Orientation

Objective

In this section, you will learn how you can handle device orientation. We'll take a look at your options for setting the UI orientation. Then, we'll show you how you can react to orientation changes within your app.

Contents

You have a few options for handling device orientation:

- Lock orientation for the whole app
- Lock the orientation for a given window
- React to orientation changes.

Before we begin, it's important to cover some caveats and subtleties on Android. With that platform, it's important that you keep in mind that the orientation values you *set* don't match those you *get*. With Android, you can set the UI orientation to any of four possibilities: portrait upright, landscape right, portrait upside-down, and landscape left. But, when you request the current orientation, you'll get one of two values: portrait or landscape. This is a *platform* feature, not a Titanium implementation issue.

A further consideration is that portrait and landscape vary between phones and tablets. A phone is in portrait mode when its "top" is at 0 degrees (hardware buttons at the bottom) and landscape when the "top" is at 270 degrees. A tablet is in landscape mode when its top is at 0 and portrait when its top is at 90 degrees. (Based on sensor degrees.) These portrait/landscape values are what you receive when you *get* the devices current orientation.

With those caveats in mind, let's proceed...

Orientation design principles

Apple's Developer documentation says: "People expect to use your app in different orientations, and it’s best when you can fulfill that expectation." In other words, don't look at handling orientation as a bother but an opportunity.

Apple further recommends that when choosing to lock or support orientation, you should consider following these principles:

- On iPhone/iPod Touch – don't mix orientation of windows within a single app; so, either lock orientation for the whole app, or react to orientation changes.
- On iPhone – don't support the portrait-upside-down orientation, because that could leave the user with their phone upside-down when receiving a phone call.
- On iPad – you should support all orientations because that matches how people use those devices.

These same principles apply to an Android app as well.

Orientation is an opportunity

Rather than considering orientation a "necessary evil" to handle, think of it as an opportunity. When a user rotates their device, you could display different content. Consider a recipe app that shows a list of ingredients when in portrait mode but shows cooking directions when the device is in landscape mode. Some handsets mute the speaker when the device is face down. You can probably think of other interesting ways your app could react to an orientation change.

Limiting orientation modes supported by your app

You specify the orientations your app can support by modifying the `tiapp.xml` file. This type of configuration controls the splash screen orientation possibilities. And it constrains which orientations the windows of your apps could possible show in, but not necessarily the orientation

of a specific window.

The techniques for iOS and Android vary, so we'll look at them separately.

Limiting orientation modes on iOS

Specify the orientation modes the application needs to support with the `UISupportedInterfaceOrientations` key in the iOS plist section of the project's `tiapp.xml` file.

By default, Titanium sets iPhone applications to support upright portrait only and iPad application to support all orientation modes.

tiapp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
    <ios>
        <plist>
            <dict>
                <key>UISupportedInterfaceOrientations~iphone</key>
                <array>
                    <string>UIInterfaceOrientationPortrait</string>
                </array>
                <key>UISupportedInterfaceOrientations~ipad</key>
                <array>
                    <string>UIInterfaceOrientationPortrait</string>
                    <string>UIInterfaceOrientationPortraitUpsideDown</string>
                    <string>UIInterfaceOrientationLandscapeLeft</string>
                    <string>UIInterfaceOrientationLandscapeRight</string>
                </array>
            </dict>
        </plist>
    </ios>
</ti:app>
```

Limiting orientation modes on Android

Limiting orientation on Android can also be accomplished via the `tiapp.xml` file, though not in the same way. The primary configuration file for Android apps is the `AndroidManifest.xml` file. At build time, entries in your project's `tiapp.xml` file are used to create the Android Manifest that's packaged with your app. To force orientation support, you'll need to copy some entries from generated Android Manifest file back into `tiapp.xml`, modify them, then build your app again.

1. Build your app in Titanium.
2. Open the `tiapp.xml` file and display its XML contents.
3. Next, you need to adjust the `<android>` node:
    a. From the line that reads `<android xmlns:android="http://schemas.android.com/apk/res/android"/>`, delete the "/" at the end (to change it from an empty tag to an opening tag).
    b. Add a new closing `</android>` tag
    c. Between those tags, add new `<manifest></manifest>` tags.
4. Open `<PROJECT_NAME>/build/android/AndroidManifest.xml` in Studio (or a text editor of your choice).
5. Copy the `<application>` node, which contains all of the `<activity>` nodes from that file, for example:

AndroidManifest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.myapp.app" android:versionCode="1" android:versionName="1.0">
 <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="19"/>

    <!-- Start Copying Here -->

 <application android:icon="@drawabe/appicon" android:label="MyApp"
android:name="MyappApplication" android:debuggable="false"
android:theme="@style/Theme.AppCompat">
  <activity android:name=".MyappActivity" android:label"@string/app_name"
android:theme="@style/Theme.Titanium"
android:configChanges="keyboardHidden|orientation|screenSize">
   <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
   </intent-filter>
  </activity>
  <activity android:name="org.appcelerator.titanium.TiActivity"
android:configChanges="keyboardHidden|orientation|screenSize"/>
  <activity android:name="org.appcelerator.tianium.TiTranslucentActivity"
android:configChanges="keyboardHidden|orientation|screenSize"
android:theme="@style/Theme.AppCompat.Translucent"/>
  <activity android:name="ti.modules.titanium.ui.android.TiPreferencesActivity"
android:configChanges="screenSize"/>
 </application>

    <!-- Stop Copying Here -->

 <uses-permission android:name="android.permission.INTERNET"/>
 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
 <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>
</manifest>
```

6. Paste them between the `<manifest></manifest>` tags you added to the `tiapp.xml` file. From now on, each time your app is built, Titanium will copy these activity tags to the Android Manifest file it generates. You're now ready to specify the UI orientation.
7. For each activity tag, add the `android:screenOrientation` attribute. Set it to the orientation type you want to use. For example, no `sensor` locks the application in the device's preferred orientation mode, which is usually portrait for phones and landscape for tablets. For a full list of orientation types, see http://developer.android.com/guide/topics/manifest/activity-element.html#screen.

The final manifest section of your `tiapp.xml` file should look similar to the example below. If you need to debug the application, set the `application` element's `android:debuggable` attribute to `true`.

tiapp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
    <android xmlns:android="http://schemas.android.com/apk/res/android"/>
     <manifest>
   <application android:icon="@drawable/appicon"
                        android:label="MyApp"
                        android:name="MyappApplication"
                        android:debuggable="false"
                        android:theme="@style/Theme.AppCompat">
    <activity android:screenOrientation="nosensor"
                         android:name=".MyappActivity"
                         android:label="@string/app_name"
                         android:theme="@style/Theme.Titanium"

android:configChanges="keyboardHidden|orientation|screenSize">
     <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
     </intent-filter>
    </activity>
    <activity android:screenOrientation="nosensor"
                         android:name="org.appcelerator.titanium.TiActivity"

android:configChanges="keyboardHidden|orientation|screenSize"/>
    <activity android:screenOrientation="nosensor"

android:name="org.appcelerator.titanium.TiTranslucentActivity"

android:configChanges="keyboardHidden|orientation|screenSize"
                         android:theme="@style/Theme.AppCompat.Translucent"/>
    <activity android:screenOrientation="nosensor"

android:name="ti.modules.titanium.ui.android.TiPreferencesActivity"
                         android:configChanges="screenSize"/>
   </application>
  </manifest>
 </android>
</ti:app>
```

Limiting orientation modes supported by a window

The preceding techniques control the orientation modes supported by your entire app, including all its windows. But what if you want window A to be in portrait while window B is in landscape? You can limit the orientation modes supported by a specific window by setting the window's `orientationModes` property. This property accepts an array of Ti.UI constants that specify the window's permitted orientations. Remember, you must have enabled the various orientations in the tiapp.xml before setting a window to that orientation.

iOS Platform Notes
Using the Window's `orientationModes` property to force the orientation of non-modal windows is considered a bad practice and will not be supported, including forcing the orientation of windows inside a NavigationWindow or TabGroup.

Modal windows should not support orientation modes that the window they are opened over do not support. Doing otherwise **may** cause bad visual/redraw behavior after the modal is dismissed, due to how iOS manages modal transitions. If the `orientationModes` property of a modal window is undefined, then the orientations supported by this window would be the orientation modes specified in the `tiapp.xml`.

```
var win = Ti.UI.createWindow({
 /* on Android, it needs to be a "heavyweight" window */
 fullscreen: false,
 /* This works on iOS */
 orientationModes: [
  Ti.UI.PORTRAIT,
  Ti.UI.UPSIDE_PORTRAIT
 ]
});
// but for Android using Titanium prior to 2.1 you have to set it after creation
win.orientationModes = [Ti.UI.PORTRAIT, Ti.UI.UPSIDE_PORTRAIT]
```

Setting orientation summary

- You want to limit to only portrait or to only landscape – set the desired orientation value in the `tiapp.xml` as described in the "Limiting orientation modes supported by your app" section above. You don't need to set `win.orientationModes`.
- You want to have some windows (or tabs) in one orientation and other windows in another orientation – enable each of the supported orientations in `tiapp.xml`, then specify each window orientation using the `win.orientationModes` property.

Reacting to orientation changes

The most powerful way to handle orientation is for your app to react to changes and update its UI. You'd reposition buttons, images, and so forth when the user turns their device. You detect orientation changes via the `Ti.Gesture` object.

```
Ti.Gesture.addEventListener('orientationchange',function(e) {
 // get current device orientation from
 // Titanium.Gesture.orientation

 // get orientation from event object
 // from e.orientation

 // Ti.Gesture.orientation should match e.orientation
 // but iOS and Android will report different values

 // two helper methods return a Boolean
 // e.source.isPortrait()
 // e.source.isLandscape()
});
```

If you've watched any of Kevin Whinnery's videos or read his Forging Titanium posts, you should be familiar with his recommendation to write "component-oriented apps." In such apps, your UI is divided into functional components that "know" how to update themselves. For example, if you look at the finished code for our Local Data lab, you'll see that the table "knows" how to populate itself.

Following his technique, in the `orientationchange` event handler, you'd fire an app-level event using `Ti.App.fireEvent()`. Within each of your UI components, you would have an app-level listener for that event which would update the component with new layout specifics.

```
Ti.Gesture.addEventListener('orientationchange',function(e) {
 Ti.App.fireEvent('orient', {portrait:e.source.isPortrait()});
});
// ... elsewhere ...
var myCustomView = function() {
 var view = Ti.UI.createView({
  top:10,
  left:10,
  /* etc */
 });
 Ti.App.addEventListener('orient', function(evt) {
  if(evt.portrait===true) {
   view.left = 10;
  } else {
   view.left = 50;
  }
 });
}
```


Don't use orientation event listeners to force orientation support

Using events to limit supported orientations is not recommended. We've see community-contributed code that suggests you add an orientation event listener in your app; when it fires, you'd set the window's orientation to a specific direction. The rationale is that doing so provides a means to specify orientation without modifying the iOS or Android configuration files. We do not recommend this for a few reasons:

- The app's screen could temporarily draw in the unwanted orientation before being forced back to the desired orientation.
- Adding an unnecessary "super-global" event listener opens the possibility of creating a memory leak. See the Managing Memory and Finding Leaks guide for further information.
- And, why use a kludge when you could follow the proper technique to limit the orientation via settings in the configuration files?

Splash screen support for various orientations

Splash screens are shown when your app launches. A default PNG file is provided with a new Titanium project to be used as your app's splash screen. You can change, but not remove entirely the splash screen: it is displayed while your app is launching and is removed when the entry-point window of your app is ready for user interaction.

- Android
  - The filename must be `default.png` with a lowercase **d**. Because this is platform specific, this file will typically be found in your project's `Resources/android` directory.
  - You can provide splash screen files specific to device resolution, density, and orientation on Android. Because the same rules that apply to Android images apply to splash screens, you can follow the conventions discussed in Images and ImageView APIs.
- iOS
  - The filename must be `Default.png` with an uppercase **D**. Because this is platform specific, this file will typically be found in your project's `Resources/iphone` directory.
  - You can provide a retina version of your splash screen, named `Default@2x.png`.
  - For iPad and Universal apps, you should supply `Default-Landscape.png` and `Default-Portrait.png` iPad splash screen files.

References

- Developer Blog – Android Window Orientation Behavior Change for 1.7.2
- Layouts Positioning and the View Hierarchy

Summary

In this section, you learned how to specify, detect, and react to device orientation. You learned can lock the orientation of the entire app, specify the orientation of a window within your app, or react to orientation changes dynamically.