Date : 27 Sep 2021

Virtual keYboard

VERSION: 1.0.1

# **Overview**

Virtual Keyboard allows users to input text without the need of physical keys.

Virtual Keyboard is a web component that generates a virtual keyboard for a user's input.

## **Use case:**

You can use this component in scenarios such as: A banking website where you want to provide a virtual keyboard to the user to enter the password. You can also enable the shuffling of the keys to enhance security.

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

80-90%.

# **Getting Started**

## **Prerequisites**

Before you start using the Virtual keyboard 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 Virtual keyboard 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.

## Text  Description automatically generated

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 Virtual Keyboard working in real time.

# **References**

## **Dynamic Usage**

You can also add **Virtual Keyboard** 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 similar to the sample code mentioned below.

 createComponent: function()

{

/\* Creating the component's object \*/

var VirtualKeyboard = new com.voltmxmp.virtualkeyboard(

{

 "clipBounds": true,

 "height": "30%",

 "id": "VirtualKeyboard",

 "isVisible": true,

 "left": "0dp",

 "top": "0dp",

 "width": "70%",

 "zIndex": 1

}, {}, {});

/\* Setting the component's properties \*/

VirtualKeyboard.shuffleOnClick = true;

/\* Defining the component's events \*/

VirtualKeyboard.onKeyClicked = function(keyText)

{

 //Entering the user input in a Text Box widget

 this.view.textBoxWidget.text = keyText;

}.bind(this);

/\* Adding the component to a Form \*/

 this.view.add(VirtualKeyboard);

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.

## **Properties**

The properties provided on the **Component** tab allow you to customize the elements in the **Virtual keyboard** component. These elements can be UI elements, service parameters, and so on. You can set the properties from the Volt MX Iris Properties panel on the right-hand side. You can also configure these properties using JavaScript code.

**General Properties**

**1.** **Shuffle on Each Click**

|  |  |
| --- | --- |
| **Description:** | Specifies whether the keyboard should shuffle the keys after every click. |
| Syntax: | shuffleOnClick |
| **Type:** | Boolean |
| **Read/Write:** | Read+Write |
| **Default Value:** | true |
| **Example:** | this. view. componentID.shuffleOnClick= true; |

**Skin Section**

**1.Normal**

|  |  |
| --- | --- |
| **Description:** | This skin links to the buttons of the keyboard. |
| **Syntax**: | btnSkin |
| **Widget Type:** | Button |
| **Example:** | this. view. componentID.btnSkin= “buttonSkin”; |

**2. Focus Skin**

|  |  |
| --- | --- |
| **Description:** | This skin links to the buttons of the keyboard when they are in focus. |
| **Syntax**: | btnFocusSkin |
| **Widget Type:** | Button |
| **Example:** | this. view. componentID.btnFocusSkin= “buttonFocusSkin”; |

## **Events**

You can define events to be executed when an action is performed. You can configure the events directly on the Actions tab or by writing a JavaScript. To configure the events on the Actions tab, click Edit against each event. For more information, refer Add Actions.

This section provides details about each event that help you define the actions by writing a JavaScript.

**1. onKeyClicked**

|  |  |
| --- | --- |
| **Description:** | Invoked when the user clicks a key on the keyboard. |
| **Syntax:** | onKeyClicked |
| **Parameters:** | *keyText [String]*:The text on the button that the user clicked on the keyboard. |
| **Remarks:** | The component joins the previous inputs with the current input of the keyboard every time the user clicks a button. |
| **Example:** | this.view.componentID.onKeyClicked = function(keyText){//Entering the user input in a Text Box widgetthis.view.textBoxWidget.text = keyText;}.bind(this); |

## **APIs**

The following APIs pertain to the Virtual Keyboard component:

**1. getuserInput**

|  |  |
| --- | --- |
| **Description:** | Fetches the entire input that the user entered from the keyboard. |
| **Syntax:** | getUserInput() |
| **Parameters:** | None |
| **Return Value:** | *userInput [String]* :The entire input that the user entered from the keyboard. |
| **Example:** | var userInput = this.view.componentID.getUserInput(); |

# **Revision History**

App version 1.0.1

## **Known Issues**

None