Infor Homepages Widget SDK Angular AOT Adoption Guide

Draft

Publication Information

Release:

Publication date: March 28, 2018

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 Homepages Angular widgets with support for AOT.

Who Should Read This Document

Roles for which this document is primarily intended:

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

Background

This document is intended for developers developing Angular widgets and the adoption instructions only applies to Angular widgets.

The Homepages framework is currently in a transition period where the Homepages framework is moving from AngularJS to Angular starting with 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.

How to migrate your AngularJS widgets to Angular is documented in “MigrationGuide.pdf”. You have the option of using jQuery or Angular. With Angular there is a cost of JIT compiling that will affect the load time of widgets in Homepages. The compilation time will differ depending on the client machine performance but it can range from 30 ms to more than 2000 ms for an individual widget depending on complexity. AngularJS also had JIT compilation. In Homepages 12.0.23 and later there is support

for AOT (Ahead of Time) compiled widgets which means that the widget templates will already be compiled.

All Angular widgets should be built with AOT and this document describes the process and the changes required to build your widget package for AOT. A new node script called “homepages” has been added to the Homepages SDK. It will replace the minify.js and the minifywidget.js scripts although the previous scrips are still available in the SDK. The homepages.js script supports packaging all types of widgets.

A 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 6 might break in Angular 7.

Homepages in 12.0.23 is on Angular <TODO insert version number here>. To keep track of the Angular compiler version that was used the version will be written in the widget manifest as aotVersion.

The widget package will contain two versions of the widget, one for JIT and one AOT version with generated component factories. The Homepages framework will load the AOT Compiled version if the compiled version matches with the frameworks version. If the versions don’t match the JIT version will be used.

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

Chapter 2 AOT Widget Project Adoption

2

This chapter describes the steps necessary to use AOT for a Homepages widget.

Install new Homepages SDK

Install or update the new Homepages SDK.

If there is an existing project, make sure to re-apply the SDK files to your file structure.

1. Copy and replace all SDK files
2. Run npm i in the Samples folder if you are using the SDK or in the root folder of the project to update all node dependencies.

New manifest property

A new optional manifest property called “aotVersion” has been added to the widget manifest. The purpose of this property is to specify the major Angular version the widget is AOT compatible with.

This property should only be added to the manifest of Angular widgets that supports AOT. The value of this property can be blank in most cases since it is updated automatically when using the homepages.js script to package a widget. The only reason to specify a value other than blank is if you package the widget without using the homepages.js script.

The existence of this property in the widget manifest is used to determine if the ngc compiler will be used to create an AOT build or not.

Property name: aotVersion

Examples:

```
"aotVersion": ""
"aotVersion": "6"
```

Framework API Changes

The IAngularWidgetConfig2 interface has a new optional property called moduleFactory that is used to set the Angular AOT module factory.

To support AOT a widget must have a secondary widget factory implementation that returns a widget instance with the module factory configured. This is done by introducing a secondary module file as described later in this document.

Preparations

Change includes for SoHo Xi components

Instead of including all SoHo components by referencing the index file the widget should only include those SoHo Xi components that are used. This will decrease the size of the generated factories.

Instead of:

```
import { SohoComponentsModule } from "@infor/sohoxi-angular";
```

Specify the individual components that are used:

```
import { SohoButtonModule } from "@infor/sohoxi-angular";
```

Widget directory name

The widget directory name must have the same name as the widget's ID. This is the naming convention used in the Samples project. If the widget directory name does not match the widgetId property in the manifest it needs to be changed to match the widgetId.

Widget directory structure

Sub directories in the widget package are no longer supported. All files in a widget package must be in the root. The code may still use sub directories since all code will be bundled. HTML templates must be inline in the component since template URLs for a widget cannot be resolved in the current framework.

Make sure that there are no sub directories that contains anything except for code in the widget directory. Rearrange if needed.

Verify preparations

Before doing any other changes verify that you can build the project in Visual Studio and using tsc.

Widget adoption

This section describes the necessary steps to add two different modules, one for AOT and one for JIT using an example widget from the Samples.

Starting with the helloworld widget.

1. Verify that the directory has the same name as the widget's ID.
2. Edit the widget.manifest file and add the aotVersion property with a blank value.
    a. Add this new property after the framework property.
    b. "aotVersion": ""
3. Locate the file that contains the widget factory. This file should have the same name as the module property in the widget manifest. The default filename would be widget.ts.
4. Copy the widget.ts file and paste it in the same directory and change the name to main.ts.
    a. The file name "main" is not mandatory, another name can be used.
    b. The main.ts file will contain all code that is shared for AOT and JIT.
5. Remove the widget factory function from the main.ts file.
6. Remove all code except the widget factory function from the widget.ts file.
7. Import classes used from the main.ts file and remove unused imports.

    For the helloworld widget the widget.ts file will look like this:

```
import { IWidgetContext2, IWidgetInstance2 } from "lime";
import { HelloWorldModule, HelloWorldComponent } from "./main";

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

};
```

8. Copy the newly updated widget.ts file and create a new file.
9. Rename the copy to "widget-aot.ts". This will be the factory file for AOT.
10. Edit the widget-aot.ts file
    a. Import the module factory from "./main.ngfactory"
    b. Set moduleFactory property to the AOT factory
    c. Delete the moduleType property

```
import { IWidgetContext2, IWidgetInstance2 } from "lime";
import { HelloWorldModuleNgFactory } from "./main.ngfactory";
import { HelloWorldComponent } from "./main";

export const widgetFactory = (context: IWidgetContext2): IWidgetInstance2 => {
      return {
            angularConfig: {
                  moduleFactory: HelloWorldModuleNgFactory,
                  componentType: HelloWorldComponent
            }
      };

   };
```

The main.ngfactory file does not exist yet as it will be generated during the ngc build when AOT compilation is done.

11. Since the widget now uses multiple modules (more than one script file) the SystemJS configuration must be modified before the widget can be tested. Open the lime-config.js file and locate the packages section. Add a new package configuration like the one below for the widget, unless it already exists.

    ```
    "infor.sample.angular.helloworld ": { defaultExtension: "js" }
    ```

12. If you are using Visual Studio or tsc to build you should exclude the widget-aot.ts file in the tsconfig.json file.

    Example:

    ```
    "exclude": [ "./**/*-aot.ts" ]
    ```

The next step would be to run the homepages command script to create the widget package. There will likely be issues in the code the needs to be resolved before the widget can be successfully AOT compiled. The common issues are:

- Use of private properties in templates.

- Use of properties that does not exist on an object.

- Failing to import all SoHo Modules used.

Chapter 3 Widget Packaging

3

This chapter describes how to package the widget.

Homepages package script

The homepages.js script located in the Samples directory of the Homepages SDK can compile, minify and package widgets.

Before running the script make sure that you have installed any new dependencies by updating dependencies in the Samples folder or your project folder by executing:

```
npm i
```

Run the homepages script with node.

Examples

```
node homepages help
```

```
node homepages pack "infor.sample.angular.helloworld"
```

```
node homepages pack --
widgets="infor.sample.angular.helloworld,infor.sample.angular.quicknote"
```

```
node homepages pack --widget "Widgets/infor.sample.angular.helloworld" --
outputPath "C:\Builds"
```

```
node homepages pack --widget "C:\\Source\Widgets\infor.sample.angular.helloworld"
--outputPath "C:\Builds" –zip=false
```

The default output location with be in a Builds directory relative to the script location unless outputPath is passed as a parameter.

Output

The output directory will contain the widget package zip, for example: infor.sample.angular.helloworld-1.0.20180323-114907.zip

Shared modules

Shared modules are supported like before. It is however important that the TypeScript file for the shared module is available in the same directory as the widget. It can be copied by a script prior to building the widget.