in size than necessary making the display slower to call up.

You can improve the performance by using general section functions in a separate shape that is not repeated in the display. General section functions in the display and from any inserted shape and can be called from any of the elements or shape elements in the display.

The performance improvements are particularly significant if there are multiple instances of the shape in a display.

Example

Before

You have a shape "flowmeter.sha" that represents a flowmeter. You insert this shape 10 times into the display. The "flowmeter.sha" shape has script in both the alphanumeric's **onupdate** event and general section.

```
function standardFlowCalculations(inputvalue) {
```

```
Do standard flow calculations based on inputValue (eg. 10 lines of
script)
Return result
}
function alpha001_onupdate
{
var result = standardFlowCalculations(alpha001.value); // only 1
line
Update UI of alpha001 based on the result
}
```

After

To improve the performance of the display, you create a new shape called "scriptholdershape.sha" and write the following general section function called "updateFlowMeterElement" in the shape. You insert a single instance of the "scriptholdershape.sha" into the display.

```
function updateFlowMeterElement(inputElement) {
Do standard flow calculations based on inputElement.value (eg. 10
lines of script)
Update UI of inputElement based on the result
}
```

You then update the **onupdate** event script in "flowmeter.sha" to reference the general script function in the other shape.

```
function alpha001_onupdate
{
updateFlowMeterElement(this);
}
…
function alpha010_onupdate
{
updateFlowMeterElement(this);
}
```