Windows Phone 8 - Adding native UI elements to hybrid applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.1/adding-native-functionality/windows-phone-8-adding-native-ui-elements-hybrid-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

You can write hybrid applications by using solely web technologies. However, IBM MobileFirst Platform Foundation also allows you to mix and match native code with web code as necessary.

For example, use native UI controls, use native elements, provide an animated native introduction screen, etc. To do so, you must take control of part of the application startup flow.

Prerequisite: This tutorial assumes working knowledge of Windows Phone development.

This tutorial covers the following topics:

- Sending commands from JavaScript code to native code
- Sending commands from native code to JavaScript code
- The SendAction sample
- Shared session
- Sample application

Sending commands from JavaScript code to native code

In MobileFirst applications, commands are sent with parameters from the web view (via JavaScript) to a native class (written in C#).
You can use this feature to trigger native code to be run in the background, to update the native UI, to use native-only features, etc.

Step 1

In JavaScript, the following API is used:
`WL.App.sendActionToNative("doSomething", {customData: 12345});`

The `doSomething` parameter is an arbitrary action name to be used in the native side (see the next step), and the second parameter is a JSON object that contains any data.

Step 2

The native class to receive the action must implement the `WLActionReceiver` protocol:
`public class action : WLActionReceiver`

The `WLActionReceiver` protocol requires an `onActionReceived` method in which the action name can be checked for and perform any native code that the action needs:

```
public void onActionReceived(string action, JObject data) {
    if (action == "displayAddress") {
        Deployment.Current.Dispatcher.BeginInvoke(() => {
            // perform required actions
        });
    }
}
```

Step 3

For the action receiver to receive actions from the MobileFirst Web View, it must be registered.
The registration can be done during the startup flow of the application to catch any actions early enough:

```
InitializeComponent();
WL.createInstance(); //Create the instance of the ActionSender API'
s
myReceiver = new Action();
Loaded += PhoneAppPage_Loaded;
WL.getInstance().addActionReceiver(myReceiver);
```

You can later remove this `ActionReceiver` registration by using the following API:

```
WL.getInstance().removeActionReceiver(myReceiver);
```

Sending commands from native code to JavaScript code

In MobileFirst applications, commands can be sent with parameters from native C# code to web view JavaScript code.
You can use this feature to receive responses from a native method, notify the web view when background code finished running, have a native UI control the content of the web view, etc.

Step 1

In C#, the following API is used:

```
WL.getInstance().sendActionToJS(doSomething, data);
```

The `doSomething` parameter is an arbitrary action name to be used on the JavaScript side and the second parameter is a `JObject` that contains any data.

Step 2

A JavaScript function, which verifies the action name and implements any JavaScript code.

```
function actionReceiver(received) {
   if (received.action == "doSomething" && received.data.someProperty == "12345")
{
      //perform required actions, e.g., update web user interface
   }
}
```

Step 3

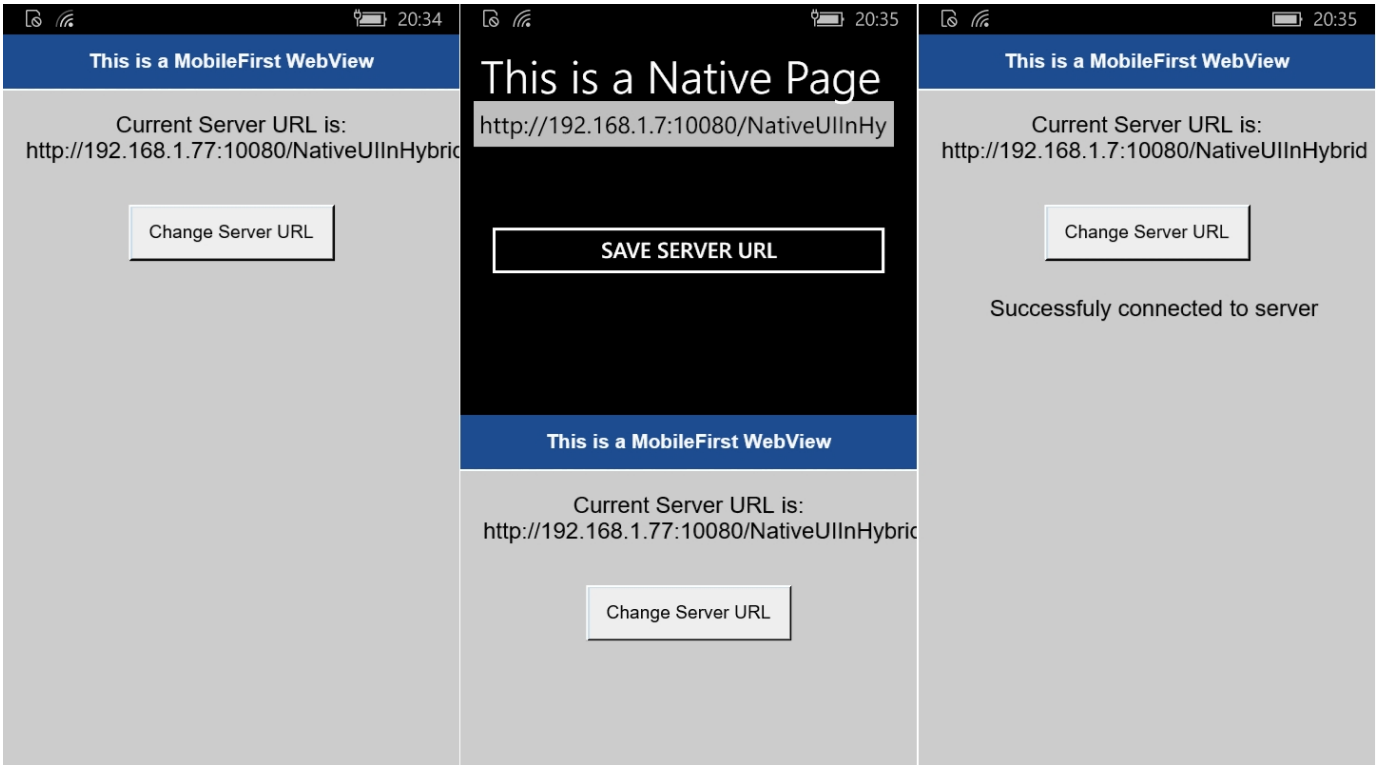
For the action receiver to receive actions, it must first be registered. This should be done early enough in the JavaScript code so that the function can handle those actions as early as possible.

```
WL.App.addActionReceiver ("MyActionReceiverId", actionReceiver);
```

The first parameter is an arbitrary name. It can be used later to remove an action receiver.

```
WL.App.removeActionReceiver("MyActionReceiverId");
```

The SendAction Sample



1. Download the NativeUIInHybrid project, which includes a hybrid application called SendAction.
2. In the first screen, you will see the current server URL to which the application will try connect. You will also see a **Change ServerURL** button.
   This action sends this action to the native C# code and it displays a `TextBox` object where you can

modify the server URL to which the application has to connect. When the **SAVE SERVER URL** button in the Native Page is pressed, the native `SendAction` method is called and sends a message back to the JavaScript code. The `actionReceiver` method in JavaScript will then reload the application and try to connect to the new server address. A successful connection is notified in the hybrid page as shown above.

HTML

The HTML page shows the following elements:

- The current server URL to which it will connect
- A button to trigger a modify to the URL in Native Page

```
<p>This is a MobileFirst WebView.</p>
<div id="currentServerURLDiv"></div>
<input class="formButton" id="changeServerURL" type="button" value="Change Server URL">
<div id="ConnectionStatusDiv"></div>
```

JavaScript

When the button is clicked, the `sendActionToNative` method is called to send the address to the native code.

```
$('#changeServerURL').on('click', function() {
    WL.App.sendActionToNative("displayNativeScreen", { })
;
});
```

The code also registers an action receiver to display potential error messages from the native code.

```
WL.App.addActionReceiver ("BackFromNative", function actionReceiver(received) {
    if(received.action == 'refreshView') {
      WL.Client.reloadApp();
    }
});
```

Shared session

When you use both JavaScript and native code in the same application, you might need to make HTTP requests to MobileFirst Server (connection, procedure invocation, etc.)

HTTP requests are explained in other tutorials about authentication, application authenticity, and HTTP adapters (both for hybrid and native applications).
IBM Worklight Foundation 6.2, and IBM MobileFirst Platform Foundation 6.3 and later, keep your session (cookies and HTTP headers) automatically synchronized between the JavaScript client and the native client.

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/NativeUIInHybrid/tree/release71) the MobileFirst project.