**Building my first Library component using SharePoint Framework**

The library component type in the SharePoint Framework (SPFx) enables you to have independently versioned and deployed code served automatically for the SharePoint Framework components with a deployment through an app catalog. Library components provide you an alternative option to create shared code, which can be then used and referenced cross all the components in the tenant.

You can create a library component solution by selecting **Library** as the **Component Type** when creating new projects with the SPFx Yeoman generator.

Library components have following characteristics:

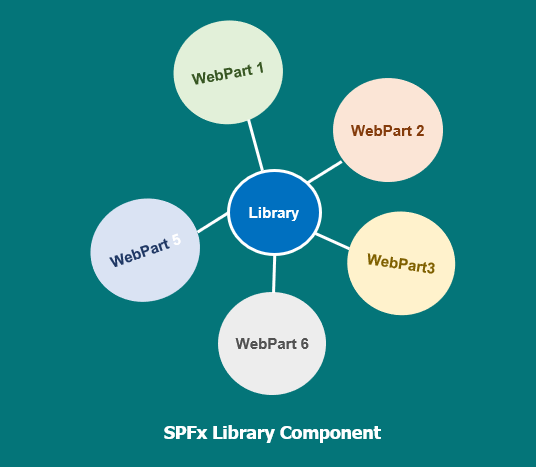
* You can only host one library component version at the time in a tenant.
* You can deploy and host library components in the tenant app catalog or the site app catalog.
* It's not supported to have other component types included in a solution, which contains library component.

You can reference library component in the SharePoint solution by defining the dependency in the **package.json** file. The bundling process detects this dependency and adds it as to the consuming component's manifest. This dependency will then be detected by the SharePoint Framework at runtime and load the library before loading the component's bundle.

**What is library component? When to use it ?**

When building web parts if we like to be able to extract common code to a separate library that can be shared amongst the different web parts.

Library component provides option to create shared code, which can be then used and referenced across all the components in the tenant.



**How to create a third-party SPFx library**

1. Create a new project directory in your favourite location

md custom-Library

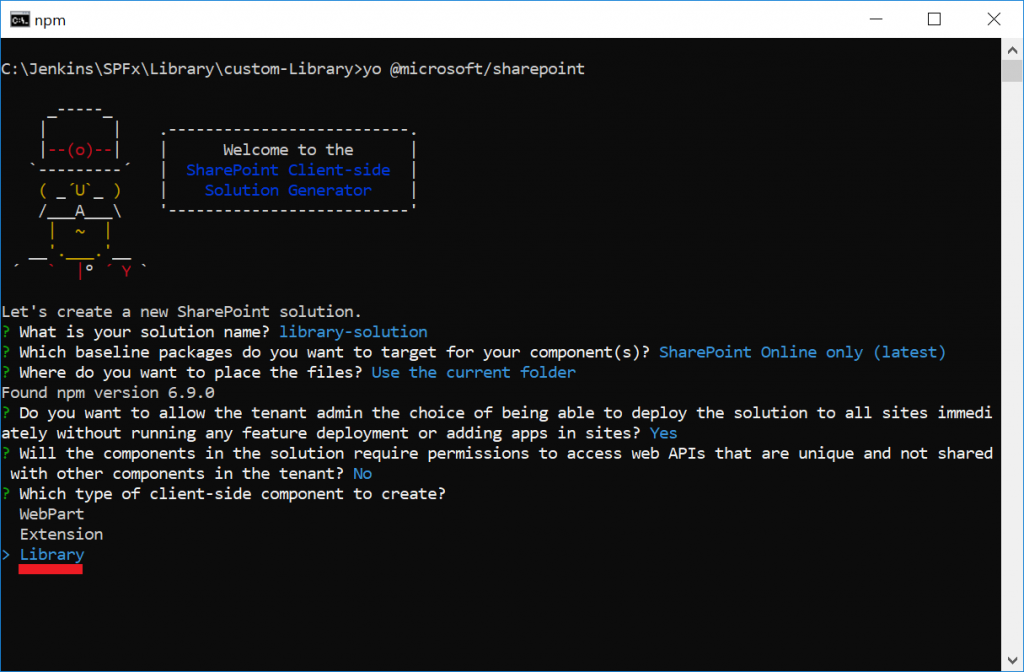
1. Go to the project directory

cd custom-Library

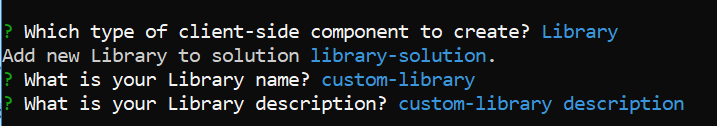
1. Create a new library by running the Yeoman SharePoint Generator

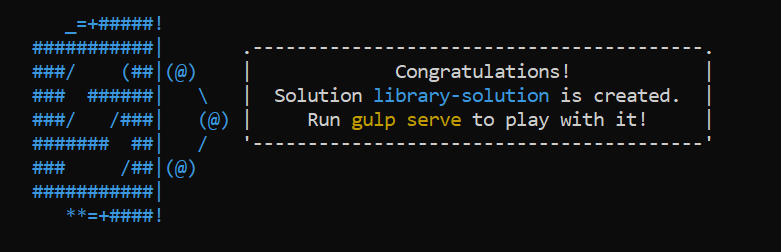
yo @microsoft/sharepoint

1. When prompted, enter the following values



1. Select **Library** as the client-side component type to be created.
2. The next set of prompts ask for specific information about your library:





Library Solution Successfully created

Type code . to open the project using visual studio code

1. Once the project is scaffolded, you will see the library created with an index.ts file containing an export from the  **CustomLibraryLibrary**created.
2. To add your method or classes, go to

src\libraries\CustomLibrary\CustomLibraryLibrary.ts

1. You will notice the default methos name()

export default class CustomLibraryLibrary {

public name(): string {

return 'CustomLibraryLibrary';

}

}

1. You can change the functionality as per your LOB, here I am return the current time for sample, replace name method as follows:

public getCurrentTime(): string {

return 'The current time as returned from the custom library is ' + new Date().toTimeString();

}

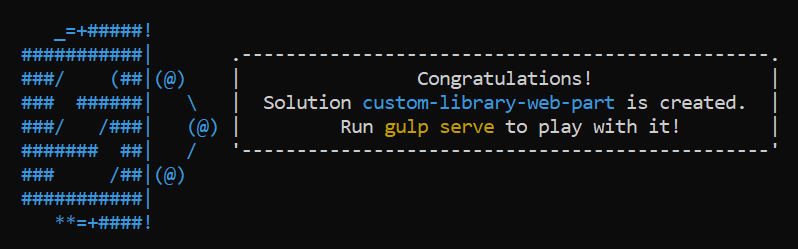
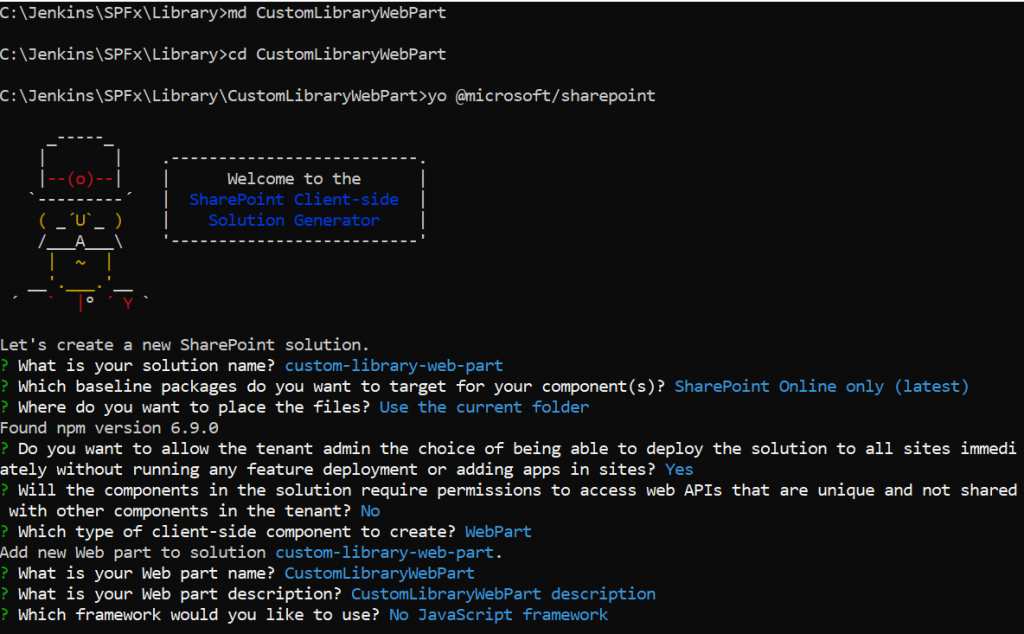
1. Run gulp on the command prompt to see everything builds fine
2. Now you are ready to use the Library to your SPFx webParts.

## How to consume a SPFx library for local testing

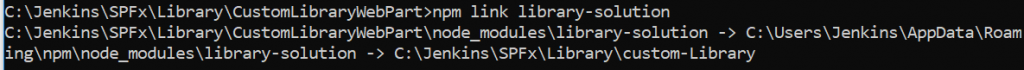
* Run **npm link** from the root directory of library solution. In this case it would be from the **custom-Library** folder.
* This will create a local npm link to the library with the name which is provided in the **package.json**

## Create a SPFx Webpart

* Create a web part project in a separate project folder, so not in the library project folder structure, if you are new to SPFx webpart, then following the instructions from [here](https://docs.microsoft.com/en-us/sharepoint/dev/spfx/web-parts/get-started/build-a-hello-world-web-part#to-create-a-new-web-part-project). Name your web part ‘**CustomLibraryWebPart**‘



* To refer the library to your project, From the root of the new web part folder, run the command **npm link library-solution**
* This will create a symbolic link to that locally built library into the web part and will make it available to your web part. refer below screenshot



* Open the web part solution using visual studio editor and navigate to **src\webparts\customLibraryWebPart\CustomLibraryWebPartWebPart.ts**
* Add an import to refer to your custom library

import \* as myLibrary from 'library-solution';

* Change the default render method and call the custom library method ‘**getCurrentTime()**‘
* Create an instance **const myInstance = new myLibrary.CustomLibraryLibrary();**
* Call the method**myInstance.getCurrentTime()**

public render(): void {

const myInstance = new myLibrary.CustomLibraryLibrary();

this.domElement.innerHTML = `

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

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

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

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

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

<p>${myInstance.getCurrentTime()}</p>

</div>

</div>

</div>

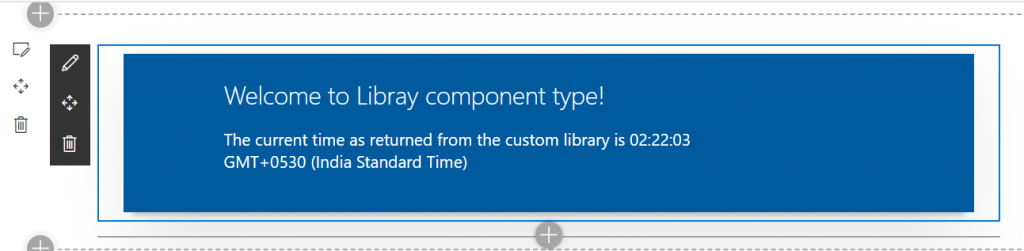
</div>`;

}

* Test your webpart

gulp serve

* Test your webpart by launching the local workbench and add the webpart to the page



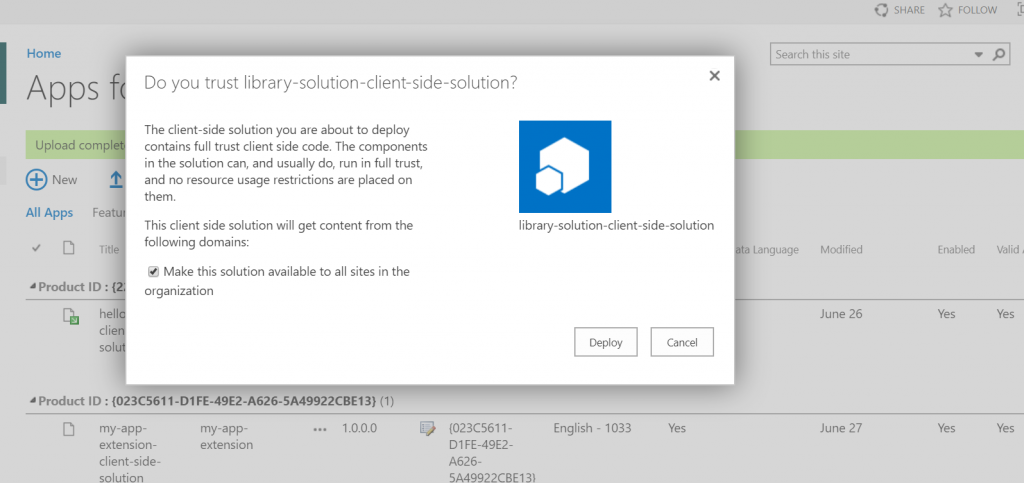
## Deploy a SPFx library component to tenant app Catalog

* This is a same steps as SPFx webpart deployment
* Navigate to the **custom-library** root folder and bundle and package the library solution

gulp bundle --ship

gulp package-solution --ship

* This will build any local changes made and package the solution into a sppkg file which is located in the sharepoint\solution folder
* Deploy this package in the tenant app catalog and make it tenant wide deployed by checking the **Make this solution available to all sites in the organization** option



* Now we deployed the Library solution

## Deploy webpart and consume library component from webpart

* Navigate to the web part solution folder. **CustomLibraryWebPart**
* Open the **package.json** file in the root of that folder, i.e.. **CustomLibraryWebPart**folder
* Add an entry to reflect the library entry and its version to the dependencies section (you can find this in the package.json file of the library solution your created earlier) as follows:

"dependencies": {

"library-solution": "0.0.1",

"@microsoft/sp-core-library": "1.9.0",

"@microsoft/sp-webpart-base": "1.9.0",

"@microsoft/sp-lodash-subset": "1.9.0",

"@microsoft/sp-office-ui-fabric-core": "1.9.0",

"@types/webpack-env": "1.13.1",

"@types/es6-promise": "0.0.33"

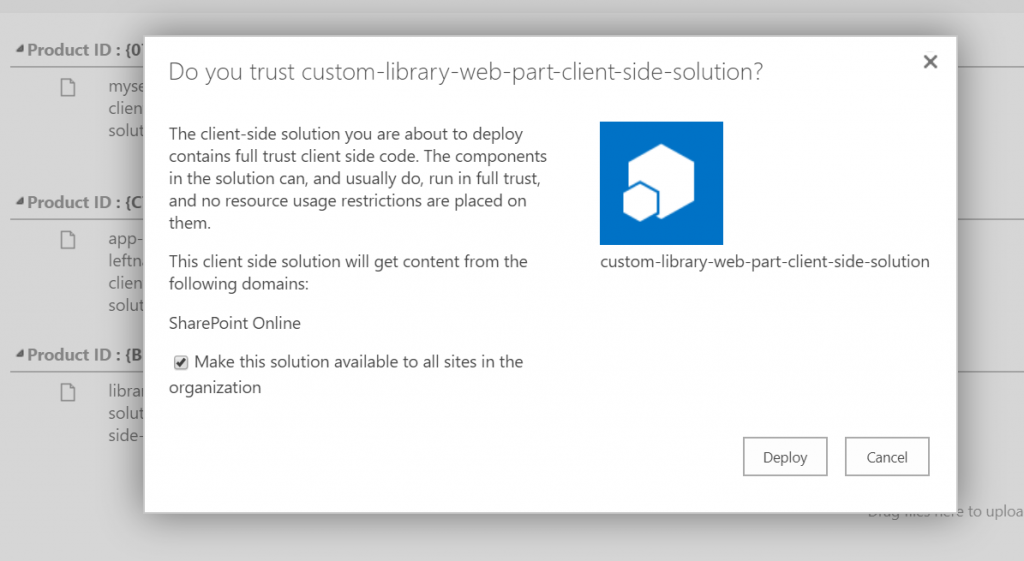
},

* Build the webpart to deploy

gulp bundle --ship

gulp package-solution --ship

* Deploy the web part solution to the tenant app Catalog.



* Add the newly added web part to a page and notice that the library is automatically made available to the web part the web part functions.
* Making any changes to the library and publishing the library to the app Catalog again will automatically update the web part without the need to rebuild/republish the web part.

