Titanium Angular - Limitations and Workarounds

Titanium Angular is currently in an early preview stage. This means that some features are still under development or already available features may still be buggy.

TypeScript API

The TypeScript type definitions are generated from our API documentation. They are brand new and Titanium Angular is the first project to makes uses of those typings. We couldn't possibly test all available APIs for the first preview release so you may encounter some typings that are just wrong or they behave differently than you may know it from JavaScript. In those cases the easiest workaround is to explicitly cast the variable to the `any` type, so you can do whatever you want with it.

Casting Titanium types to any

```
const view = Titanium.UI.createView();
// cast only for a single method call or property access
(<any>view).callAnythingOnTheView();
// cast to a new variable of type any
const castedView = <any>view;
```

Oh, and don't forget to file a bug report under the FRAME project in Jira so we can fix it! Community help is always welcomed!

Working with Titanium elements

The basic features of all elements under the UI namespace are supported as of now. However, some platform specific views are still missing. Please refer to the following list to see which elements are **NOT** yet supported in templates (you can still create them programmatically in your components):

- Android
    - DrawerLayout
    - ProgressIndicator
    - SearchView
- iOS
    - CoverFlowView
    - SplitWindow
- Multiple platforms
    - NavigationWindow

Accessing the Titanium view/proxy of template elements

Sometimes you need to access the underlying Titanium view/proxy of template elements. To do so you can use the ViewChild/ViewChildren decorators from Angular in combination with the AfterViewInit lifecycle hook. First assign a template reference variable to the element you want to access and then retrieve the Titanium object instance via the `.nativeElement.titaniumView` property.

```
@Component({
  template: `
    <Label #myLabel></Label>
  `
})
class MyComponent implements AfterViewInit {
  @ViewChild('myLabel') labelRef: ElementRef;

  ngAfterViewInit() {
    const myLabel = this.labelRef.nativeElement.titaniumView as Titanium.UI.Label;
    myLabel.center = {x: 50, y: 50};
  }
}
```

Configuring template elements programmatically

One thing you may be missing if you are familiar with Alloy are some convenient features to configure your views directly from the template. For example, in Alloy you can do the following to configure a text fields button:

```
<Alloy>
    <Window>
      <TextField id="textfield" platform="ios">
        <LeftButton>
          <Button onClick="sayHi" class="textButton">left</Button>
        </LeftButton>
        <RightButton>
          <Button onClick="doAlert" class="textButton">right</Button>
        </RightButton>
      </TextField>
    </Window>
</Alloy>
```

Titanium Angular currently does not currently support this, but it is on our To-Do list. In the meantime you can utilize the `AfterViewInit` lifecycl e hook of your component to further configure it's view.

```
@Component({
  template: `
    <Window>
      <TextField #textfield platform="'ios'"></TextField>
    </Window>
  `
})
class ExampleComponent implements AfterViewInit {
  @ViewChild('textfield') textFieldRef: ElementRef;

  ngAfterViewInit() {
    const textField = this.textFieldRef.nativeElement.titaniumView;
    textField.leftButton = Ti.UI.createButton({title: 'left', height: 24, width: 24});
    textField.leftButton.addEventListener('click', () => { });
    textField.rightButton = Ti.UI.createButton({title: 'right', height: 24, width: 
24});
    textField.rightButton.addEventListener('click', () => { });
  }
}
```

Titanium Router

Currently you can only use the Angular's routing feature for normal window and tab groups. The support to open modal windows is also not yet implemented.

Note: Prior to Titanium 8.0.0, platform specific navigation components like the NavigationWindow on iOS are not yet supported.