Business Value Dashboard

Software Version: 10.63

Developer Guide

Document Release Date: December 2017
Software Release Date: November 2017

Hewlett Packard Enterprise

Legal Notices

Warranty

The only warranties for Seattle SpinCo, Inc and its subsidiaries (“Seattle”) products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Seattle shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Except as specifically indicated, valid license from Seattle required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

AMD, the AMD Arrow symbol and ATI are trademarks of Advanced Micro Devices, Inc.

Citrix® and XenDesktop® are registered trademarks of Citrix Systems, Inc. and/or one more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries.

Google™ and Google Maps™ are trademarks of Google Inc.

Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.

iPad® and iPhone® are trademarks of Apple Inc.

Java is a registered trademark of Oracle and/or its affiliates.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft®, Windows®, Lync®, Windows NT®, Windows® XP, Windows Vista® and Windows Server® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

NVIDIA® is a trademark and/or registered trademark of NVIDIA Corporation in the U.S. and other countries.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

Red Hat® is a registered trademark of Red Hat, Inc. in the United States and other countries.

SAP® is the trademark or registered trademark of SAP SE in Germany and in several other countries.

UNIX® is a registered trademark of The Open Group.

Contents

Developer Guide 4
Custom Widgets 5
Modify the Visio shape 6
Define the custom widget code 9
Send documentation feedback 20

Developer Guide

This guide provides more information on how to extend the BVD functionality.

See the following topics:

- "Custom Widgets" on page 5

Custom Widgets

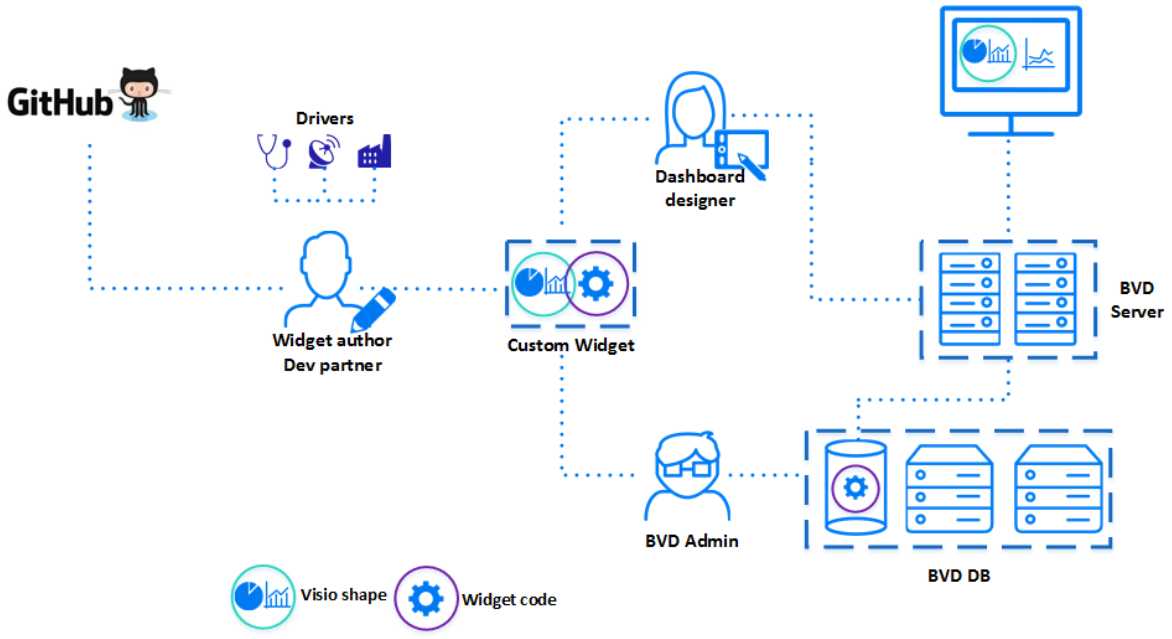
You can create custom widgets to visualize your data in ways other than what is possible by default with the BVD Visio Stencil. By coding your own personalized widgets, you can illustrate your business value in any form required.

To create custom widgets, you must provide BVD with a Visio shape and the corresponding widget code.

- **Visio shape**. A Visio shape acts as the placeholder for your custom widget. The shape is the visual representation of your widget. Depending on the custom widget you want to create, it can be simple or advanced with interactive elements. Additionally, you can save the shape as a stencil so that it becomes reusable. For more information, see "Modify the Visio shape" on the next page.
- **Widget code**. To code how the custom widget behaves, an API is provided which enables you to create the widget by using DOM and SVG manipulation. The result is saved as JavaScript file. For more information, see "Define the custom widget code" on page 9.

After you have uploaded a dashboard containing the Visio shape and saved the JavaScript file on the BVD server, you can view your customized widget in BVD. Whenever you open a dashboard in BVD which contains the custom widget, the JavaScript file is loaded from the BVD server and the custom widget code is executed.

For full code examples, see microfocus.github.io/ColorYourData.



Modify the Visio shape

Custom widgets require a Visio shape that represents their appearance in the dashboard. You can create the shape when you design the dashboard in Visio. Additionally, you can make it reusable by saving it as a stencil. The shape that you create in Visio is the visual representation of your widget in the dashboard. In its simplest form, it is just a shape. But it can be extended to provide interactive elements that define graphical attributes, like the hole size of a donut chart.

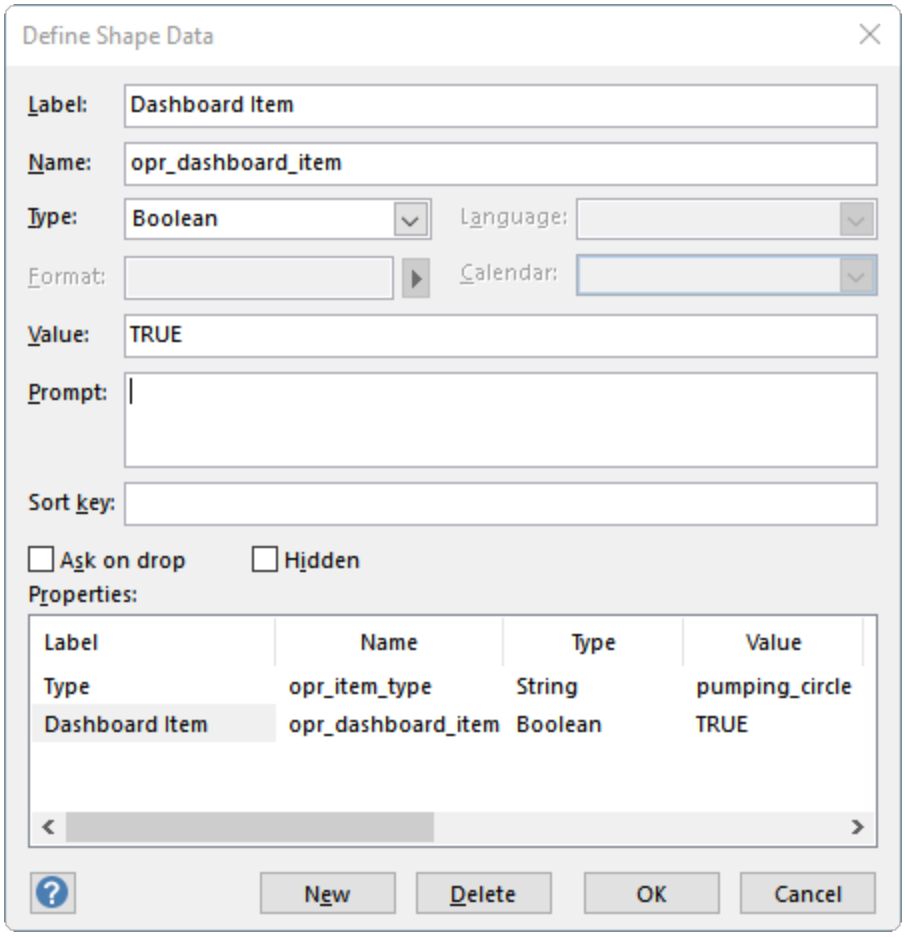
To identify a shape as a custom widget, you must define at least two properties for the shape: `opr_item_type` and `opr_dashboard_item`.

You can define these properties in Visio's shape data menu:

1. Make sure that Visio is running in Developer mode (**Options > Advanced > General > Run in developer mode**).
2. Design your dashboard and add any shape that will later be used as the custom widget.
3. Right click on the shape and select **Data > Define Shape Data...**
4. In the Define Shape Data dialog box, enter the required properties:

```
Name: opr_item_type
Type: string
Value: <type-of-your-widget>
```

```
Name: opr_dashboard_item
Type: Boolean
Value: TRUE
```



These properties will tell BVD that this shape is a widget and needs to be treated like one.

`opr_item_type` is used to internally identify the widget type. When BVD finds such a shape, it looks for a JavaScript file according to the value of the `opr_item_type` property and adds the code to the running dashboard. Do not use `opr` as a prefix for the widget type, as this prefix is reserved for BVD internal use.

> **Example**: The following is an example for a pumping circle (the full pumping circle example is available on GitHub):
>
> ```
> opr_item_type: pumping_circle
> opr_dashboard_item: TRUE
> ```

5. *Optional*. Define the additional shape data property `opr_grouping_item`. For details, see "Widget shape type" on the next page.
6. Save and export the dashboard as SVG.

7. *Optional*. Save the custom widget's shape as stencil. For details, see "Save shape as stencil" below.

Widget shape type

In BVD, there are two types of widget shapes:

- **Real shape**: The shape that represents the real widget appearance. You can modify and adjust the appearance of the shape in your widget code. For example, you can hide the real shape and use your own instead, or you can manipulate the shape by adding and changing SVG elements and attributes.
- **Reference shape (grouping item)**: The shape that acts as a reference for other shapes that need to be grouped together. This type of shape will receive the channel's data, but will display the other shape in the dashboard. This kind of shape is required, for instance, if you want to display a shape that has the rules and data of another shape. Usage examples for this shape type are the Status Color Group and Status Visible Group widgets. To create a Reference shape (grouping item) widget, define one additional shape data property:

  ```
  Name: opr_grouping_item
  Type: Boolean
  Value: TRUE
  ```

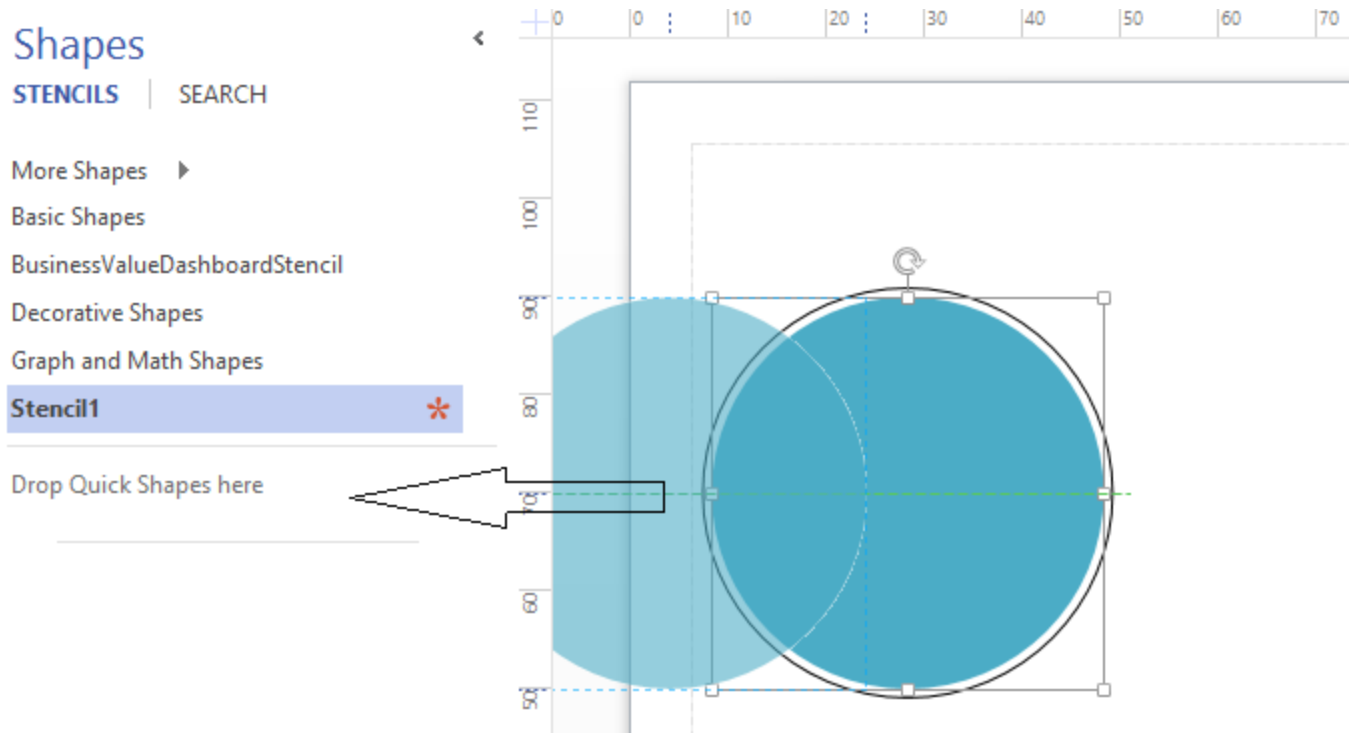
  If this property is defined, the shape is removed from the dashboard and all properties assigned to it will affect the shapes grouped with this shape.

Save shape as stencil

In order to make a custom widget shape reusable, you can save it as a Visio stencil. In the following example, you learn to create a new stencil and save it as a .vssx file so that it can be reused later.

To create a stencil, do the following:

1. In the Shapes panel in Visio, click **More Shapes > New Stencil** (either US Units or Metric).
2. A new stencil is created. If necessary, you can rename this stencil later when saving it to the file.
3. Drag and drop the custom widget shape to the empty area of the stencil list.



Optional. You can rename your custom widget by double clicking the image in the stencil, or by right-clicking and selecting **Rename Master**.

Click the Save Stencil button, or right-click the stencil and select **Save**. The stencil is saved in the .vssx format.

To use this stencil when creating a dashboard, you can load the stencil into Visio. Click **More Shapes > Open Stencil** and select your .vssx file.

Define the custom widget code

To code how the custom widget behaves, a Custom Widget API is provided. The API enables you to create your own widget type by using DOM and SVG manipulation.

See the following sections to learn how to code the custom widget:

- "Prerequisites" on the next page
- "bvdPluginManager API" on the next page
- "Custom Widget configuration UI" on page 15
- " Save the JavaScript file" on page 17

Prerequisites

- **Third-party libraries**. The following third-party libraries are available in the global name space and can be used inside a custom widget:

Variable name	Library	Version
d3	d3.js	3.5.x
$	jQuery	3.2.x
_	lodash	3.10.x

  The AngularJS $http (version 1.6.x) service is passed to the widget as part of its context. Additional libraries needed by a custom widget can be loaded using the `jsLoadService` function described below.

- **Accessing the API**. You can access the API by accessing the `bvdPluginManager` global object directly. In BVD, this service object is assigned in the window scope (`window.bvdPluginManager`), and is therefore made effectively global.

```
window.bvdPluginManager.registerWidget({...}) // access from window's scope
bvdPluginManager.registerWidget({...}) // access it directly
```

bvdPluginManager API

As mentioned in the prerequisites, in order to create a plugin, you have to access the `bvdPluginManager` API. This API only exposes one method:

`registerWidget`

- *Functionality*: This method creates and registers a custom widget according to the widget configuration object that you provide. This method only supports one parameter of the JavaScript object type.
- *Params*: This is the widget configuration object — a plain object that contains a set of supported configuration properties as listed below:

Property	Type	Default value	Description

`id`	object	-	ID of the widget. It must be the same as the value of the `opr_item_type` Visio property and the JavaScript file name of this custom widget.
`displayName`	string	-	Name as it appears in the Dashboard editor.
`init`	function	-	Callback function to initialize the widget with context that is passed back by BVD. This function is where you do most of your custom widget coding. For details, see "Init Plugin Callback " on the next page.
`customProperty`	object	-	Array of objects that is used to render the Custom Widget donfiguration UI.
`hasData`	boolean	true	The value `true` indicates that the widget has a data field. For details, see "Custom Widget configuration UI" on page 15.
`isMultiselectDataField`	boolean	false	The value `true` indicates that the user can select multiple data fields in the widget configuration UI (multi select input field will be shown) if `hasData` is true. Otherwise, only one data field can be selected.
`hasDataChannel`	boolean	true	The value `true` indicates that the widget has a data channel. For details, see "Custom Widget configuration UI" on page 15.

Example: Below is an example of the widget configuration file:

```
{
        id: 'pumping_circle',
        displayName: 'The Pumping Circle',
        init: function(ctx) {
                // widget implementation
        },
        customProperty: [{
                id: 'bvd_range',
                label: 'Range',
                type: 'number',
                default: 100
        }, {
                id: 'opr_coloring_rule',
                label: 'Coloring Rule',
                type: 'text'
        }],
        hasData: true,
        hasDataChannel: true
}
```

Init Plugin Callback

The init callback function is the place where you do all of the coding for your custom widget. This function will be executed by the BVD UI when it loads the widget. The init function receives a context object as a parameter, which has a set of properties as defined below:

Property	Type	Description
`parentGroup`	d3.js object	The parent SVG group of the shapes generated by Visio.
`svgGroup`	d3.js object	An empty SVG group element to attach your own elements (DOM located one level inside the parent group).
`placeHolder`	array of d3.js objects	A list of placeholder shapes generated by Visio.
`dataField`	string	The name of the data field containing the values for this widget.
`bbox`	object	The computed bounding box of the placeholder (x, y, width, height).

`jsLoadService`	function	Service to load external JavaScript files.
`getStatusColor`	function	Function to get the calculated status color.
`$http`	object	AngularJS $http service: can be used by the widget to get additional data from outside of BVD.
`onChange`	function	Widget data change registration function.
`onInit`	function	Widget initial data function (data from cache).
`getProperty`	function	Function to get the value of the widget's property.
`addAdminNotification`	function	Notify users of problems with the widget.
`getChannelProperties`	function	Gets additional information for this data channel.

`dataField` property

Functionality: Contains the name of the property of the data channel that has been selected by the user in the widget configuration to hold the value for this widget. If `isMultiselectDataField` is true, all selected data fields must be listed separated by semicolons.

`getStatusColor` function

Functionality: Get the calculated status color based on the last received value. For details, see the information about the `opr_coloring_rule` in "Special properties" on page 16.

`jsLoadService` function

- *Functionality*: Loads external JavaScript files asynchronously. If you use methods from this external JavaScript file in `onChange` or `onInit`, you must execute `onChange` or `onInit` after the promise of `jsLoadService` is resolved.
- *Params*: location/address of the JavaScript file (string)
- *Return*: promise object

> **Example**:
>
> ```
> ctx.jsLoadService.loadScript(
> 'http://www.example.net/js/great_library.min.js'
> ).then(function success() {
> // do something (e.g. execute onInit or onChange)
> }, function failure() {
> // do something
> });
> ```

`onChange` callback function

- *Functionality*: Register an event when the widget receives a data channel update.
- *Params*: plain object which contains these properties:
  - channel: the channel name (optional)
  - callback: the change handler (will be called with the data received from the BVD server)

```
ctx.onChange({
        channel: 'myChannel',
        callback: function(dataUpdateEnvelope) {
                if (dataUpdateEnvelope && dataUpdateEnvelope.data) {
                // update displayed date with content of dataUpdateEnvelope.data
                }
        }
});
```

`onInit` callback function

- *Functionality*: Get the initial data displayed by this widget.
- *Params*: plain object which contains these properties:
  - channel: the channel name (optional)
  - itemCount: number of items from cache (max 9.999.999) or timestamp (Unix time ms). Using a timestamp will return all data from that time until now.
  - callback: the init handler (will be called with the data received from the BVD server)

```
ctx.onInit({
        channel: 'myChannel',
        itemCount: 1,
        callback: function(envelopeArray) {
                if (Array.isArray(envelopeArray) && envelopeArray.length) {
                // display initial data inside widget
                }
        }
});
```

`getProperty` function

- *Functionality*: Get the value of the widget's property, which is defined in the `customProperty` object of the widget configuration. For details, see "Custom Widget configuration UI" on the next page.
- *Params*: property name (string)

- *Return*: object of property value

```
var range = ctx.getProperty('bvd_range');
```

`addAdminNotification` function

- *Functionality*: Notify the current user of problems with the widget. Users with administrative rights will see these notifications in BVD next to their user name.
- *Params*: object which contains these properties:
  - severity: `info`, `warning`, or `error` (string)
  - title: the title of this notification (string)
  - text: the notification text (string)
- *Return*: none

`getChannelProperties` function

- *Functionality*: Get additional information for the selected data channel.
- *Params*: none
- *Return*: object which contains the property `isRealtime` (boolean). If `isRealtime` is false, this data channel does not get updated and contains historical data only. Widgets can skip updating themselves after the initial data is received.

Custom Widget configuration UI

Custom widgets require a widget configuration UI for the Dashboards page. This UI is generated by the `bvdPluginManager` API.

The BVD Plugin Manager generates the UI based on the `customProperty` field in the widget configuration. `customProperty` is an array of objects that must contain the following set of properties:

Property	Type	Description
`id`	string	Unique ID of the property.
`type`	string	String of any supported input type: `number`, `text`, or `channel`.
`label`	string	Label that will be displayed on the Dashboards page.
`mandatory`	boolean	*Optional*. Indicate that the property is mandatory for proper functionality of the widget (default is `false`).

`default`	object	*Optional*. Default value of this property.

Example:

```
{
        ...
        customProperty: [
                {
                        id: 'my_custom_property',
                        label: 'My Custom Label',
                        type: 'number',
                        default: 100,
                        mandatory: true
                },
                {
                        id: 'my_custom_channel',
                        label: 'My Custom Channel',
                        type: 'channel'
                }
        ]
        ...
}
```

Do not use the prefix `opr` for custom properties IDs, as this prefix is reserved for BVDinternal use. The only exception is the ID `opr_coloring_rule`.

Default properties

The following properties are created for each custom widget and cannot be removed:

ID	Description
`opr_field`	Created if `hasData` is set to true in the `registerWidget` property.
`opr_channel`	Created if `hasDataChannel` is set to true in the `registerWidget` property.
`opr_visibility_rule`	Created if `hasDataChannel` is set to true in the `registerWidget` property. This allows the user to control the visibility of this widget. For more information, see Visibility Rule.
`opr_hyperlink`	Always created. Allows the specification of an URL which is opened when the user clicks on this widget.

Special properties

`opr_coloring_rule` string

Functionality: If a custom property with this ID is added to a widget, BVD will calculate a status color based on this string, as described in Status Color Group. Custom widgets can get the resulting color string from the `getStatusColor` function of the Init Plugin Callback. This color can be used in the `onChange` and `onInit` handlers to color the widget or parts of the widget.

Save the JavaScript file

After you created the custom widget code, you must save it as a JavaScript file on the BVD server to be able to use it.
Place your custom widget code into the directory `<conf_volume>/bvd/var/bvd/widgets`
`<conf_volume>` is the NFS directory for suite-related configuration files that you specified during the installation. The proposed directory is `/var/vols/itom/conf`.

The file name must match the widget configuration ID: `<widget_configuration_id>.js`.
When you open a dashboard in BVD which contains this custom widget, its JS file is loaded from this directory. The custom widget code is then executed, and the new custom widget will be registered in BVD.

Below is an example of how you can use the `bvdPluginManager.registerWidget({})` with your widget configuration file. This example will create a custom widget called pumping circle.

Example:

```
'use strict';

/**
* The pumping circle widget
* This widget shows a circle that changes its radius
* based on the data coming through the channel
*/

bvdPluginManager.registerWidget({
        id:'pumping_circle',
        displayName:'The Pumping Circle',


init: function(ctx) {
const circleGroup= ctx.svgGroup,
        range = ctx.getProperty('bvd_range') ||100

       console.log('range = '+ range);

       /* hide the placeholder */
       ctx.placeHolder.attr('style', 'visibility: hidden;');

       const circle = circleGroup
        .attr('transform', 'translate('+ ((ctx.bbox.x+ ctx.bbox.width/2)) +','+
(ctx.bbox.y+ ctx.bbox.height/2) +')')
        .append('circle')
        .attr('r', ctx.bbox.width/4)
        .attr('cx', 0)
        .attr('cy', 0);

       const pumpCircle=function(envelope) {
if (!envelope ||!envelope.data) {
return;
        }

       const currentColor= circle.attr('fill'),
        msg = envelope.data,
        radius = msg[ctx.dataField] / range * ctx.bbox.width/2;

        circle.transition()
        .duration(300)
        .attr('r', radius)
        .attr('fill', ctx.getStatusColor() || currentColor);
        };

/* get initial state of this widget*/
ctx.onInit({
        itemCount:1,
callback: function(envelopeArray) {
if (envelopeArray && envelopeArray.length>0) {
pumpCircle(envelopeArray[0]);
                        }
```

```
                }
        });

/* subscribe to changes */
ctx.onChange({
        callback: pumpCircle
        });
},

customProperty: [{
        id:'bvd_range',
        label:'Range',
        type:'number',
        default:100
}, {
        id:'opr_coloring_rule',
        label:'Coloring Rule',
        type:'text'
        }]
});
```

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Developer Guide (Business Value Dashboard 10.63)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to ovdoc-asm@hpe.com.

We appreciate your feedback!