Infor Homepages Widget SDK Migration Guide

Draft

Copyright © 2017 Infor

Important Notices

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Trademark Acknowledgements

The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade or service names referenced may be registered trademarks or trademarks of their respective owners.

Publication Information

Release:

Publication date: September 18, 2017

Document code:

Contents

About this guide

Version log

The version log describes the changes between versions of this document.

Version	Date	Changes
1.0		First version of the document.

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at www.infor.com/inforxtreme.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1 Introduction

1

Purpose

The purpose of this document is to describe the steps necessary to upgrade widgets to Homepages framework version 2.

Who Should Read This Document

Roles for which this document is primarily intended:

Role	Skills
Web Application developer	JavaScript, TypeScript, Angular, jQuery

Background

The Homepages framework is moving from AngularJS to Angular from version 12.0.17 (framework version 1.0.17). During a transition period spanning several releases the Homepages framework will support both AngularJS and Angular. In addition to this there are some APIs that will be duplicated to support this transition. When Homepages framework version 2 is released the support for AngularJS will be dropped and some of the older APIs will be removed. This means that all widgets must be migrated before Homepages framework version 2 is used in production environment. The exact date or version for Homepages framework version 2 has not yet been decided.

Widget migration

The amount of work required to migrate a widget depends on how it was implemented. The table below summarizes the areas that needs to be addressed depending on the current widget implementation. A simple widget implemented in JavaScript and jQuery may only require a minor change to the manifest while a complex AngularJS widget will need a lot of work.

	Manifest	Widget API	ION API*	Angular migration
JavaScript jQuery	X		X	
TypeScript jQuery	X	X	X	
JavaScript AngularJS	X	X	X	X
TypeScript AngularJS	X	X	X	X

* Only applies to widgets that use ION API

Chapter 2 Manifest changes

2

This chapter describes the widget manifest changes required for all types of widgets.

New mandatory property

A new mandatory property called “framework” has been added to the widget manifest. The purpose of this property is to specify the type of framework the widget is using. The reason to have this property in the manifest is to avoid overhead when loading a widget.

Property name: framework
Valid values: jquery | angular

Example:
```
"framework": "angular"
```

When the value “angular” is specified the framework will create a container component that will host the Angular widget component. When the value “jquery” is specified the framework will only provide a DOM element where the widget may add its content (same as before).

Default value

During the transition period the default value of this property will be “angularjs”. This value will only be used temporarily during the transition period. Existing widgets that do not specify the framework property in the manifest will get the default value which works for both AngularJS and jQuery widgets. This means that existing widgets should continue to work without any changes during the transition period.

After the transition period the default value will be changed to “jquery” but widgets should always specify the framework property instead of relying on the default value.

Chapter 3 API Changes

3

This chapter describes the framework API changes that needs to be addressed by some types of widgets.

Observable instead of Promise

All asynchronous functions in the new APIs now returns Observable<T> instead of Promise<T>. The promise implementation was part of AngularJS so that has now been replaced by Observable that is implemented by RxJS. The reason for using RxJS was that Angular depends on RxJS so it had to be included anyway.

In most cases the change is as simple as replacing calls to the function “then” of a Promise with a call to the function “subscribe” of an Observable. An example of this can be seen in the ION API section below.

For additional information on how to use Observable and other parts of RxJS see: http://reactivex.io/rxjs/

New interfaces

To be able to support both old and new widgets during the transition period a set of new interfaces have been introduced. Since these interfaces need to coexist with the old interfaces they all have the suffix “2”. When Homepages framework version 2 is released the old interfaces will be removed. The affected interfaces are listed in the table below.

If you have used any of the old interfaces in the widget code replace them with the new version and then address any issues related to the change. See the following sections for more details.

Old interface	New interface
IWidgetContext	IWidgetContext2
IWidgetInstance	IWidgetInstance2

IWidgetSettingsContext	IWidgetSettingsContext2
IWidgetSettingsInstance	IWidgetSettingsInstance2
IAngularWidgetConfig	IAngularWidgetConfig2

Widget factory function

The signature of the widget factory function has changed so that it uses the new version 2 interfaces.

Before:

```
widgetFactory(context: IWidgetContext): IWidgetInstance;
```

After:

```
widgetFactory(context: IWidgetContext2): IWidgetInstance2;
```

Widget instance implementation

Widget instance classes that previously implemented the IWidgetInstance interface should now implement the new IWidgetInstance2 interface. The changes to the interface are summarized below. The amount of changes required for each widget depends on what functionality that has been used.

New interface

This change affects all widgets.

The widget instance implementation should implement the new IWidgetInstance2 interface.

Note that there may be some cases where no change is required, such as a basic JavaScript widget.

Asynchronous metadata

This change only affects widgets that loads settings metadata asynchronously.

The optional asynchronous function getMetadataAsync should return an Observable instead of a Promise. This only affects widgets that previously implemented the getMetadataAsync function.

Angular configuration

This change only affects widgets implemented in Angular.

The angularConfig property used for Angular widgets is now of the type IAngularWidgetConfig2. Note that the new IAngularWidgetConfig2 is very different compared to the previous IAngularWidgetConfig interface used for AngularJS widgets due to the changes between AngularJS and Angular.

Widget settings factory function

This change only affects widgets with a custom settings UI.

The signature of the widget settings factory function for widgets with a custom settings UI has changed so that it uses the new version 2 interfaces.

Before:

```
widgetSettingsFactory?: (context: IWidgetSettingsContext) => IWidgetSettingsInstance;
```

After:

```
widgetSettingsFactory?: (context: IWidgetSettingsContext2) => IWidgetSettingsInstance2;
```

Widget settings instance implementation

This change only affects widgets with a custom settings UI.

Widget settings instance classes that previously implemented the IWidgetSettingsInstance interface should now implement the new IWidgetSettingsInstance2 interface.

Angular configuration

This change only affects widgets implemented in Angular.

The angularConfig property used for Angular widgets is now of the type IAngularWidgetConfig2. Note that the new IAngularWidgetConfig2 is very different compared to the previous IAngularWidgetConfig interface used for AngularJS widgets due to the changes between AngularJS and Angular.

ION API functions

This change only affects widgets that makes ION API requests.

The two asynchronous functions getIonApiContextAsync and executeIonApiAsync in IWidgetContext2 returns Observable<T> compared to Promise<T> in IWidgetContext.

Before:

```
this.widgetContext.executeIonApiAsync(request).then(response => {
   this.updatePhoto(<IUserDetailResponse>response.data);
}, (error) => {
   this.onRequestError(error);
});
```

After:

```
this.widgetContext.executeIonApiAsync(request).subscribe(response => {
   this.updatePhoto(<IUserDetailResponse>response.data);
}, (error) => {
   this.onRequestError(error);
});
```

Chapter 4 Development environment changes

4

This chapter describes the changes to the development environment for widgets.

Introduction

Since the Homepages framework is now using Angular there are some development environment changes that affects all widgets.

The Samples project in the SDK is updated so you can use that as a base. If you want to keep an existing project, you must update it to match the Samples project by moving all relevant files to the existing project. This includes script files, new index.html, package.json and other configuration files.

New and updated files

If you are updating an existing project the following files and directories needs to be updated.

- Samples/Widgets/
    - scripts
    - svg
    - index.html
    - lime-config.js
    - lime-main.js
    - tsconfig.json

New dependencies

The package.json file has new dependencies that needs to be installed using the Install.cmd file or the “npm i” command.

New syntax for development page container

The syntax for the development page container in the index.html file has changed slightly. The reason is that the development container is now an Angular component. The actual values that are set are the same so it is just syntax changes as shown below.

The attributes names are now specified in camel-case and enclosed in square brackets, compared to the previous kebab-case. The values also must be enclosed in single-quotes which can be easy to miss.

Before:

```
<lm-page-container dev-widget="infor.sample.helloworld"></lm-page-container>
```

After:

```
<lm-page-container [devWidget]="'infor.sample.helloworld'"></lm-page-container>
```

Chapter 5 AngularJS to Angular migration

5

This chapter describes the steps for migrating a widget implemented in AngularJS to Angular.

Introduction

Before starting the migration of an existing AngularJS widget, you should take some time to consider the technology choice. The current options are Angular or jQuery and even if the widget was previously implemented in AngularJS the choice is not obvious. Depending on the complexity of the widget UI you might want to choose jQuery instead of Angular.

If you decide to migrate to Angular the rest of this chapter will give an overview of the required steps.

Resources

If you are completely new to Angular development you might want to try the QuickStart and follow the tutorial. The Angular concepts will apply to widget development but the development environment will differ.

https://angular.io/guide/quickstart
https://angular.io/tutorial

There is guide for upgrading from AngularJS to Angular on the angular.io site. It is recommended to read the entire guide but you should be aware that not all parts apply to Homepages widget development.

https://angular.io/guide/upgrade

There is a Quick Reference that describes the syntax changes for template directives etc.

https://angular.io/guide/ajs-quick-reference

There is also a lot of more detailed information such as the Fundamentals section starting with the Architecture Overview.

https://angular.io/guide/architecture

Supported Angular modules

Widgets are only allowed to use a subset of the available Angular modules. The modules that are supported for widget development are listed below.

- @angular/core
- @angular/common
- @angular/common/http
- @angular/forms
- @infor/sohoxi-angular

Angular AOT vs JIT

The Homepages framework is built using AOT (ahead of time) compilation for performance reason. This means that the HTML templates are compiled once at build time instead of each time in runtime which is the case with JIT.

The goal is use to use AOT for widgets as well in the future but the problem is that Angular does not yet guarantee compatibility of the generated component factories between major versions. This would mean that widgets built for AOT in Angular version 4 might break in Angular 5. For the time being the widgets will be compiled using JIT.

Since the goal it to introduce AOT for widgets in the future it is recommended to keep the widget code compatible with AOT. See the links below for more information about Angular AOT and how to keep the code AOT compatible.

Angular AOT

https://angular.io/guide/aot-compiler

Angular AOT do’s and don’ts

https://github.com/rangle/angular-2-aot-sandbox#aot-dos-and-donts

Migration example

There are many ways to migrate a widget so the steps and the order of the steps described here is just one suggestion. In the section after this there is also an example of a minimal Angular widget.

1. Install and configure the Samples from the new SDK
2. Test the infor.sample.angular.helloworld sample widget
    a. This is a very basic Angular widget that will verify that you can run an Angular widget in the development page container. This will rule out any issues related to the Angular dependencies.
3. Copy the existing widget directory to the Sample solution
4. Add imports for the new interfaces IWidgetComponent, IWidgetContext2, IWidgetInstance2
    a. `import { IWidgetComponent, IWidgetContext2, IWidgetInstance2 } from "lime";`
5. Change the main widget AngularJS controller class to an Angular component
    a. Optionally change the name
        i. Ex: MyWidgetCtrl > MyWidgetComponent
    b. Add the @Component annotation
    c. Add a very simple preliminary test template
        i. `@Component({ template: `<div>MyWidgetComponent</div>` })`
    d. Implement the AfterViewInit and IWidgetComponent interfaces
        i. export class MyWidgetComponent implements AfterViewInit, IWidgetComponent {
    e. Add the input properties defined in IWidgetComponent
        i. @Input() widgetContext: IWidgetContext2;
        ii. @Input() widgetInstance: IWidgetInstance2;
    f. Add the ngAfterViewInit function
        i. ngAfterViewInit() { }
6. Add an Angular module for the widget
    a. Create a new class
        i. Ex: MyWidgetModule
    b. Add the NgModule annotation with the arrays imports, declarations and entryComponents
    c. Import the modules that the widget requires

            i. Ex: `CommonModule, FormsModule, SohoComponentsModule`

    d. Add the widget component to the declarations and entryComponents arrays

7. Change the widget factory function
    a. Use the new interfaces IWidgetContext2 and IWidgetInstance2
    b. Return an instance of IAngularConfig2 with the properties moduleType and componentType set
8. Verify that the widget compiles without errors
9. Change the index.html file to use your widget in the development page container
10. Run the widget and verify that the content of the template is displayed in the widget
11. Migrate the old AngularJS template to an Angular template
    a. Add smaller parts at a time and make the necessary syntax changes
    b. Verify that the template works before adding more parts

Example widget

The code below show a minimal example widget. It defines a module that imports some other common module and a single widget component.

```
import { AfterViewInit, Component, Input, NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { SohoComponentsModule } from "@infor/sohoxi-angular";
import { IWidgetComponent, IWidgetContext2, IWidgetInstance2 } from "lime";

@Component({
    template: `<div>MyWidgetComponent</div>`
})
export class MyWidgetComponent implements AfterViewInit, IWidgetComponent {
    @Input() widgetContext: IWidgetContext2;
    @Input() widgetInstance: IWidgetInstance2;

    ngAfterViewInit() {
    }
}

@NgModule({
    imports: [CommonModule, FormsModule, SohoComponentsModule],
    declarations: [MyWidgetComponent],
    entryComponents: [MyWidgetComponent]
})
export class MyWidgetModule {
}

export var widgetFactory = (context: IWidgetContext2): IWidgetInstance2 => {
    return {
        angularConfig: {
            moduleType: MyWidgetModule,
            componentType: MyWidgetComponent
        }
    };

};
```