CHRIS' BLOG

SmartUI development without the SmartUI SDK in OpenText Content Server

Christopher Meyer Thu Apr 23 2020

OpenText Content Server JavaScript SmartUI

A few weeks ago I was tasked with writing a SmartUI widget. It had been a year since I last worked with the SmartUI SDK, and I recalled being somewhat overwhelmed. Everyone I had spoken with expressed grief due to the lack of documentation, non working examples, lack of support with newer versions of Node, and breaking changes with new versions. Great.

At the time I had a coworker walk me through the steps of building a simple "Hello World" widget. This helped a lot, and I was able to modify the example to display the output of a Simplate. I left it at that.

While revisiting the project I noticed the output from the SDK wasn't anything special. It was little more than a concatenation and minification of the project files. I then realised all this development could be done without the overhead of the SmartUI SDK.

The SmartUI SDK is not required for SmartUI development

To be clear, I'm not commenting on the licensing aspect of the SmartUI SDK. I'm only asserting that SmartUI development can be done without it. A Content Server developer license is probably still required to write the OScript to register the plugin.

A Basic Widget

The recipe for creating a basic widget without the SmartUI SDK is as follows. Replace `mymodule` with the name of your module:

Register the plugin in your module

- set `fModulePrefix` to the name of your module (e.g., `mymodule`)
- set `fModuleName` to the verbose name of your module (e.g., "My Great Module")
- optionally, set `fExtensionPathPrefix` to the relative path in the support directory (e.g., `smartui/` makes `opentext/support/mymodule/smartui/` the location of your JS, CSS, images, etc.)

The remaining file paths are relative to the `opentext/support/` directory.

Add the plugin configuration file

Create a `mymodule/smartui/mymodule-extensions.json` file.

The `{fModulePrefix}-extensions.json` is the default filename and can be changed by overriding the `CSUI:CsuiRoot:Extension:fPluginConfigName` feature in the module.

```json
{
    "csui/models/server.module/server.module.collection": {
        "modules": {
            "mymodule": {
                "version": "1.0"
            }
        }
    },
    "csui/models/widget/widget.collection": {
        "widgets": {
            "mymodule": [
                "mymodule/widgets/mymodule"
            ]
        }
    }
}
```

Add the bundle index

Create the `mymodule/smartui/bundles/` directory and add the `mymodule/smartui/bundles/mymodule-index.json` file.

The `{fModulePrefix}-index.json` is the default filename and can be changed by overriding the `CSUI:CsuiRoot:Extension:fBundleIndexName` feature in the module.

```
        "mymodule/widgets/mymodule/mymodule.view",
        "json!mymodule/widgets/mymodule/mymodule.manifest.json"
    ]
}
```

Add a CSS file

Create a `mymodule/smartui/bundles/mymodule-all.css` file:

```css
.binf-widgets .mymodule-smartui {
    background-color: #fff;
    width: 100%;
    height: 100%
}
```

Add the main JavaScript file

Create the `mymodule/smartui/bundles/mymodule-all.js` file:

```js
csui.define("mymodule/widgets/mymodule/mymodule.view", ["csui/lib/marionette"], fu

    return Marionette.ItemView.extend({
        className: 'mymodule-smartui',
        template: function() {
            return 'Loading...'
        },
        onRender: function() {
            this.$el.html(this.options.data.message)
        }
    });

});

csui.define("json!mymodule/widgets/mymodule/mymodule.manifest.json", {
    "$schema": "http://opentext.com/cs/json-schema/draft-04/schema#",
    "title": "MyModule SmartUI Panel",
    "description": "This panel displays a message.",
    "kind": "tile",
    "supportedKinds": ["fullpage", "tile", "header"],
            "message": {
                "title": "Message",
                "description": "The message to display",
                "type": "string"
            }
        }
    }
});

// this loads the CSS file above
csui.require(["require", "css"], function(_require, _css) {
    _css.styleLoad(_require, "mymodule/bundles/mymodule-all")
});
```

A few notes...

The naming of these files are important and relate to one another. It should be possible to replace each instance of "mymodule" with the name of your module to get the widget to work.

The widget should display as a choice when editing the SmartUI page layout. A layout can be chosen (Single Width, Double Width, or Full Page) and a message can be configured. I suspect, but haven't confirmed, that a similar approach can be used with any of the SmartUI integration points.

Content Server needs to be restarted whenever `mymodule-extensions.json` or `mymodule-index.json` are modified. Restarts are not necessary when modifying `mymodule-all.js` or `mymodule-all.css` , but care should be taken to ensure the browser doesn't cache these files when refreshing the page.

Minification is not a concern for small JavaScript and CSS files. You can still minify and transpile the files if you want (e.g., if you'd like to write in ES6 and support older browsers), but the SmartUI SDK isn't required for this.

One thing I don't understand is why the SmartUI SDK maps the `define` and `require` global functions to `csui.define` and `csui.require` in its build process. This has given me some problems, which I'll discuss in a future post.

Next steps

frameworks like React, Vue.js, and Angular taking the spotlight.

My worry with SmartUI development is forwards compatibility. Will something I write today work with future versions of SmartUI? What if OpenText abandons these frameworks for something else in the future? Who actually likes Bootstrap?

In an upcoming blog post I'll discuss how I'm addressing forwards compatibilty with the use of some modern JavaScript frameworks. Stay tuned!

0 Comments schwiiz.org 1 Login

♡ Favorite 2 Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.