iOS 3D Touch

Introduction

Titanium SDK supports the Peek and Pop, and Quick Action features of 3D touch, which provides additional functionality and responsiveness for iOS users. Both features require a 3D Touch enabled device running iOS 9 or later. Note that you can only test 3D touch features on device.

- **Peek and Pop** provides a way for the user to quickly preview item content in the application by pressing on it, then optionally switching to the peeked item.
- **Quick Action** provides application shortcuts when the user presses the application icon on the Home Screen. The shortcuts allow the user to quickly perform an action with your application without navigating through the application.

3D Touch != Force Touch
Apple also has a feature called Force Touch for the Apple Watch, MacBook and Magic Trackpad, which should not be confused with 3D touch.

Quick Actions

Press firmly on the app icon in the Home screen to reveal the Quick Actions (or Application Shortcuts). Quick actions can either be static (always present) or dynamic (generated by the application).

To use quick actions, first create static or dynamic actions, then listen for the `shortcutitemclick` event to determine when the user taps a quick action.

Add Static Shortcuts

To define static shortcuts, add the `UIApplicationShortcutItems` key to the `ios/plist/dict` element in the `tiapp.xml` file. Set the `UIApplicationShortcutItems` key to an array of dict items containing additional keys that define the shortcut. You must specify the `UIApplicationShortcutItemType` and `UIApplicationShortcutItemTitle` keys :

- `UIApplicationShortcutItemTitle` (required): title of the quick action. May be assigned an `i18n` localized string in the `app.xml` file.
- `UIApplicationShortcutItemType` (required): unique identifier of the quick action passed to the event. Use a reverse domain notation, for example, `com.foocompany.fooapp.fooshortcut`.
- `UIApplicationShortcutItemSubtitle`: string to display underneath the title in the quick action. May be assigned an `i18n` localized string in the `app.xml` file.
- `UIApplicationShortcutItemIconType`: Set to a UIApplicationShortcutIcon enum to display an icon with the quick action.
- `UIApplicationShortcutItemIconFile`: Hash of a 35 x 35 dp icon to display with the quick action. To retrieve the hash of the icon, build the project once and retrieve the hash of the corresponding image from `build/iphone/Assets.xcassets.`
- `UIApplicationShortcutItemUserInfo`: custom dictionary that is passed to the event.

Localization
Place localized strings for the title and subtitle of the quick actions in the `app.xml` file as opposed to the `strings.xml` file.

Custom Quick Action Icons
If you specify a custom quick action icon using the `UIApplicationShortcutItemIconFile`, you need to enable slicing, that is, adding images to the asset catalog. To enable slicing, add the `use-app-thinning` element to the `ios` element in the `tiapp.xml` file and set the value to `true`.

tiapp.xml

```
<ti:app>
  <ios>
    <use-app-thinning>true</use-app-thinning>
  </ios>
</ti:app>
```

Example:

tiapp.xml

```
<ti:app>
    <ios>
        <plist>
            <dict>
                <!-- Define static shortcuts here -->
                <key>UIApplicationShortcutItems</key>
                <array>
                    <!-- First item uses hard-coded values and a default icon -->
                    <dict>
                        <key>UIApplicationShortcutItemIconType</key>
                        <string>UIApplicationShortcutIconTypeAdd</string>
                        <key>UIApplicationShortcutItemTitle</key>
                        <string>Add an Image</string>
                        <key>UIApplicationShortcutItemSubtitle</key>
                        <string>JPEG, PNG or GIF</string>
                        <key>UIApplicationShortcutItemType</key>
                        <string>com.appcelerator.example.addimage</string>
                    </dict>
                    <!-- Second item uses localized strings and a custom icon -->
                    <dict>
                        <!-- Find the corresponding hash in
build/iphone/Assets.xcassets -->
                        <key>UIApplicationShortcutItemIconFile</key>
                        <string>6ce9fb071294c440a20ff73b7c09fef2082c2206</string>

                        <!-- Title and subtitle in an i18n/<lang>/app.xml file -->
                        <key>UIApplicationShortcutItemTitle</key>
                        <string>add_photo_title</string>
                        <key>UIApplicationShortcutItemSubtitle</key>
                        <string>add_photo_subtitle</string>
                        <key>UIApplicationShortcutItemType</key>
                        <string>com.appcelerator.example.addphoto</string>
                        <!-- Custom dictionary (object) to receive in the event -->
                        <key>UIApplicationShortcutItemUserInfo</key>
                        <dict>
                            <key>myCustomKey</key>
                            <string>myCustomValue</string>
                        </dict>
                    </dict>
                </array>
            </dict>
        </plist>
    </ios>
</ti:app>
```

Add Dynamic Shortcuts

To create or remove dynamic shortcuts, use the `Ti.UI.iOS.ApplicationShortcuts` API.

To create a dynamic shortcut:

1. Use the `Ti.UI.iOS.forceTouchSupported` to test if the device supports 3D touch.
2. Create an instance of an ApplicationShortcut using the `Ti.UI.iOS.createApplicationShortcuts()` method.
3. Invoke the `addDynamicShortcut()` method on the ApplicationShortcut instance and pass method a dictionary with the following parameters:

- `itemtype` (required): unique identifier of the quick action passed to the event. Use a reverse domain notation, for example, `com.foocompany.fooapp.fooshortcut`.
- `title` (required): title of the quick action.
- `subtitle`: subtitle of the quick action displayed beneath the title.
- `icon`: icon to display with the quick action. May be assigned a `Titanium.UI.iOS.SHORTCUT_ICON_TYPE_*` constant, an image URL or a `Titanium.Contacts.Person`. If you use an image file, you need to enable slicing. See the note in the previous section for instructions.

To remove a dynamic shortcut, invoke the ApplicationShortcut instance's `removeDynamicShortcut()` method and pass it the `itemtype` identifier of the dynamic shortcut to remove or invoke the `removeAllDynamicShortcuts()` method to remove all dynamic shortcuts.

app.js

```
if (Ti.UI.iOS.forceTouchSupported) {
    // Create an applicationShortcuts instance
    appShortcuts = Ti.UI.iOS.createApplicationShortcuts();
    // Add a dynamic shortcut
    appShortcuts.addDynamicShortcut({
        itemtype: 'com.appcelerator.example.details',
        title: 'Open the last picture',
        icon: Ti.UI.iOS.SHORTCUT_ICON_TYPE_LOVE,
        userInfo: {
            filename: 'foo.png'
        }
    });
}
```

Respond to Quick Actions

When the user taps a Quick Action, the `shortcutitemclick` event is fired for the `Ti.App.iOS` static class. The payload includes all properties set for the static or dynamic shortcut except the icon. Simply use the `itemtype` to identify the shortcut and act accordingly.

```
function respondToShortcut(e) {
    switch(e.itemtype) {
        case 'com.appcelerator.example.addphoto':
            addPhoto();
            break;
        case 'com.appcelerator.example.details':
            openImage(e.userInfo.filename);

    }
};

Ti.App.iOS.addEventListener('shortcutitemclick', respondToShortcut);
```

Peek and Pop

To use Peek and Pop, press firmly on a peek-supported view. As you start applying more force, the rest of the screen will blur and reveal the preview. Maintaining the press will eventually open (pop) the detailed window. Swipe up while you peek to reveal any available quick actions.

To enable Peek and Pop:

1. Use the `Ti.UI.iOS.forceTouchSupported` to test if the device supports 3D touch.
2. Create a PreviewContext object using the `Titanium.UI.iOS.createPreviewContext()` method. Pass the method a dictionary

with the following properties:
    - `preview`: view object to use as the peeked view.
    - `actions`: array of PreviewActions or PreviewActionGroup objects to use as the quick actions.
    - `contentHeight`: height of the preview window. Defaults to fill most of the screen.
3. Add the `peek` and `pop` events to the created preview context to receive updates about the current preview state.
4. Attach the PreviewContext object to a view. Set the view's `previewContext` property to the PreviewContext object.

Example:

The following example shows how to attach a PreviewContext to an ImageView in an Alloy view. Note that the example omits the preview and pop views referenced in the controller code.

app/views/index.xml

```
<Alloy>
  <Window>
    <ImageView id="peekView"/>
  </Window>
</Alloy>
```

app/controllers.index.js

```
function popView() {
    Alloy.createController('pop').getView().open();
};

if (Ti.UI.iOS.forceTouchSupported) {

    var share = Ti.UI.iOS.createPreviewAction({
        title: "Share",
        style: Ti.UI.iOS.PREVIEW_ACTION_STYLE_DEFAULT
    });

    share.addEventListener("click", function(e) {
        // Implement share logic here
    });

    $.peekView.previewContext = Ti.UI.iOS.createPreviewContext({
        preview: Alloy.createController('preview').getView(),
        pop: popView(),
        actions: [share]
    });

    $.peekView.previewContext.addEventListener("peek", function() {
        Ti.API.warn("The view was peeked - Update the preview here if you want to");
    });

    $.peekView.previewContext.addEventListener("pop", function() {
        Ti.API.warn("The view was popped - Open the full context here");
        popView();
    });
}
```

Example

For a full example, see https://github.com/appcelerator-developer-relations/appc-sample-3dtouch.

Further Reading

- Ti.UI.iOS.ApplicationShortcuts API reference
- Ti.UI.iOS.PreviewContext API reference
- Apple Human Interface Guidelines: 3D Touch
- Appel Documentation: Getting Started with 3D Touch