Step 1: Index

Table of content

1 Step 1: Index

Because an application would normally live in a folder, and the default file that web servers return when the folder itself is requested is often configured to be `index.html`, the name of our app's index HTML file is indeed `index.html`.

Minimal Content

The heart of an application is defined in its Component part, which means that we should aim for very minimal content in the index.html file itself. Like this:

```
<!DOCTYPE HTML>
<html>

    <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta charset="UTF-8">

        <title>TDG Demo App</title>

        <script id="sap-ui-bootstrap"
        src="/path/to/resources/sap-ui-core.js"
        data-sap-ui-libs="sap.m"
        data-sap-ui-theme="sap_bluecrystal"
        data-sap-ui-xx-bindingSyntax="complex"
        data-sap-ui-resourceroots='{
            "sap.ui.demo.tdg": "./"
        }'>
        </script>

        <script>
            sap.ui.getCore().attachInit(function() {
                new sap.m.Shell({
                    app: new sap.ui.core.ComponentContainer({
                        height : "100%",
                        name : "sap.ui.demo.tdg"
                    })
                }).placeAt("content");
            });
        </script>
    </head>

    <body class="sapUiBody" id="content" />

</html>
```

We have a header and the standard variant SAPUI5 bootstrap, the application startup, and a particular focus on resources and namespaces.

Header and Bootstrap

In the header, we have the HTML5 document type declaration, and a couple of meta tags. One that instructs Internet Explorer to use the newest rendering mode; for SAPUI5 mobile this would be in preparation for future IE support, for SAPUI5 desktop this is required to make sure IE does not use a compatibility mode. The other is to declare that the character set used in the app is UTF8 (change this if required to whatever is necessary).

We also have the SAPUI5 bootstrap; in this example we have the so-called "standard variant", pulling in the SAPUI5 core from wherever it is normally served. With the bootstrap there are some data parameters. These parameters serve to load the `sap.m` library, so that we can specify the `sap.m.Shell` control later on (`data-sap-ui-libs`), declare the theme to be "Blue Crystal" (`data-sap-ui-theme`), set up the extended syntax for calculated fields in property bindings (`data-sap-ui-xx-bindingSyntax`) and, finally, where the resources, or artifacts, can be found (`data-sap-ui-resourceroots` - see below for more information on this).

The Application Startup

Once the SAPUI5 core has been loaded in the bootstrap, we start the application on the Init event via `sap.ui.getCore().attachInit`. Starting our app involves instantiating an `sap.ui.core.ComponentContainer` and placing it inside an sap.m.Shell in the body of our HTML document. Note that we're using the "content" id directly on the HTML `<body>` element, rather than in a child `<div>` element.

The use of the Shell is to control the width of the app on the desktop. On particularly large monitors, a full-width rendering of an application would be too wide. Placing the app's core inside a Shell container will give you appropriate left and right vertical margins.

Resources and Namespaces

Your application should live inside a namespace. This sample application's identifier is "tdg", within the "sap.ui.demo" namespace, thus the name `sap.ui.demo.tdg` is the fully qualified identifier for this application.

This name is used in two places in the index file:

- in the bootstrap's `data-sap-ui-resourceroots` data parameter
- in the specification of the Component to load into the `sap.ui.core.ComponentContainer`

Pairing a namespace and folder location in the `data-sap-ui-resourceroots` declares where artifacts with that namespace can be found. In this example we are saying that the artifacts starting "sap.ui.demo.tdg" are to be found in the same folder ("./") as this index file.

Note

While we're here, notice the way that the namespace and folder location are specified in a JSON structure inside the value of an HTML tag's attribute. You'll come across this syntax again when we see the use of complex binding syntax in declarative (XML) views, where paths, types and formatters are specified for property bindings.

When the component is specified with the name property of the `sap.ui.core.ComponentContainer`, it is by means of the "sap.ui.demo.tdg" namespace. By default, the container will look for a component called "Component" in this namespace, which thus translates to `sap.ui.demo.tdg.Component`. This will be found in the file Component.js in the same folder, because we've specified this same folder as being where resources in the `sap.ui.demo.tdg` namespace are to be found.

Further Notes

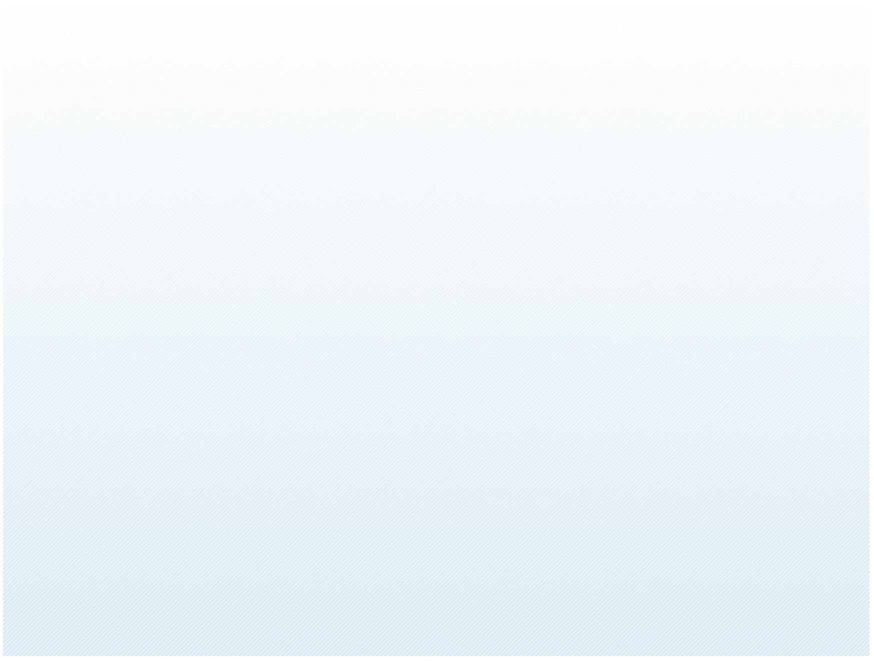
Specific applications may require additional script tags to load external files or some HTML structure within the body or a script tag before loading SAPUI5 to configure the SAPUI5 runtime. This suggestion here does not say such extensions are not allowed.

Progress Check

This is the first step, so we're not expecting much. All we have is the index.html file.

```
tdg/
  |
  +-- index.html
```

But it's the root container for our whole application, so at least we can try to call it up in our browser. This is what we get.



Ok. Not a lot. But not an entirely blank screen - we can see already that the styling is from the Blue Crystal theme. Most importantly, we see the significant error in the developer console:

```
Uncaught Error: failed to load 'sap/ui/demo/tdg/Component.js' from ./Component.js: 404 -
        Not Found
```

This is not unexpected. The Component Container has been instantiated and is trying to retrieve the Component ... but failing. In the next step, we'll look at making that Component available.