Contact Information

Seeking technical assistance
For technical queries, contact Honeywell Customer Support.
For contact information, visit:

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 .......................................................................................... 6
   1.1 About this Document .................................................................................. 6
   1.2 About HMIWeb Display Builder ............................................................... 6
   1.3 Scripting for Graphics .............................................................................. 6

2. DISPLAY SCRIPTING .................................................................................. 7
   2.1 Overview .................................................................................................... 7
   2.2 Scripting for Uniformance Insight Graphics ............................................. 7
   2.3 Object Model ............................................................................................ 7
   2.4 Cascading Style Sheet (CSS) ...................................................................... 7
   2.5 Setting the default scripting language ...................................................... 8

3. UNIFORMANCE INSIGHT OBJECT MODEL REFERENCE .................. 9
   3.1 Overview .................................................................................................... 9
   3.2 Objects ........................................................................................................ 9
      Alphanumeric Object ...................................................................................... 10
      Indicator Object .......................................................................................... 12
      Shape Object ............................................................................................... 14
      TimeControl Object .................................................................................... 15
      Vector Object .............................................................................................. 16
      Trend Object ............................................................................................... 17
      Axis Object .................................................................................................. 18
      HairlineCollection Object ........................................................................... 19
      Hairline Object ............................................................................................ 20
      Legend Object .............................................................................................. 21
      TraceCollection Object ............................................................................... 22
      Trace Object ................................................................................................ 24
      TraceCanvas Object ..................................................................................... 26
   3.3 Event .......................................................................................................... 27
      Onupdate Event ............................................................................................ 27

4. SCRIPTING EXAMPLES ............................................................................. 28
4.1 Overview ................................................................. 28
4.2 Changing context text based on a numeric value ................. 28
   Scenario ................................................................. 28
   Solution .................................................................. 28
4.3 Creating a shape sequence .............................................. 29
   Scenario ................................................................. 29
   Solution .................................................................. 29
4.4 Showing/hiding an object with item values ......................... 30
   Scenario ................................................................. 30
   Solution .................................................................. 30
4.5 Showing a different picture on mouse click ....................... 31
   Scenario ................................................................. 31
   Solution .................................................................. 31
4.6 Stepping through time .................................................. 32
   Scenario ................................................................. 32
   Solution .................................................................. 32
5. SCRIPTING PERFORMANCE .......................................... 34
5.1 Overview .................................................................. 34
5.2 Using general section functions in a display ..................... 34
   Example .................................................................. 34
5.3 Using script holder shapes ............................................. 35
   Example .................................................................. 35
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:

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:


### 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')`):

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayValue</td>
<td>String</td>
<td>Gets the value of an alphanumeric as a string after all formatting options, including localization, have been applied.</td>
</tr>
<tr>
<td>fillColor</td>
<td>String</td>
<td>Gets or sets the fillColor of the object. The format of this return is browser-dependent.</td>
</tr>
<tr>
<td>lineColor</td>
<td>String</td>
<td>Gets or sets the lineColor of the object. The format of this return is browser-dependent.</td>
</tr>
<tr>
<td>textColor</td>
<td>String</td>
<td>Gets or sets the textColor of the object. The format of this return is browser-dependent.</td>
</tr>
</tbody>
</table>

**Examples**

```javascript
alpha001.fillColor = "#f3f3f3";
alpha001.fillColor = "green";
var color = alpha001.fillColor; // green
```

```javascript
alpha001.lineColor = "#f3f3f3";
alpha001.lineColor = "black";
var lineColor = alpha001.lineColor; // black
```

```javascript
alpha001.textColor = "#f3f3f3";
alpha001.textColor = "gray";
var textColor = alpha001.textColor; // gray
```
## Property Type Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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
Uniformance Insight Object Model Reference
Objects

Indicator Object

Properties

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| value    | Number | Gets or sets the value of an indicator.  
**Example**  
```javascript
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*

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCustomProperty</td>
<td>Returns the value of the specified custom property as a string:</td>
</tr>
<tr>
<td>(string type, string name)</td>
<td>• type - The type to find, as specified for the custom property in display builder.</td>
</tr>
<tr>
<td></td>
<td>• name - The name of the custom property.</td>
</tr>
<tr>
<td></td>
<td><em>Example</em></td>
</tr>
<tr>
<td></td>
<td>var property = shape001.getCustomProperty(&quot;General&quot;, &quot;BilgePump&quot;);</td>
</tr>
<tr>
<td>objects (string name)</td>
<td>Returns the shape's child object with the specified name or null if not found.</td>
</tr>
<tr>
<td></td>
<td><em>Example</em></td>
</tr>
<tr>
<td></td>
<td>var element = shape001.objects(&quot;alpha001&quot;);</td>
</tr>
</tbody>
</table>
TimeControl Object

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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**

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

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getVisibleDataAsStrings()</td>
<td>Returns an array of the visible data in the trend in a csv friendly format.</td>
</tr>
</tbody>
</table>
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

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

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hairlines</td>
<td>Array</td>
<td>Array of Hairline Object. The hairlines are sorted by date.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hairline(date)</td>
<td>Hairline Object</td>
</tr>
</tbody>
</table>
Hairline Object

*Description*

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

*Properties*

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>number</td>
<td>Returns the date of the hairline.</td>
</tr>
</tbody>
</table>
| values   | Array  | Returns an Array of objects representing where the hairline intersects with traces in the trend. Each object contains:  
|          |        | • traceId   |
|          |        | • value     |

*Methods*

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue</td>
<td>Returns the value for the given traceId at the point where the trace intersects with the hairline.</td>
</tr>
<tr>
<td>(number traceId)</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dockPosition</td>
<td>string</td>
<td>Gets or sets the docking position of the legend. Available options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;float&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;bottom&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;top&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;left&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;right&quot;</td>
</tr>
<tr>
<td>minimized</td>
<td>boolean</td>
<td>Gets or sets whether the legend is minimized.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Gets or sets the visibility of the legend.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>number</td>
<td>Gets the number of traces in a Trend Object.</td>
</tr>
</tbody>
</table>

Example

```javascript
var numberOfTraces = trend001.traces.count;
```

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>Returns the trace object associated with the specified traceId or null if no trace exists with the traceId.</td>
</tr>
</tbody>
</table>

Example

```javascript
var trace = trend001.traces.trace(0);
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTrace</td>
<td>Adds a trace to the trend with the following parameters passed as arguments:</td>
</tr>
<tr>
<td>(string itemName,</td>
<td></td>
</tr>
<tr>
<td>string dataSource,</td>
<td></td>
</tr>
<tr>
<td>string aggregate,</td>
<td></td>
</tr>
<tr>
<td>number sampleInterval,</td>
<td></td>
</tr>
<tr>
<td>string requestUnits,</td>
<td></td>
</tr>
<tr>
<td>number traceId)</td>
<td>• itemName: The item/tag name to add to the trace (e.g. 11AFC01.PV).</td>
</tr>
<tr>
<td></td>
<td>• dataSource: The data source of the tag.</td>
</tr>
<tr>
<td></td>
<td>• aggregate: The aggregate to request from the data source (e.g. &quot;Auto&quot;, &quot;Raw&quot;, &quot;BestFit&quot;). Note that not all data sources support all aggregates.</td>
</tr>
<tr>
<td></td>
<td>• sampleInterval: The requested sample interval in seconds.</td>
</tr>
<tr>
<td></td>
<td>• requestUnits: The unit type to request from the data source. Leave as the empty string (&quot;&quot; to request the default units for the item.</td>
</tr>
<tr>
<td></td>
<td>• traceId – [Optional] The trace id to assign to the new trace. If left blank, the next available traceId is automatically assigned.</td>
</tr>
</tbody>
</table>

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, ");

<table>
<thead>
<tr>
<th>method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>string</td>
<td>Gets or sets the aggregate of the trace.</td>
</tr>
<tr>
<td>color</td>
<td>string</td>
<td>Gets or sets the color of the trace line.</td>
</tr>
<tr>
<td>continuousAutoScale</td>
<td>boolean</td>
<td>Gets or sets if the scale of the trace will be automatically updated to fit the visible data.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Gets the description of the trace.</td>
</tr>
<tr>
<td>id</td>
<td>number</td>
<td>Gets the ID of the trace.</td>
</tr>
<tr>
<td>itemName</td>
<td>string</td>
<td>Gets or sets the item/tag name of the trace.</td>
</tr>
<tr>
<td>lineWidth</td>
<td>number</td>
<td>Gets or sets the width of the trace line.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Gets or sets the visibility of the trace.</td>
</tr>
<tr>
<td>sampleInterval</td>
<td>number</td>
<td>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.</td>
</tr>
<tr>
<td>selected</td>
<td>boolean</td>
<td>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.</td>
</tr>
<tr>
<td>step</td>
<td>boolean</td>
<td>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.</td>
</tr>
<tr>
<td>tagSource</td>
<td>string</td>
<td>Gets or sets the data source for the trace.</td>
</tr>
<tr>
<td>units</td>
<td>string</td>
<td>Gets or sets the unit of measure for the trace.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getYAxisScale</td>
<td>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.</td>
</tr>
<tr>
<td>setYAxisScale ([number minScale, number maxScale])</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>string</td>
<td>Gets or sets the background color of the trace canvas.</td>
</tr>
</tbody>
</table>
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:

   ```javascript
   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:

   ```javascript
   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;
   }
   ```
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:

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

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

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

3. Add a general script to the display:
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:

```javascript
function textbox001_onclick {
    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 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:

```javascript
//////// 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) {
            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:

```javascript
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);

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

```javascript
function NextDay_onclick
{
    // Add a script to the onclick event of the NextDay element:
    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);
}
```

```
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.

```plaintext
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”.

```plaintext
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);
}