09 May 2023

Responsive heAder

VERSION: 1.0.2

# **Overview**

 A sample header showcasing how to build reusable components for responsive web.

## **Use case:**

A sample header showcasing how to build reusable components for responsive web.

## **Percentage of re-use:**

80-90%.

# **Getting Started**

## **Prerequisites**

Before you start using the Responsive Header component, ensure the following:

• [HCL Foundry](https://manage.hclvoltmx.com/)

• Volt MX Iris

## **Platforms Supported**

### PWA

## **Importing the app**

## You can import the Forge components only into the apps that are of the Reference Architecture type.

##  **To import the Responsive Header component, do the following:**

## Open your app project in Volt MX Iris.

 2. In the Project Explorer, click the **Templates** tab.

 

3.Right-click **Components**, and then select **Import Component**. The **Import Component** dialog box appears.



4.Click **Browse** to navigate to the location of the component, select the component, and then click **Import**. The component and its associated widgets and modules are added to your project.

##

Once you have imported a component to your project, you can easily add the component to a form. For more information, refer [Add a Component to a Form](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/C_UsingComponents.html%22%20%5Cl%20%22add-a-component-to-a-form)

## **Building and previewing the app**

After performing all the above steps, you can build your app and run it on your device. For more information, you can refer to the [Building and Viewing an Application](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/Cloud_Build_in_VoltMX_Iris.html#cloud) section of the Volt MX User Guide.

 You can then run your app to see the Responsive Header work in real time.

# **References**

## **Dynamic Usage**

You can also add **Responsive Header component** dynamically. To do so,

1. In Project **Explorer**, on the **Projects** tab, click **Controllers** section to access the respective **Form Controller**. Create a method and implement the code snippet like the sample code mentioned below.

createComponent: function(){

var ResponsiveHeader = new com.mig82.ResponsiveHeader({

 "height": "50dp",

 "id": "ResponsiveHeader",

 "isVisible": true,

 "left": "0dp",

 "masterType": constants.MASTER\_TYPE\_USERWIDGET,

 "skin": "headerFlexInvertedSkin",

 "top": "0dp",

 "width": "100%",

 "zIndex": 1

 }, {}, {});

this.view.add(ResponsiveHeader);

},

In the code snippet, you can edit the properties of the component as per your requirement. For more information, see Setting Properties.

2. Save the file

**External Dependencies**

This project uses AmplifyJs's core module. This is bundled within the component so once the component is imported this JavaScript module will be added to the project's global modules directory.

**Responsiveness**

In a VoltMX responsive web application, changing the width of the browser will fire the ***FlexForm***'s **onBreakpointChange** event handler. From there, in order to also notify any component instances in the form of this event, the ***FlexForm***'s event handler could directly invoke a custom method in each component that's meant to be responsive -e.g.:

// A tightly coupled form controller.

define({

 onBreakpointChange: function(form, width) {

 this.view.FooComponent.adjust(width);

 this.view.BarComponent.adjust(width);

 // For each and every component in the form.

 this.view.QuxComponent.adjust(width);

 },

 onNavigate: function() {

 this.view.onBreakpointChange = this.onBreakpointChange;

 }

});

The downside with this approach is two-fold. First, it forces you to [expose this hypothetical custom adjust method](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/C_CreatingComponent.htm#Methods) for each component.

Second, it tightly couples the form controller to the components. You have to call this hypothetical custom adjust method for each and every component in the form and add or remove calls as you add or remove components to it, resulting in less maintainable codebase.

**We can do better...**

Instead of that, this project uses the [Publish-Subscribe Pattern](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern), by leveraging [AmplifyJs's core module](http://amplifyjs.com/api/pubsub/) to achieve low coupling between each form and the instances of the components inside it. This way, the form's controller stays lean, and you do not need to update it to keep it in sync with whatever components you've added to it. Here's an example in a standard barebones form controller:

//A loosely coupled form controller.

define({

 onBreakpointChange: function(form, width) {

 amplify.publish("onBreakpointChange", form, width);

 },

 onNavigate: function() {

 this.view.onBreakpointChange = this.onBreakpointChange;

 }

});

Then, each component just listens for the onBreakpointChange event using AmplifyJs as well:

//A loosely coupled component controller.

constructor: function(baseConfig, layoutConfig, platformSpecificConfig) {

 //Then let's subscribe to the form's onBreakpointChange.

 amplify.subscribe(

 "onBreakpointChange", //topic

 this, //context

 this.onBreakpointChange, //callback

 1 //priority

 );

},

onBreakpointChange: function(form, width){

 // Here we adjust whatever widgets need to be resized,

 // repositioned, made visible/invisible or re-skinned

 // depending on the new screen width.

},

Components implemented like this can be seamlessly added to a form, without the need to expose additional custom methods and without having to code additional invocations in the form's controller.

### **Resources**

* [Source code at Github.com](https://github.com/mig82/responsive-header-component)
* [How to create responsive reusable components](https://basecamp.kony.com/s/article-detail/a042K000016Q3h3QAC/how-to-create-a-responsive-reusable-component)

# **Revision History**

App version 1.0.2