Exercise 2-1

Exploring the SharePoint Framework API

In this exercise, you'll explore a few different APIs available to you in the SharePoint Framework that are useful in many scenarios.

Open the web part project from the previous exercise or create a new webpart project with no framework

## Checking the current page mode and state

Locate the web part file **src/webparts/helloWorld/HelloWorldWebPart.ts**.

At the top of the file, locate the following line:

import { Version } from '@microsoft/sp-core-library';

Update this line to the following to import a few additional objects and enumerations into the file:

import {

Version,

DisplayMode,

Environment,

EnvironmentType

} from '@microsoft/sp-core-library';

Now locate the render() method and add the following two lines to the top of it, right after the method declaration:

const pageMode : string = (this.displayMode === DisplayMode.Edit)

? 'You are in edit mode'

: 'You are in read mode';

const environmentType : string = (Environment.type === EnvironmentType.Local)

? 'You are in local environment'

: 'You are in SharePoint environment';

These two lines determine the current mode of the page and environment in which the web part is running. These can prove to be useful for the web part to render either mock data when running in the local web server, or real SharePoint data from a SharePoint list if you're running the web part in a SharePoint environment such as the hosted workbench or in a real SharePoint page.

Display the values in these two members by adding the following two lines to the HTML that is added to the <div> element that contains the rendered web part. Add these lines just after the **Welcome to SharePoint** message:

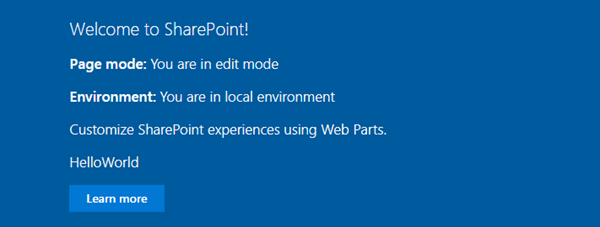
<p class="${ styles.subTitle }"><strong>Page mode:</strong> ${ pageMode }</p>

<p class="${ styles.subTitle }"><strong>Environment:</strong> ${ environmentType }</p>

Test the web part to see what we get. Start the local web server using the provided gulp **serve** task:

gulp serve

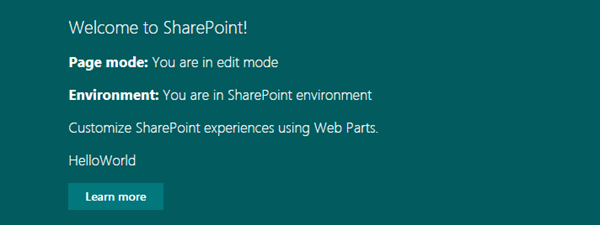
When the browser loads the local workbench, add the web part to the page. Notice the values of the page mode and environment type:



Now, open a new browser tab and navigate to one of your SharePoint Online sites and append the following to the end of the root site's URL: **/\_layouts/workbench.aspx**.

Add the web part to the page.

Notice the values of the page mode and environment type:



## Leverage the SharePoint Framework Logging API

The SharePoint Framework also provides a way to log messages to the console with additional information than the traditional console.log() method provides.

Locate the web part file **src/webparts/helloWorld/HelloWorldWebPart.ts**.

At the top of the web part file, locate the following lines:

import {

Version,

DisplayMode,

Environment,

EnvironmentType

} from '@microsoft/sp-core-library';

Add an additional reference to Log the existing list so it looks like this:

import {

Version,

DisplayMode,

Environment,

EnvironmentType,

Log

} from '@microsoft/sp-core-library';

Add the following lines to the end of the render() method, immediately before it closes. These will look different messages to the browser's console window:

Log.info('HelloWorld', 'message', this.context.serviceScope);

Log.warn('HelloWorld', 'WARNING message', this.context.serviceScope);

Log.error('HelloWorld', new Error('Error message'), this.context.serviceScope);

Log.verbose('HelloWorld', 'VERBOSE message', this.context.serviceScope);

Go back to the browser tab containing the local workbench and open your browser's developer tools.

Open the **Console** tab (it may have a slightly different name depending on the browser you're using).

There will be numerous messages logged to the console, so use the filter technique to filter based on the name of your web part, **HelloWorld**.

Notice in the following image that each message is prefixed with the unique name of the web part.



## Addressing delayed loading web parts

Your web part may have a number of calculations to perform or have a delay in fetching data before it renders the first time. Thankfully the SharePoint Framework provides an API you can use to address this.

Locate the web part file **src/webparts/helloWorld/HelloWorldWebPart.ts**.

Locate the following line in the render() method:

this.domElement.innerHTML = `

Add the following lines just before the this.domElement.innerHTML line:

this.context.statusRenderer.displayLoadingIndicator(this.domElement, "message");

setTimeout(() => {

this.context.statusRenderer.clearLoadingIndicator(this.domElement);

Add the following line just after the code that attaches the event handler to the button:

}, 5000);

The code you just added displays the string **message** in a loading indicator for 5 seconds before clearing it out and writing HTML to the page.

The resulting code should look similar to this:

public render(): void {

const pageMode : string = this.displayMode === DisplayMode.Edit ? 'You are in edit mode' : 'You are in read mode';

const environmentType : string = Environment.type === EnvironmentType.Local ? 'You are in local environment' : 'You are in sharepoint environment';

this.context.statusRenderer.displayLoadingIndicator(this.domElement, "message");

setTimeout(() => {

this.context.statusRenderer.clearLoadingIndicator(this.domElement);

this.domElement.innerHTML = `

<div class="${ styles.helloWorld }">

<div class="${ styles.container }">

<div class="${ styles.row }">

<div class="${ styles.column }">

<span class="${ styles.title }">Welcome to SharePoint!</span>

<p class="${ styles.subTitle }"><strong>Page mode:</strong> ${ pageMode }</p>

<p class="${ styles.subTitle }"><strong>Environment:</strong> ${ environmentType }</p>

<p class="${ styles.description }">${escape(this.properties.description)}</p>

<a href="#" class="${ styles.button }">

<span class="${ styles.label }">Learn more</span>

</a>

</div>

</div>

</div>

</div>`;

}, 5000);

Log.info('HelloWorld', 'message', this.context.serviceScope);

Log.warn('HelloWorld', 'WARNING message', this.context.serviceScope);

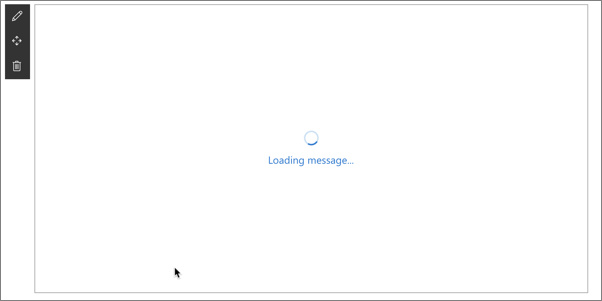
Log.error('HelloWorld', new Error('Error message'), this.context.serviceScope);

Log.verbose('HelloWorld', 'VERBOSE message', this.context.serviceScope);

}

Go back to the browser and, if it hasn't reloaded, refresh the page.

When the web part initially loads, it displays the loading message:



After five (5) seconds, notice the web part is rendered as it was before because the timeout concludes.

Close the browser and stop the local web server by pressing CTRL+C in the command prompt.

## Summary

In this exercise, you explored a few different APIs available to you in the SharePoint Framework that are useful in many scenarios.