28 Mar 2024

**GEOLOCATION FEATURE APP**

**VERSION: 1.0.1**

# **Overview**

The GeoLocation Feature App highlights the usage and capabilities of the Geolocation API in Quantum stack.

This app highlights three basic functionalities of the API:

* Getting current position
* Watching position
* Clear Watch

## **GeoLocation Feature App Functionality**

## This section explains the functionality of the Geolocation Feature App. It is a feature app as it highlights the features and capabilities of the Geolocation API in the Quantum stack.

## **Geolocation API:**

## The Geolocation API defines a high-level interface to location information, such as latitude and longitude associated with the mobile device. The API uses the common sources of location information like Global Positioning System (GPS) to infer details like IP address, RFID, WIFI and Bluetooth MAC addresses, and GSM/CDMA cell IDs. This API does not always return the device's actual location.

For more information on Geolocation API, you can refer to the following documents:

* [Geo Location Api](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_api_dev_guide/content/geolocation_api_watchposition.html)
* [Voltmx.location Namespace](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_api_dev_guide/content/voltmx.location_functions.html)

## The GeoLocation Feature App has two screens:

* Landing Screen - Contains all the features that the app highlights.
* Feature Screen - Shows the feature that is selected on the landing screen.
* Knowledge Framework - It is an additional layer that contains a list of key features covered in the app.

The features that this app highlights are:

* Get Current Position
* Watch Position and Clear Watch

**Get Current Position**

On tap of the Current Position row on the landing page, the app shows the Current Position Feature screen. The details present on the screen correspond to the current location of the user's device. This functionality makes use of the getCurrentPosition function in the voltmx.location namespace. The values that the function returns in the success call back are put on the screen.

These values are described in the following table.

|  |  |
| --- | --- |
| **Latitude:** | The latitude of the device in decimal degrees. |
| **Longitude:** | The longitude of the device in decimal degrees. |
| **Altitude:** | The height of the location in meters. This height is calculated from the ellipsoid of the Earth. |
| **Accuracy:** | The accuracy level of the latitude and longitude coordinates in meters. |
| **Heading:** | The direction of travel in degrees. These degrees are counted clockwise relative to the true north. |
| **Speeding:** | The current ground speed of the device in meters per second. |
| **Timestamp:** | The time when the position object was acquired. |

### **Watch Position and Clear Watch**

On tap of the Watch Position row on the landing page, the app shows the Watch Position Page. The details present on the screen correspond to the current location of the user's device. This functionality makes use of the watchPosition function in the voltmx.location namespace. This function returns the same parameters as the getCurrentPosition function. The difference is that watchPosition returns these values every time the position of the devices changes. The watch operation continues till the clearWatch function is called with the corresponding identifier.

### **Knowledge Framework**

The Knowledge Framework screen of the app is accessible by tapping on the blue floating button at the bottom right corner of the screen. On tap of the Knowledge Framework button, the app presents a layover screen with a list of the key features covered in the app.

Upon selecting an item from the list presented, you are navigated to a detailed page containing the What and How of the corresponding key feature along with the appropriate links to the documentation and tutorials.

The What s it? section provides an overview of the feature whereas the How to use? section contains usage information of the feature. The latter may include blogs, video tutorials or code fragments as applicable. This app carries all the code fragments used to render the UI or functionalities in the app.

# **Getting Started**

## **Prerequisites**

Before you start using the Geolocation Feature App, ensure you have the following:

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

• Volt MX Iris

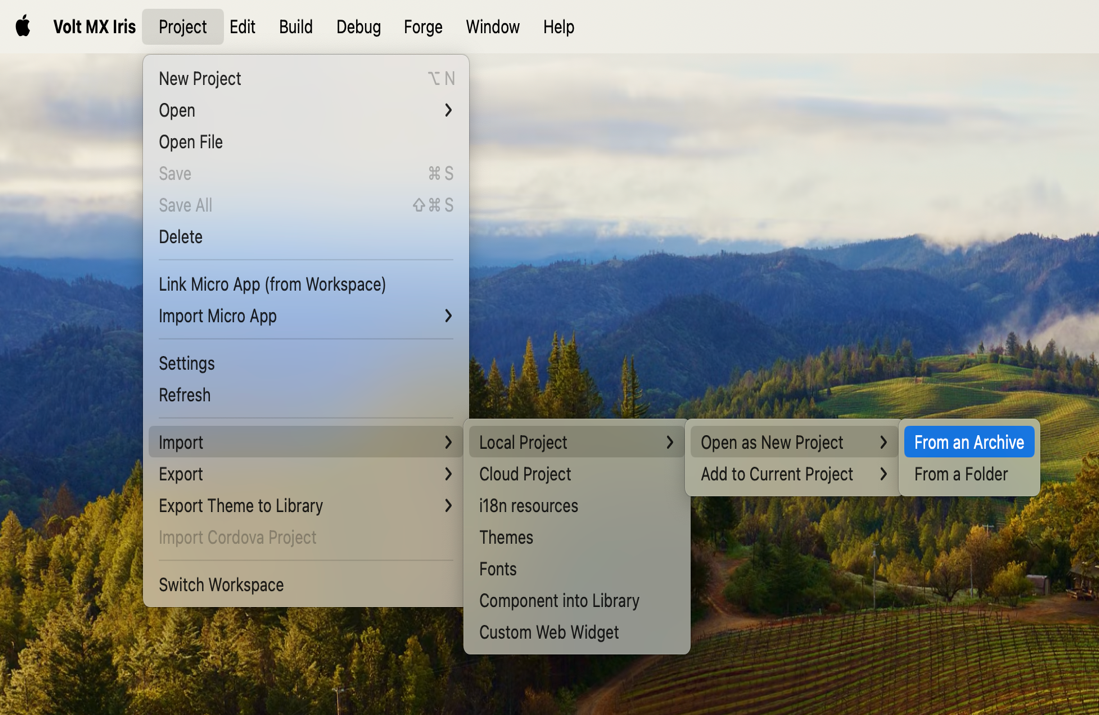
## **Platforms Supported**

### Mobile

#### iOS

## **Importing the GeoLocation Feature App to Visualizer**

1. Open a web browser and enter <https://marketplace.hclvoltmx.com/>in the address field. The Forge page displays with the list of components.
2. Search for **GeoLocation Feature App** and click the corresponding thumbnail. The **GeoLocation Feature App** detail page appears.
3. Click **Login to Download** in case you are not logged in already. Otherwise, click **Download**. The app is downloaded to your local system in a **zip** format.
4. Open VoltMX Iris App and Sign In.
5. Starter:
   1. On the **Project** menu, point to **Import**, and click **Local Project**., select **Open as New Project** and **select From an Archive.**



* 1. Browse and select the downloaded zip file.
  2. Click **Open**.

## **Viewing the GeoLocation Feature App on your device**

## You can view the GeoLocation Feature App on your device by using the VoltMX Mobile App.

## To view the GeoLocation Feature app on your device, do the following:

## Once the app is imported into the iris, Click on Build > Publish Live Preview.

## A dialog box appears with a set of channels. Select the appropriate channels and click on **Publish.**

1. A dialog box will appear with a five digit **preview code**.

## On the device open voltmx app, on the landing screen select cloud, Type in the five-digit preview code that you got from Voltmx iris then click Launch.

## **Configuring the UI of the GeoLocation Feature App**

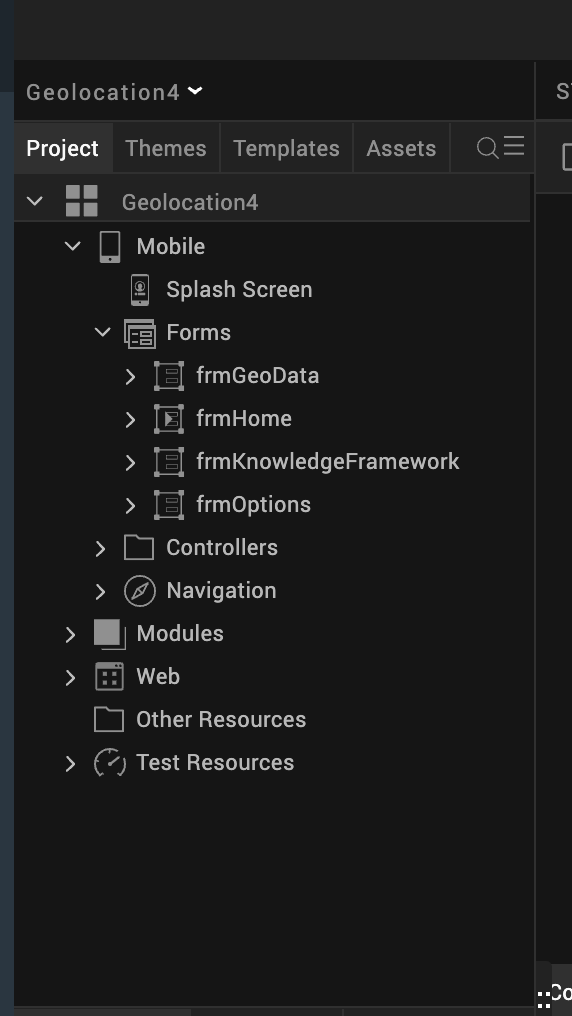
The GeoLocation Feature App comes with a pre-configured UI. You don't need to make any changes to the wiring or coding to make the app work. You can import the app to Visualizer Enterprise and build it to see it working. However, if you need to modify the UI of the app, you can follow the guidelines given below.

The UI of this app contains **four** forms:

* frmHome
* frmOptions
* frmGeoData
* frmKnowledgeFramework

To modify the UI of **GeoLocation Feature App**, you can follow the guidelines given below.

1. Import the project to VoltMX Iris.
2. Under the **Project** tab, expand **Mobile** > **Forms**. You will see all four forms as shown in the given image.



**frmHome** is used as a startup form. This form describes the app. Expand **frmHome** to see the detailed structure of the form, as shown in the given image.

A screenshot of a computer

Description automatically generated

* **flxRoot** is a flex container in a **Flow Vertical** layout. This holds all other widgets on this form.
* **flxHeader** is a flex container that holds a label widget, **lblHeader**, to show the title of the form.
* **lblInfoHeader** is a label widget used to show the description of the app.
* **flxInfo1**, **flxInfo2**, and **flxInfo3** are three flex containers that hold one image and one label widget to show a bullet list of options.
* **btnContinue** is a button widget used to navigate to the form **frmOptions**.
* **btnFramework** is a button used to navigate to the form **frmKnowledgeFramework**.

To change the positioning of any widget, follow these steps:

1. Expand the widgets in the **Project** tab under **Mobile** > **Forms** > **frmHome**.
2. Choose the widget for which you want to change the positioning.
3. Go to the Look tab under the Properties window on the right side in Visualizer.
4. Here, you can change the values for Left, Right, Top, Bottom, Width, Height, CenterX, CenterY, Z Index, and so on.

To change the skin of any widget, follow these steps:

1. Expand the widgets in the **Project** tab under **Mobile** > **Forms** > **frmHome**.
2. Choose the widget for which you want to change the skin.
3. Go to the **Skins** tab under the **Properties** window on the right side in Visualizer.
4. Here, you can change the values for Background Color, Background Opacity, Border Size, Border Color, Font Size, Font Family, Font Color, and so on.

The onClick action of the **Continue** button navigates to the form **frmOptions**. To achieve this, you can write the snippet given below to the onClick action of the **Continue** button.

var ntf = new voltmx.mvc.Navigation ("frmOptions");

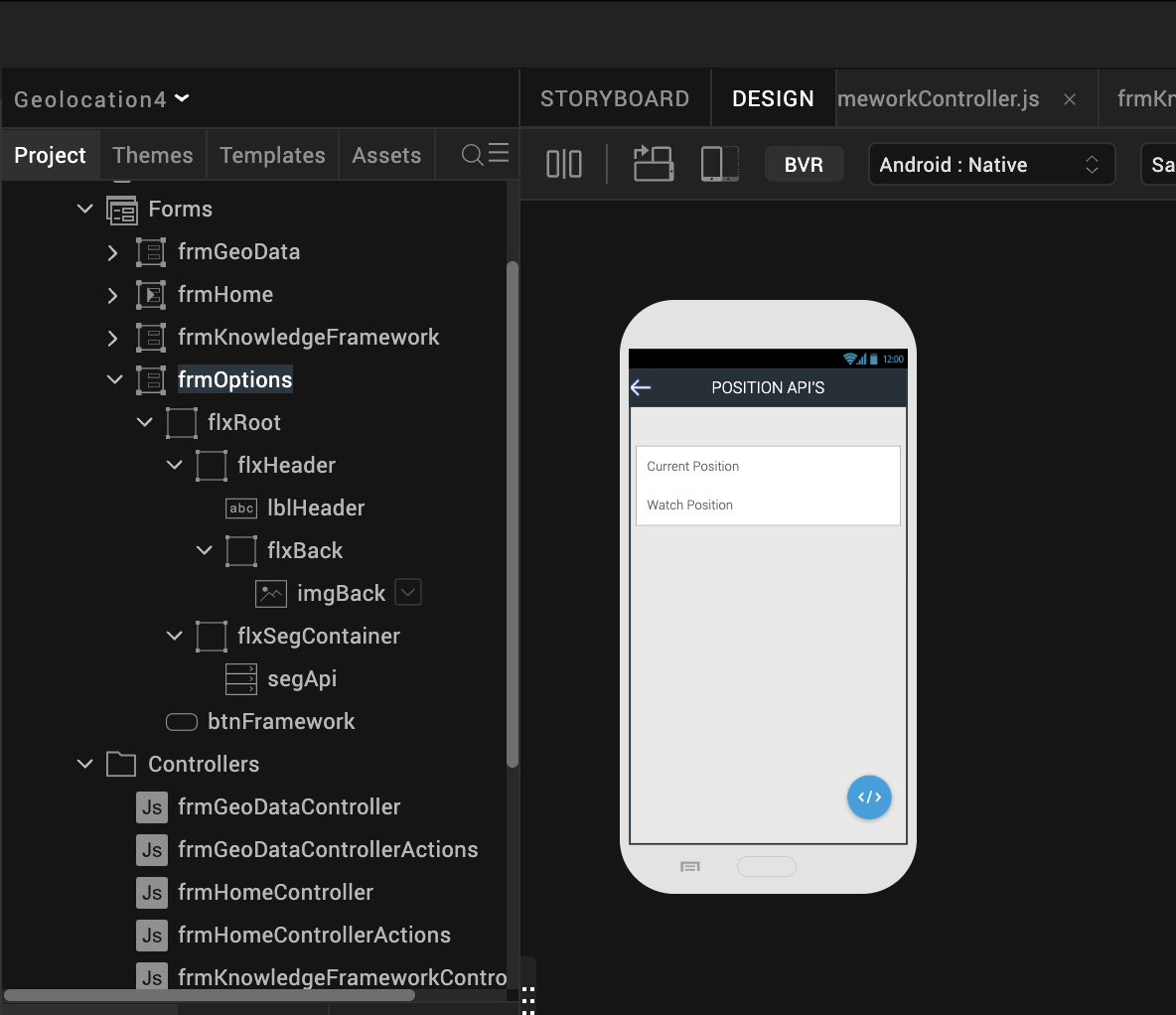
ntf.navigate();

You can also use the **Action Editor** to navigate to **frmOptions**. To achieve this, follow these steps:

1. Select the Widget.
2. Click on the **Action** tab in the **Properties** window on the right side.
3. Click on the **Edit** button that is opposite to **onClick**.
4. Select **Navigate to Form**.
5. Select **frmOptions** from the list.

The onClick action of the Knowledge Framework button in the application navigates to the form frmKnowledgeFramework. To achieve this, you can follow the same steps as navigating to frmOptions.

**frmOptions** is the form used to show the list of features that are available under the Geolocation API. Expand **frmOptions** to see a detailed structure of the form, as shown in the given image.

****

* **flxRoot** is a flex container in a **Flow Vertical** layout. This holds all other widgets on this form.
* **flxHeader** is a flex container that holds a label widget, **lblHeader**, to show the title of the form. It also holds a flex container, **flxBack**, to navigate to the previous form.
  + - **flxBack** is a flex container that holds an image widget, **imgBack**, that holds the image for the back arrow.
* **flxSegContainer** is a flex container which holds a Segment widget, **segApi**.
  + **segApi**, is a segment widget used to show a list of APIs, which is a repetitive UI that is defined as **tempSegTitle** under the **Templates** tab in the **Project Explorer** window on the left side of Iris, as shown in the given image.

A screenshot of a computer

Description automatically generated

* + **tempSegTitle** contains one label widget, **lblTitle** to show the API name.
  + To set data into the segment, select the segment widget and click on the **Edit** option of the **Master Data** in the properties window.
  + A window will popup and you can provide the list of data in it, as shown in the given image.
* **btnFramework** is a button that has its **onClick** action set to navigate to **frmKnowledgeFramework**.
* The **onRowClick** action of the segment **segApi** navigates to the form **frmGeoData**. It also sends a parameter to invoke the appropriate API in **frmGeoData**.
* To configure the positioning or skin of any widget, you can follow the [steps](https://docs.kony.com/marketplace/V8Marketplace/Content/Marketplace/geolocationapp.htm#steps2) mentioned earlier.
* **onSegRowClick** is the function that is invoked in the **onRowClick** action of **segApi**.  
  The code for the **onSegRowClick** function can be found in the **frmOptionsController**.

onSegRowClick:function (eventobj)

{

var selectedIndex=this.view.segApi.selectedRowIndex[1];

if(selectedIndex===0)

{

this.navObj.navigate("GEO\_POSITION");

}

else if(selectedIndex===1)

this.navObj.navigate("WATCH\_POSITION");

}

* **navObj** is a global variable pointing to **frmGeoData** which gets initialized in the **onNavigate** event of **frmOptions** as shown in the given code snippet.

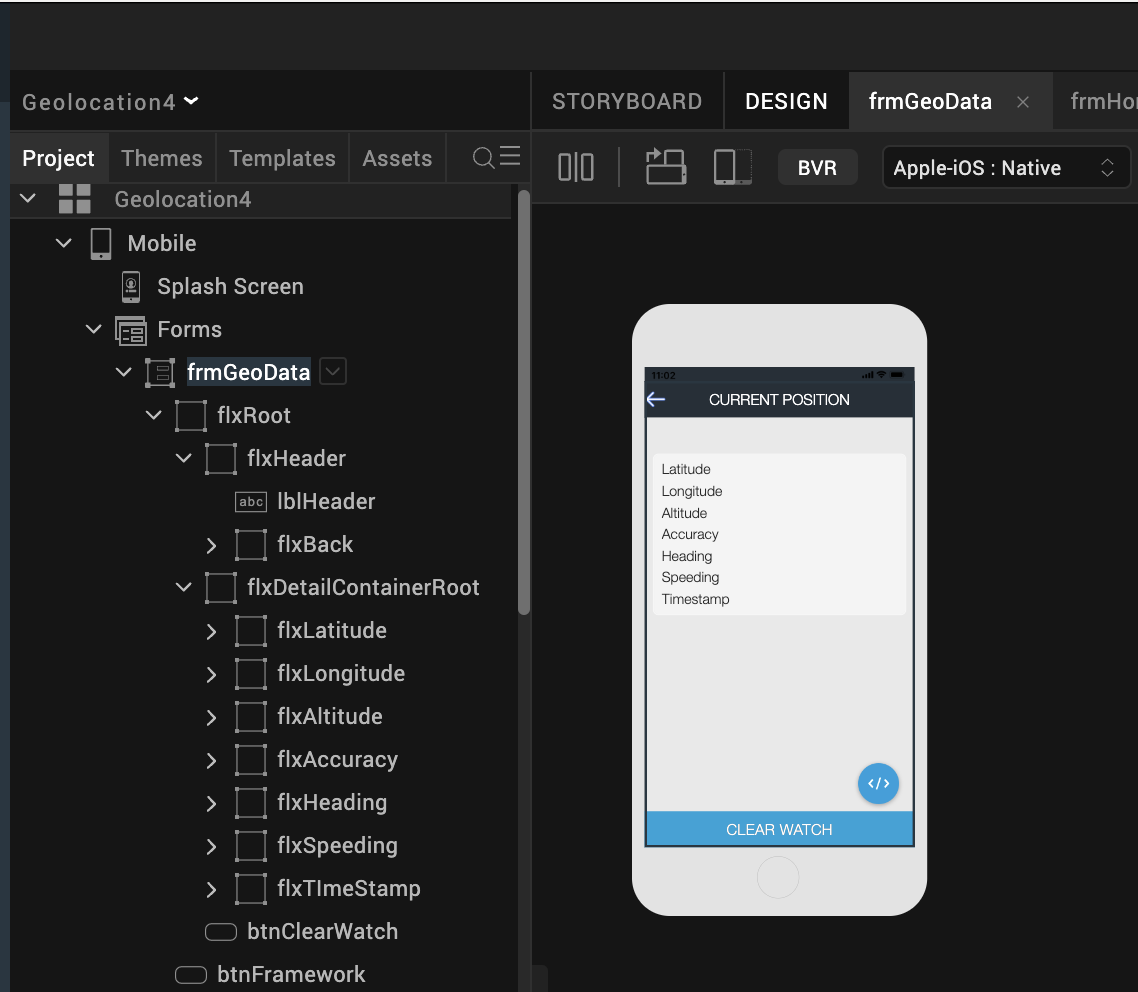
onNavigate: function()

{

this.navObj = new voltmx.mvc.Navigation ("frmGeoData");

}

**frmGeoData** is a form that shows the usage of the Geolocation APIs. Expand **frmGeoData** to see the detailed structure of the form, as shown in the given image.

****

* **flxRoot** is a flex container in a **Flow Vertical** layout. This holds all other widgets on this form.
* **flxHeader** is a flex container that holds a label widget, **lblHeader**, to show the title of the form. It also holds a flex container, **flxBack**, to navigate to the previous form.
  + **flxBack** is a flex container that holds an image widget, **imgBack**, that holds the image for the back arrow.
* **btnFramework** is a button that has its **onClick** action set to navigate to **frmKnowledgeFramework**.
* **flxDetailsContainerRoot** is a flex container that holds two label widgets to show key and value pairs of the Geolocation response.  
  This includes **latitude**, **longitude**, **Altitude**, **Accuracy**, **Heading**, **Speeding** and **Timestamp**.
* The code snippet below gets invoked when the application navigates to the form **frmGeoData** to achieve the current position and watch position functionality.

onNavigate:function(param)

{

if(param==="GEO\_POSITION")

{

this.view.lblHeader.text="CURRENT POSITION";

this.view.btnClearWatch.setVisibility(false);

this.getPosition();

}

else if(param==="WATCH\_POSITION")

{

this.view.lblHeader.text="WATCH POSITION";

this.view.btnClearWatch.setVisibility(true);

this.watchPosition();

}

},

geoSuccessCallback:function(result)

{

try

{

//read latitude, longitude, altitude, accuracy, heading from result object

var lat =result.coords.latitude;

var lang=result.coords.longitude;

var alt=result.coords.altitude;

var acry=result.coords.accuracy;

var hedin= result.coords.heading;

var speed;

//set latitude, longitude, altitude, accuracy, heading values to corresponding labels to //show in UI.

if(lat!==null)

this.view.lblLatValue.text=": "+lat;

else

this.view.lblLatValue.text=": unavailable";

if(lang!==null)

this.view.lblLonValue.text=": "+lang;

else

this.view.lblLonValue.text=": unavailable";

if(alt!==null)

this.view.lblAltValue.text=": "+alt;

else

this.view.lblAltValue.text=": unavailable";

if(alt!==null)

this.view.lblAccValue.text=": "+acry;

else

this.view.lblAccValue.text=": unavailable";

if(hedin!==null)

this.view.lblHeadValue.text=": "+hedin;

else

this.view.lblHeadValue.text=": unavailable";

if(result.timestamp!==null)

this.view.lblTimeValue.text=": "+result.timestamp;

else

this.view.lblTimeValue.text=": unavailable";

// read speed from result object and set to label to show in UI.

speed=result.coords.speeding;

if(speed!==null&&speed!==undefined)

this.view.lblSpeedValue.text=": "+speed;

else

this.view.lblSpeedValue.text=": unavailable";

}

catch(excp

{

// alert(excp.message);

}

voltmx.application.dismissLoadingScreen();

//alert(JSON.stringify(result));

},

geoFailureCallback:function(result)

{

voltmx.application.dismissLoadingScreen();

//alert(JSON.stringify(result));

},

getPosition:function()

{

var self=this;

voltmx.application.showLoadingScreen("sknFrmLoading","please wait...",constants.LOADING\_SCREEN\_POSITION\_ONLY\_CENTER, false,true,null);

geoPosition(self.geoSuccessCallback.bind(this),self.geoFailureCallback.bind(this));

},

watchPosition:function()

{

var self=this;

voltmx.application.showLoadingScreen("sknFrmLoading","please wait...",constants.LOADING\_SCREEN\_POSITION\_ONLY\_CENTER, false,true,null);

this.watchID=watchPosition(self.geoSuccessCallback.bind(this),self.geoFailureCallback.bind(this));

}

The **geoPosition** and **watchPosition** functions are available in the module **geo\_module.js** under the common modules folder. The code in the **geo\_module.js** file is shown in the given snippet.

function geoPosition(successCallback,failureCallback)

{

//var positionoptions1={};

var positionoptions={};

positionoptions.enableHighAccuracy=true;

positionoptions.timeout=10000;

positionoptions.maximumAge=1000;

//watchFlag = false;

try

{

//geolocation API to get current position

voltmx.location.getCurrentPosition(successCallback, failureCallback,positionoptions);

}

catch(exception)

{

alert("Exception is ::"+exception.message);

}

}

function watchPosition(successCallback,failureCallback)

{

//var positionoptions1 = {};//maximumage: 3000;

var positionoptions={};

positionoptions.enableHighAccuracy=true;

positionoptions.timeout=20000;

positionoptions.maximumAge=1000;

positionoptions.minimumTime=2000;

positionoptions.minimumDistance=2;

//watchFlag = true;

var watchID=null;

try

{

watchID = voltmx.location.watchPosition(successCallback,failureCallback, positionoptions);

}

catch(err)

{

alert("Exception is ::"+err.message);

}

return watchID;

}

1. **REVISION HISTORY**

Version 1.0.1

1. **KNOWN ISSUES**

* “voltmx.phone.openEmail” api is Available on all platforms except Server-Side Mobile Web, SPA, and Desktop Web.