Common UI controls

Overview

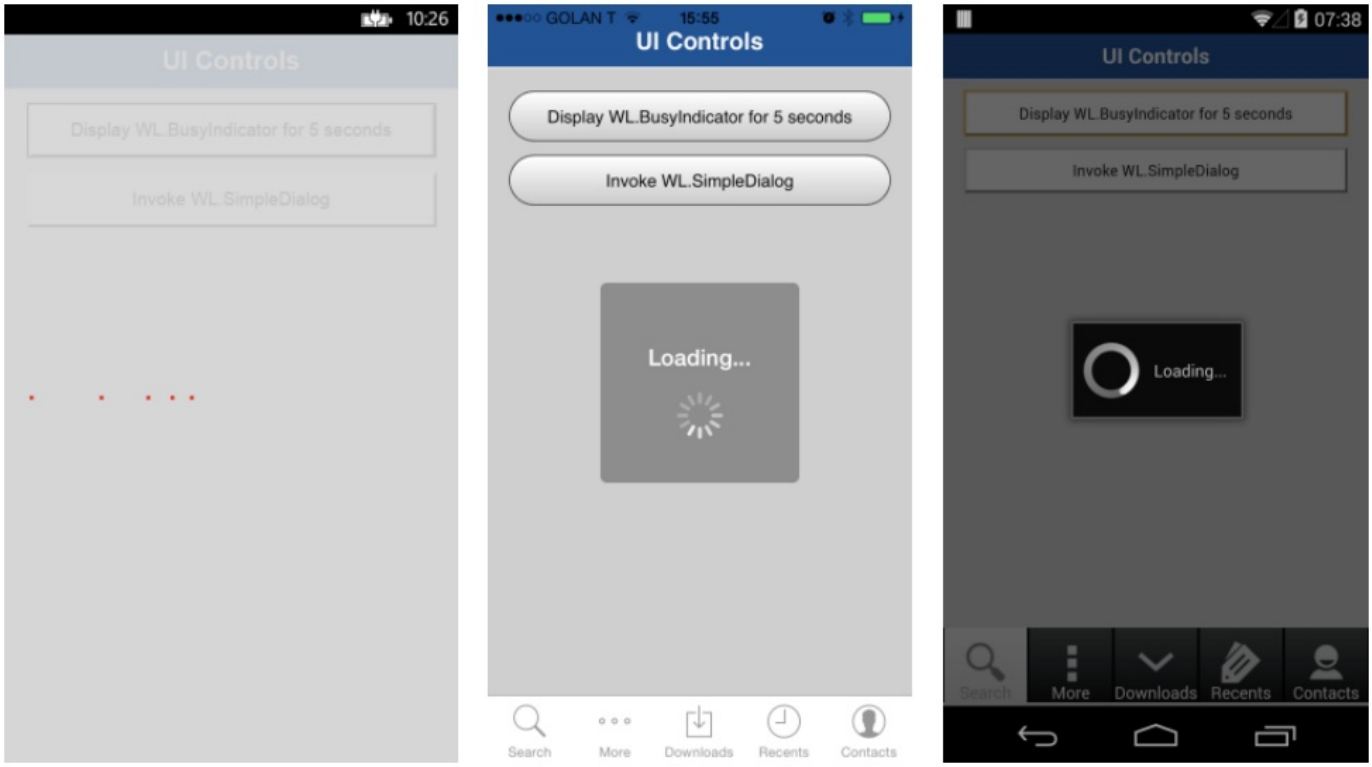
Some controls are common to most hybrid environments, such as modal pop-up windows, loading screens, and tab bars.

With IBM MobileFirst Platform Foundation, you can use a JavaScript API to invoke these controls regardless of the environment. This API automatically renders these
controls in a native way for each mobile platform.

- WL.BusyIndicator
- WL.SimpleDialog
- WL.TabBar
- WL.OptionsMenu
- Splash Screen
- Sample application

WL.BusyIndicator

`WL.BusyIndicator` implements a common API to display a modal activity indicator.
It uses native implementation on the following platforms: Android, iOS, BlackBerry 10, and Windows Phone 8.



It must be initialized before use.

```
busyIndicator = new WL.BusyIndicator( null, {text : 'Loading...'});
```

The first parameter, the parent element ID for `WL.BusyIndicator`, is ignored in iOS, Android, Windows Phone, and BlackBerry environments. It only applies to the web environment.
In the second parameter, available options are:

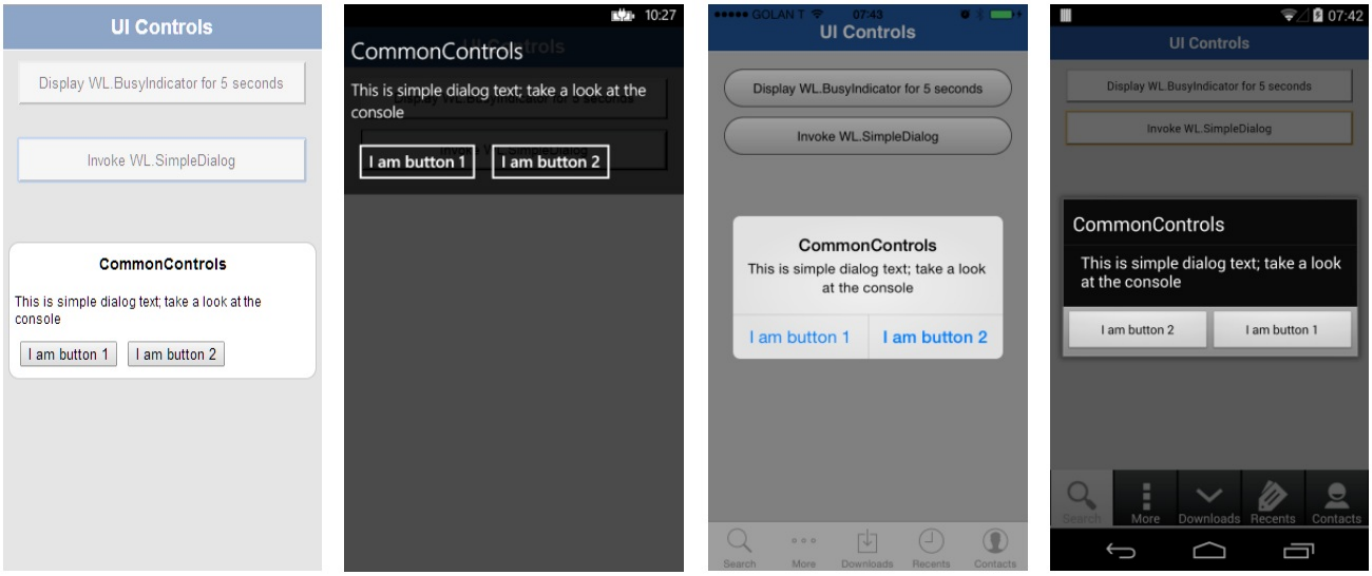
- **text** – set the modal text.
- **color** – set the text color.
- **fullScreen** – should modal message be displayed full screen (iOS only).

For more information about the options, see the MobileFirst user documentation.
`WL.BusyIndicator` provides the following API:

- `void myBusyIndicator.show()` – displays busy indicator.
- `void myBusyIndicator.hide()` – hides busy indicator.
- `boolean myBusyIndicator.isVisible()` – returns whether the busy indicator is visible.

WL.SimpleDialog

The `WL.SimpleDialog` implements a common API for showing a modal dialog window with buttons.
It uses a native implementation on the following platforms: Android, iOS, Windows Phone 8, and BlackBerry 10.
Adobe Air, BlackBerry 6/7, Desktop webpage, and Mobile Web use a JS-based implementation.



The invocation syntax is:

```
WL.SimpleDialog.show(title, text, buttons, options);
```

Parameters are `title`, `text`, and `buttons` as an array of button objects.
The dialog is closed when any of the buttons is pressed.

Each button object has two properties:

- **text** – the text that displayed on the button.

- **handler** – the function to invoke if the button is pressed.

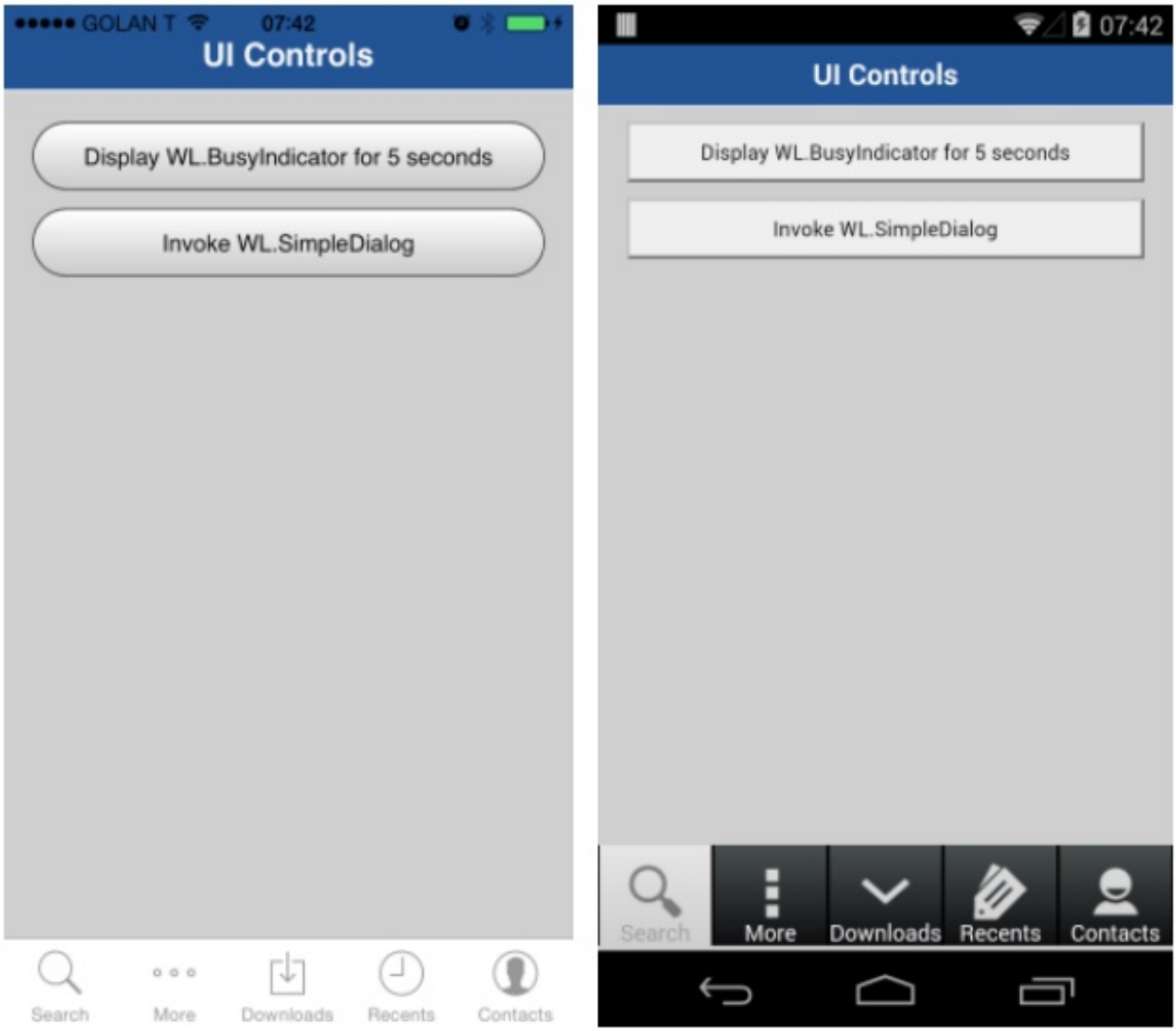
```
var dialogTitle = "CommonControls";
var dialogText = "This is simple dialog text; take a look at the console";
WL.SimpleDialog.show(dialogTitle, dialogText, [
    {
        text : 'I am button 1',
        handler : simpleDialogButton1Click
    }, {
        text : 'I am button 2',
        handler : simpleDialogButton2Click
    }
]);
```

Limitations

- In **Windows Phone 8**, you can use at most four buttons in each instance of `WL.SimpleDialog`
- In **Android**, you can use at most three buttons in each instance of `WL.SimpleDialog`
- Only 1 SimpleDialog can be open at a time

WL.TabBar

`Wl.TabBar` provides application navigation with a tab bar component.
Supported environments are Android and iOS.



The iOS implementation uses a native component, but Android uses an HTML-generated tab bar. The syntax is similar, though with some minor differences.
`WL.TabBar` must be initialized before it can be used.

Because `WL.TabBar` is only available for Android and iOS, it is recommended to initialize it at the environment level (`android\js\main.js` and `iphone-or-ipad\js\main.js`) and not in the common folder (`common\js\main.js`).

Use the following syntax to add a tab bar item:

```
WL.TabBar.addItem(id, callback, title, options);
```

- **itemID** – Internal reference for this tab.
- **callback** – JavaScript function to run when a tab item is pressed.
- **title** – The text to display on the tab bar item.
- **options** – Varies between iOS and Android. See below.

iOS Options

- **badge** – string to display on the badge of the item.
- **image** – file name of an image to use or native iOS button identifier:

- tabButton:More
- tabButton:Favorites
- tabButton:Featured
- tabButton:TopRated
- tabButton:Recents
- tabButton:Contacts
- tabButton:History
- tabButton:Bookmarks
- tabButton:Search
- tabButton:Downloads
- tabButton:MostRecent
- tabButton:MostViewed

```
WL.TabBar.addItem("item1",
   function(){ alert("item 1 pressed"); }
,
   "Item 1",{
      image: "tabButton:Search",
      //image: "images/tabImage.png",}
);
```

Android Options

- image – file name of an image to use for an unselected state.
- imageSelected – file name of an image to use for a selected state.

```
WL.TabBar.addItem("item1",
   function(){ alert("item 1 pressed"); }
,
   "Item 1",{
   image: "images/tabImage.png",}
);
```

Other API Signatures

- `WL.TabBar.init()`
- `WL.TabBar.addItem (returns WL.TabBarItem)`
- `WL.TabBar.removeAllItems (iOS only)`
- `WL.TabBar.setParentDivId (Android only)`
- `WL.TabBar.setVisible(true/false)`
- `WL.TabBar.setSelectedItem(itemID)`
- `WL.TabBar.setEnabled (true/false)`
- `WL.TabBarItem.setEnabled(true/false)`
- `WL.TabBarItem.updateBadge(string) (iOS only)`

WL.OptionsMenu

Supported environments: Android 2.x, Windows 8, and Windows Phone 8.
`WL.OptionsMenu` allows to display a menu of options.
In Windows Phone 8 this also functions as a tab bar.
Note: If your application targets Android 3.0 (API level 11) or higher, WL.OptionsMenu might have no effect, depending on the device. For more information, see the MobileFirst user documentation.

Because WL.OptionsMenu is only available for Android 2.x, Windows 8 and Windows Phone 8, it is recommended to initialize it at the environment level (`android\js\main.js`, `windows8\js\main.js`, `windowsphone8\js\main.js`) and not in the common folder (`common\js\main.js`).

WL.OptionsMenu must be initialized before use.
Here are the API signatures:

- `WL.OptionsMenu.init()`
- `WL.OptionsMenu.addItem` (return reference to a new options item)
- `WL.OptionsMenu.getItem(itemID)`
- `WL.OptionsMenu.getItem(itemID).setEnabled (true / false)`
- `WL.OptionsMenu.setVisible (true / false)`
- `WL.OptionsMenu.setEnabled (true / false)`
- `WL.OptionsMenu.removeItem (itemID)`

- `WL.OptionsMenu.removeItems()`

Use the following syntax to add an option of a menu:

```
WL.OptionsMenu.addItem(id, callbackFunction, title, options);
```

<

- **itemID** – Internal reference for this menu option
- **callback** – JavaScript function to run when the menu option is pressed.
- **title** – The text of the menu item.
- **options** – An options object with the following properties:
    - **image** – A path to a designated image, relative to resource root directory.
    - **enabled** – Boolean stating if the item is enabled or disabled.

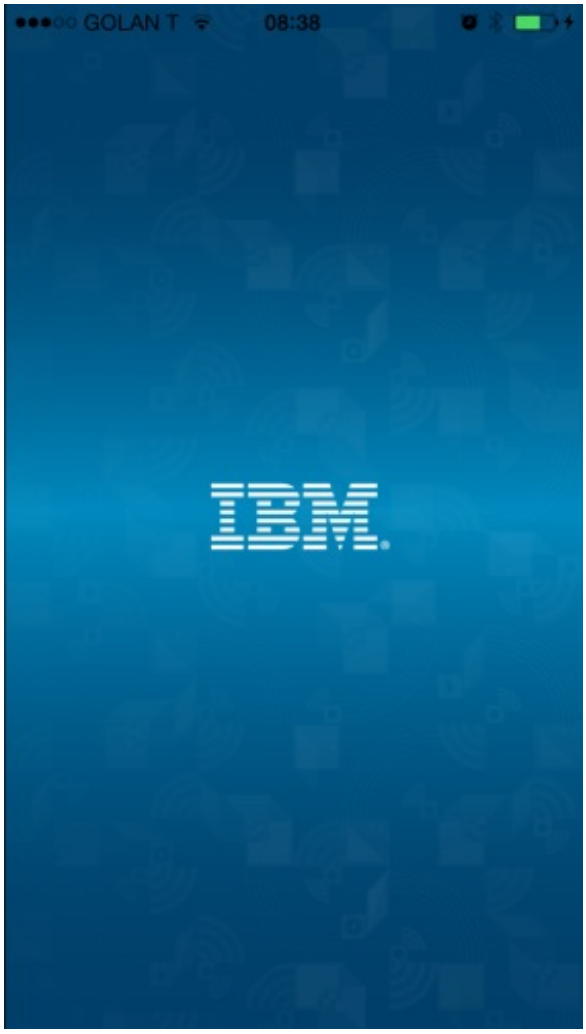
```
WL.OptionsMenu.addItem("item2",
   function(){ alert("item 2 pressed");}
,
   "Contacts", {
   image: "contacts.png"}
);
```

Paths to image files must not be given; instead, place the files at the following locations:

- **Android**: nativeResouces\drawable-*
- **Windows 8**: Resources\applicationBar
- **Windows Phone 8**: nativeResources\applicationBar

Splash Screen

Supported environments: Android, iOS and WP8.



The framework provides a default splash screen loading behavior.
The splash screen is shown once the application launches.
This is done in

- **android\native\src\com\app-name\app-name.java**
- **iphone-or-ipad\native\Classes\app-name.m**
- **Windowsphone8\native\App.xaml.cs**

The splash screen is then hidden once the framework finishes initializing
The following explains how to extend the default behavior or create a new one altogether.
For more information, see the user documentation topic: “Managing the splash screen with JavaScript APIs”.

As mentioned, you can handle when the splash screen to be hidden by using MobileFirst framework. To do so:
Uncomment the **autoHideSplash** option the **initOptions.js** file.
Use the following API method in at the point in the JavaScript code where you want the splash screen to be hidden. For example:

```
function wlCommonInit() {
    WL.App.hideSplashScreen()
;
}
```

If an application requires extra processing time while it launches, the splash screen duration can, for example, be extended by implementing custom JavaScript code that does so.
This might happen when waiting for data from a backend, or while loading of more frameworks.

```
function wlCommonInit(){
   // Custom app logic...
      customLogicCallback();
}
function customLogicCallback() {
   WL.App.hideSplashScreen();
}
```

Similarly, the splash screen can be manually displayed again by using the JavaScript API method:

```
WL.App.showSplashScreen();
```

In this example we wish to force a reload of the application.

```
function wlCommonInit() {
   WL.App.hideSplashscreen();
   // Custom app logic…
   reloadApplication();
}
function reloadApplication() {
   WL.App.showSplashScreen()
;
   // More custom app code…
   WL.Client.reloadApp();
}
```

By default, the splash screen that is used in a MobileFirst application is a static image.
To use a different image, replace the following with another image:

- **Android**: **native\res\drawable\splash.9.png**
- **iOS**: **native\Resources\Default-*.png**
- **WP8**: **native\SplashScreenImage.png**

The splash screen can also be more than a static image.
A developer can implement custom code that will, extending the time the splash screen is displayed, display an animated “loading…” screen, display a short video clip...

> For more information about creating a custom splash screen, see the user documentation topic “Managing the splash screen”.

Sample application

Click to download the sample application for this tutorial.
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/CommonUIControlsProject.zip) Studio project.