Date: 22-Oct-2021

Animated text FIELD

version: 1.1.1

# **overview**

Energize your textboxes using this bigger and better version. The animated textboxes component creates better user interactions, giving superior quality to your application, right out of the box. Labels for text entry fields can be animated thereby creating an enhanced user experience for the end user. This also enables greater visibility for the label, which makes the data entry operation seamless. This component can be used in almost any situation where any kind of data entry is involved.

## **Use case:**

* Consider a case that you are developing a registration form, which is one of the screens in your mobile app. In the registration form, you can use the Animated Text Field component for all data entry fields to enhance the user experience.

## **Features**

* Text box with animation to placeholder text when the user taps it
* Advanced and intuitive UI element for mobile apps
* Enhanced user experience and interaction
* Useful component as a data entry field.

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

90%

# **Getting Started**

## 

## **Prerequisites**

Before you start using the Animated Text Field component, ensure you have the following:

* [HCL Foundry](https://manage.hclvoltmx.com/)
* Volt MX Iris

## **Platforms Supported**

### Mobile

#### iOS

#### Android

### Tablets

### PWA

## **Importing the app**

Before you start importing the component to Volt MX Iris, you must download the component from the HCL Volt MX Marketplace website.

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

**To import the Animated Text Field component, do the following:**

1. Open your app project in Volt MX Iris.
2. In the Project Explorer, click the **Templates** tab.

Text

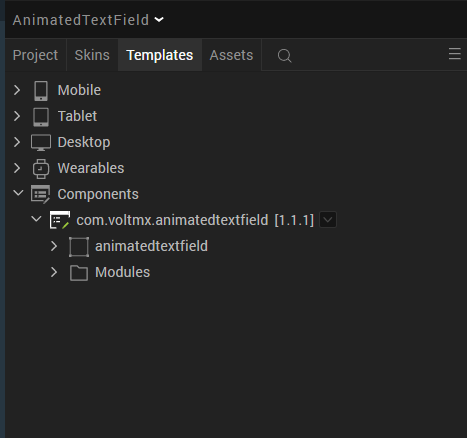
Description automatically generated

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

Graphical user interface, text, application

Description automatically generated

1. 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.
2. In the **Component Name** box, the **com.voltmx.animatedtextfield** name is displayed by default.



1. 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#add-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 Iris User Guide.

You can then run your app to see the Animated Text Field work in real time.

# **References**

## **Dynamic Usage**

If you want to use the Animated Text Field Form component dynamically, you will need to import the component into your project Templates. Follow the given steps to do so

### Download the component from HCL VoltMX Marketplace as a zip file.

### Go to the Templates tab in your project explorer.

### Right click on Components and select Import Component.

### Navigate to where you downloaded your zip file and import it into Iris.

After you import the component into your project templates, you can add it to your app dynamically. To do so, follow the given steps.

### Access the FormController of the form you want to add the component into.

### Create a function called createComponent(); and write the code inside it to create and configure the component.

### You can call the createComponent() function based on your project requirement. [for eg: onNavigate, postShow , preShow].

### You can refer to the given sample code for more information.

/\* creating a component's Object \*/

var textboxComponent = new com.voltmx.animatedtextfield(

{

"clipBounds": true,

"height": "10%",

"id": "textbox",

"isVisible": true,

"centerX": "50%",

"centerY": "50%",

"width" : "80%",

"skin": "slFbox",

"zIndex": 1

}, {}, {});

/\* Setting component's properties \*/

textboxComponent.placeholderText = "Username";

textboxComponent.maxTextLength = 10;

textboxComponent.secureTextEntry = false;

textboxComponent.placeholderSkin = "sknPlaceholder";

textboxComponent.placeholderFocusSkin = "sknPlaceholderFocus";

textboxComponent.underlineSkin = "sknUnderline";

textboxComponent.underlineFocusSkin = "sknUnderlineFocus";

/\*Adding Component to the Form\*/

this.view.add(textboxComponent);

You can use a component's **Properties** to customize and configure the elements. These elements can be UI elements, service parameters, and so on. You can set the properties from the Iris's Properties panel on the right-hand side. You can also configure these properties using a JavaScript code.

## **Properties**

The properties provided on the **Component** tab allow you to customize the UI elements in the Animated Text Field component. You can set the properties directly on the **Component** tab or by writing a JavaScript. This section provides information about how to set the properties by writing a JavaScript.

**General**

#### **1. Placeholder Text**

|  |  |
| --- | --- |
| **Description:** | Specifies a placeholder text to be displayed in the Text field component. |
| **Syntax:** | placeholderText |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.placeholderText = "Sample Text"; |

#### **2. Maximum Characters**

|  |  |
| --- | --- |
| **Description:** | Specifies the maximum number of characters accepted by the Text field. Using the property, you can limit a user to enter the text up to a specified length. |
| **Syntax:** | maxTextLength |
| **Type:** | Number |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.maxTextLength = 12; |

**3. Mask Text**

|  |  |
| --- | --- |
| **Description:** | Controls whether or not to mask the text in the Text field. |
| **Syntax:** | secureTextEntry |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Remarks** | If the secureTextEntry field is set to true, the Text field behaves as a password field that masks the text entered. |
| **Example:** | this.view.componentID.secureTextEntry = true; |

**Skin**

**4. Placeholder Skin**

|  |  |
| --- | --- |
| **Description:** | Specifies the skin to be applied to the placeholder text. |
| **Syntax:** | placeholderSkin |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Remarks** | * Before you set the property, ensure that the skin ID that you specify already exists in your app project. * The skin ID assigned to the property must be the skin of the Label. |
| **Example:** | this.view.componentID.placeholderSkin = "sknPlaceholder"; |

**5. Placeholder Focus Skin**

|  |  |
| --- | --- |
| **Description:** | Specifies the skin to be applied to the placeholder text when the Text field is in focus. |
| **Syntax:** | placeholderFocusSkin |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Remarks** | * Before you set the property, ensure that the skin ID that you specify already exists in your app project. * The skin ID assigned to the property must be the skin of the Label. |
| **Example:** | this.view.componentID.placeholderFocusSkin = "sknPlaceholderFocus"; |

**6. Underline Skin**

|  |  |
| --- | --- |
| **Description:** | Specifies the skin to be applied to the underline below the Text field. |
| **Syntax:** | underlineSkin |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Remarks** | * Before you set the property, ensure that the skin ID that you specify already exists in your app project. * The skin ID assigned to the property must be the skin of the Label. |
| **Example:** | this.view.componentID.underlineSkin = "sknUnderline"; |

**7. Underline Focus Skin**

|  |  |
| --- | --- |
| **Description:** | Specifies the skin to be applied to the underline when the Text field is in focus. |
| **Syntax:** | underlineFocusSkin |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Remarks** | * Before you set the property, ensure that the skin ID that you specify already exists in your app project. * The skin ID assigned to the property must be the skin of the Label. |
| **Example:** | this.view.componentID.underlineFocusSkin = "sknUnderlineFocus"; |

## **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 the **Edit** button against each event.

* **onDone**

|  |  |
| --- | --- |
| **Description:** | The event is invoked when a user clicks on the keyboard action label. |
| **Syntax:** | onDone() |

## **APIs**

### **1. getText()**

**Description :** The API retrieves the text entered in the Text field.

**Syntax:**  getText ()

**Parameters:** None

**Return Value: Text [String] :** Returns the text entered in the Text field

**Example:**

var textboxText = this.view.componentID.getText();  
alert(textboxText);

# **Revision History**

App version 1.1.1:

## **Limitations**

* Land Scape mode is not supported.
* The text box size should be adjusted according to the screen size.