Platform - Worklight

Version: 2013.11.14

Topics

External Documents

Dealing with New Device O/S Versions

Examples:

- Soft keyboard affects `viewport.height` when shown/hidden dynamically in iOS7, but did not in iOS6.
    - This type of change is done by the browser implementation, and controlled via `<meta>` viewport tag
    - See this Stackoverflow article
- It is possible that browser (even native) screen dimensions available to apps might change between major OS versions.
    - The proper way to deal with this is CSS media queries, just as if it's a new type of device with different screen dimensions (even though in this case its caused by the OS/software updates)
    - Specific example is the addl 20px height given to all apps in iOS7 (native/hybrid/web)
    - One possible general solution if you have a header `<div>` in your app is to add `padding-top: 20px` within a media query that matches the iOS7 screen heights (one for portrait and one for landscape) vs. the default that doesn't add the padding
    - Or vice-versa, make the new default styling with the extra padding going forward, and remove the extra top padding in a media query matching the older screen dimensions

Simple and Clean Previews of Worklight Apps

For most day to day developer testing, having the full Worklight simulator environment can be obstructive. It takes longer to load, makes it harder to examine the DOM, and generally is not helpful. Granted it is useful and needed when testing device features, such as the camera. But the 80% of the time, developers only need a simple browser based experience for debugging.

Use the following URL pattern to load your Worklight app in preview mode without the simulator accouterments.

```
http://{wlserver}:{wlport}/{wlproject}/apps/services/preview/{wlapp}/
{wlenv}/{wlappver}/default/index.html
```

Where:

- **{wlserver}:{wlport} -** The host and port of your embedded Worklight server. Typically, this will be **localhost:10080**
- **{wlproject}** : The Worklight project name being tested
- **{wlapp}** : The Worklight app being tested
- **{wlenv}** : The environment being tested. Possible values are: android, iphone, ipad,
- **{wlappver}** : The Worklight version being tested. Unless overridden, this will be **1.01.0**

Example URL:

```
http://localhost:10080/MyProject/apps/services/preview/MyApp/android/
1.01.0/default/index.html
```

Source Code ignore list

The following list of directories and file should be added to your source code control’s "ignore" lists:

```
/apps/{AppName}/android/native/assets/www
/apps/{AppName}/android/native/bin
/apps/{AppName}/android/native/gen

/apps/{AppName}/blackberry/native/www
/apps/{AppName}/blackberry10/native/build
/apps/{AppName}/blackberry10/native/www

/apps/{AppName}/ipad/native/build
/apps/{AppName}/ipad/native/cordova
/apps/{AppName}/ipad/native/cordovaLib
/apps/{AppName}/ipad/native/WorklightSDK
/apps/{AppName}/ipad/native/www
/apps/{AppName}/ipad/package

/apps/{AppName}/iphone/native/build
/apps/{AppName}/iphone/native/cordova
/apps/{AppName}/iphone/native/cordovaLib
/apps/{AppName}/iphone/native/WorklightSDK
/apps/{AppName}/iphone/native/www
/apps/{AppName}/iphone/package

/apps/{AppName}/windowsphone*/native/Bin
/apps/{AppName}/windowsphone*/native/obj
/apps/{AppName}/windowsphone*/native/www

/bin
```

Native folder management

The other issue with source code control and Worklight is the "native" folders. Almost 100% of it can (and probably should) be ignored. There are many irrelevant files that get changed on each build that are then marked dirty for SCCS and get disbursed to the team. Worse yet, if multiple developers do a build, then we show file conflicts. Another common issue when importing shared projects, is that things don't work unless you delete the "native" folder, thus losing any custom settings. While this contradicts the InfoCenter, we recommend that you ignore the entire “native” directory.

If you need to alter any files located under an environment's "native" folder -- such as editing the androidManifest.xml, plist files, or adding cordova plugins -- copy that file into the "nativeResources" folder first, and edit that file instead. Any files in this folder are copied to the native folder prior to build. A side benefit is that it is obvious which native files have been customized. As long as you are careful about having all native tree changes made in the nativeResources folder instead, and making sure the nativeResources folder is properly managed under SCCS, then you get the best of both worlds.

Exporting projects

When exporting a project to a zip archive, DO NOT include the following files and dirs:

```
/bin
/jslib
/apps/{AppName}/android/bin/{APP}.apk
/apps/{AppName}/android/native

/apps/{AppName}/blackberry/native
/apps/{AppName}/blackberry10/native

/apps/{AppName}/ipad/native
/apps/{AppName}/ipad/package

/apps/{AppName}/iphone/native
/apps/{AppName}/iphone/package

/apps/{AppName}/windowsphone8/native
```

: Any customizations made in the native folder of the respective environments should be reflected in the associated "nativeResources" directory, as described in the previous tip.

For Dojo-enabled projects that you are sharing, be sure to also export the associated Dojo Library project (ie dojoLib). This is needed to provide any loose modules that have not been added directly to the main app project's www directory.

While there is no current tooling to support this, 3rd party libraries should not be generally exported, but rather they should be loaded once the project has been imported into Worklight Studio. These libraries would typically come from bower, npm, brew, or some other package management system.

Worklight Image Sizes

Note: May need updating for IOS7

Worklight Image Sizes	**w x h**
common/images/icon.png common/images/thumbnail.png common/images/favicon.ico	128 x 128 90 x 90 16 x 16
Android Images	**w x h**
android/native/res/drawable/icon.png android/native/res/drawable/push.png	48 x 48 25 x 25
android/native/res/drawable-hdpi/icon.png android/native/res/drawable-hdpi/push.png android/native/res/drawable-hdpi/settings.png	72 x 72 24 x 38 72 x 72
android/native/res/drawable-hdpi-v11/push.png	36 x 36
android/native/res/drawable-ldpi/icon.png android/native/res/drawable-ldpi/push.png android/native/res/drawable-ldpi/settings.png	36 x 36 12 x 19 36 x 36
android/native/res/drawable-ldpi-v11/push.png	18 x 18
android/native/res/drawable-mdpi/icon.png android/native/res/drawable-mdpi/push.png android/native/res/drawable-mdpi/settings.png	48 x 48 16 x 25 48 x 48
android/native/res/drawable-mdpi-v11/push.png	24 x 24
android/native/res/drawable-xhdpi/icon.png android/native/res/drawable-xhdpi-v11/push.png	96 x 96 48 x 48
IOS (Applies for both ipad and iphone trees)	**w x h**
ipad/Resources/Default-iphone.png (Splash) ipad/Resources/Default@2x-iphone.png (Splash, retina)	320 x 480 640 x 920
ipad/Resources/Default-Landscape-ipad.png ipad/Resources/Default-Landscape@2x-ipad.png ipad/Resources/Default-Portrait-ipad.png ipad/Resources/Default-Portrait@2x-ipad.png	1024 x 768 2048 x 1496 768 x 1024 1536 x 2008

ipad/Resources/Icon.png ipad/Resources/Icon@2x.png ipad/Resources/Icon-72.png ipad/Resources/Icon-72@2x.png (@ 72dpi) ipad/Resources/Icon-small.png ipad/Resources/Icon-small-50.png	57 x 57 114 x 114 72 x 72 144 x 144 29 x 29 50 x 50
ipad/Resources/ITunesArtwork.png	512 x 512
Blackberry	**w x h**
blackberry/native/icon.png blackberry/native/splash.png	80 x 80 200 x 38
Windows	**w x h**

App Versioning and Publishing Worklight Apps

In Worklight, the “application-descriptor.xml” file contains the apps “public” version.

For IOS deployments, an **.ipa** has two versions embedded into it.

- **Public Version** - This is the version that is shown to users on the AppStore, and is visible on the device. This is the version number that is maintained within the Worklight app-desc file. This version is localizable as needed.

  This is also known as the “Bundle Short Version String”. Within the info.plist file of the app this version is stored in the CFBundleShortVersionString.

- **Build Version** - This is the private iTunes App Store only version. It is used to compare app versions during AppStore submission, and must be higher than any previously submitted versions. We believe this is an options version number, until it has been used once, at which point it becomes mandatory (for version comparison). Worklight does not touch this property, it is the customer's responsibility to ensure that it is valid and consistent when used.

  This version is also known as the “Bundle Version”. Within the info.plist file of the app this version is stored in the CFBundleVersion.

Best Practice for complex CSS applications

This entry describes how to properly utilize CSS stylesheets within Worklight. I'm currently putting my env specific CSS files into each environment's css/ directory as part of my initial dojo build. Then in each env's MyApp.css file, I do an @import of the specific file.

Worklight Limitation

Worklight concatenates CSS files of the common and environment specific trees during the build process. This presents an issue with regards to `@import` in CSS files within Worklight applications. To illustrate, here is an example use case:

In **common/css/MyApp.css**, we have the following:

```
@import("iphone.css");
h1 { ... }
.myRule { ... }
.another { ... }
```

Then in **android/css/MyApp.css**, we have:

```
@import("android.css");
.myAndroidRule { ... }
```

When WL builds the app, it concatenates these files, which breaks the rule of `@import` statements needing to be at the top of the CSS file, and the android CSS file is never read in!
To workaround this issue, there are two options:

1. Do not use `@imports` in Worklight projects. This is simply not realistic in ANY non-trivial application.
2. If `@import`s are used, Do not have **any** discreet rules in the "MyApp.css" files.

Solution pattern

So given that option 2 above is the only practical solution, the css files would now resemble the layout shown below. This assumes that you have multiple environments. There are two basic strategies to follow regarding CSS. You can either start with a common theme that carries over all environments, or you have a unique theme for each environment. Both are valid depending on your needs. The following layout can safely support either configuration. Shown below is an example suggested layout of a complex CSS structure.

/common/css/{MyApp}.css

```
/*
-------------------------------------------------------------
This file may contain only @import statements, or only
rules. If rules are placed here, then the environment
specific MyApp.css may not contain @imports!
-------------------------------------------------------------
Global base css when using common look and feel
Usually taken from base env file (ie iphone.css)
-------------------------------------------------------------
*/

@import url("common.css");

/*
-------------------------------------------------------------
Other non-conflicting external css files, typically
library files
-------------------------------------------------------------
*/
@import url("someLibrary.css");
...
```

/common/css/common.css

```
/*
-------------------------------------------------------------
This file is used to define common (global) css rules.
It is used when the application will have a common look
and feel shared by all environments. It is typically based
on a dojox.mobile theme file (ie iphone.css).
-------------------------------------------------------------
*/
```

/common/css/common_custom.css

```
/*
-------------------------------------------------------------
This file is used to override common (global) css rules.
It should be loaded after the common.css, library, and
environment specific rules are applied.
-------------------------------------------------------------
*/
```

/{environment}/css/{MyApp}.css

```
/*
-------------------------------------------------------------
This environment specific file may contain @import and/or
rules. It is recommended that only @imports be used here
and rules be placed in the {environment}_custom.css file.
-------------------------------------------------------------
Environment specific base rules, when there IS NOT a
common look and feel
-------------------------------------------------------------
*/

@import url("iphone.css");

/*
-------------------------------------------------------------
Other non-conflicting external css files, specific to
this environment
-------------------------------------------------------------
*/
@import url("someLibrary.css");
...

/*
-------------------------------------------------------------
Bring in global common override rules for either the
global and/or environment rules
-------------------------------------------------------------
*/
@import url("common_custom.css");

/*
-------------------------------------------------------------
Bring in environment specific override rules AFTER the
common rules are applied.
-------------------------------------------------------------
*/
@import url("iphone_custom.css");
```

/{environment}/css/{environment}.css

```
/*
-------------------------------------------------------------
This file is used to define an environment specific theme.
Typically, you will define either a common (global) theme,
or environment specific themes.
-------------------------------------------------------------
*/
```

/{environment}/css/{environment}_custom.css

```
/*
-------------------------------------------------------------
This file is used to override environment specific theme
rules. It should be loaded last after: common, environment,
and common_custom.
-------------------------------------------------------------
*/
```

When combined, we have a valid master CSS file that contains all @imports. When a final dojo build is run, it will combine the css files for each env, and result in a single css file that contains all rules. So again, when WL combines the post built files, its still valid in that sense as well.