# Exercise 4: Creating a command set extension

## Task 1: Create your project

1. From the PowerShell command prompt, change to the C:/LabFiles/SharePoint directory by executing the following command: cd c:/LabFiles/SharePoint
2. Make a new directory for your SharePoint project files by executing the following command: md SPFxCommandSet
3. Navigate to the newly created SharePoint directory by executing the following command: cd SPFxCommandSet
4. Run the SharePoint Yeoman generator by executing the following command: yo @microsoft/sharepoint
5. Use the following to complete the prompt that is displayed:
   * **What is your solution name?**: SPFxCommandSet
   * **Which baseline packages do you want to target for your component(s)?**: SharePoint Online only (latest)
   * **Where do you want to place the files?**: Use the current folder
   * **Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without running any feature deployment or adding apps in sites?**: No
   * **Will the components in the solution require permissions to access web APIs that are unique and not shared with other components in the tenant?**: No
   * **Which type of client-side component to create?**: Extension
   * **What type of client-side extension to create?**: ListView Command Set
   * **What is your Command Set name?**: CommandSetDemo
   * **What is your Command Set description?**: CommandSetDemo description
6. After provisioning the folders required for the project, the generator will install all the dependency packages using NPM.
7. From the PowerShell command prompt execute the following command: gulp trust-dev-cert
8. Open the project in Visual Studio Code by executing the following command: code .

## Task 2: Observe the Code of your ListView Command Set

1. From Visual Studio Code, open the **CommandSetDemoCommandSet.ts** file in the \*\*src\* folder.
2. Notice that the base class for the ListView Command Set is imported from the **sp-listview-extensibility** package, which contains SharePoint Framework code required by the ListView Command Set.

* import { override } from '@microsoft/decorators';  
  import { Log } from '@microsoft/sp-core-library';  
  import {  
   BaseListViewCommandSet,  
   Command,  
   IListViewCommandSetListViewUpdatedParameters,  
   IListViewCommandSetExecuteEventParameters  
  } from '@microsoft/sp-listview-extensibility';  
  import { Dialog } from '@microsoft/sp-dialog';

The behavior for your custom buttons is contained in the **onListViewUpdated()** and **OnExecute()** methods.

The **onListViewUpdated()** event occurs separately for each command (for example, a menu item) whenever a change happens in the ListView, and the UI needs to be re-rendered. The event function parameter represents information about the command being rendered. The handler can use this information to customize the title or adjust the visibility, for example, if a command should only be shown when a certain number of items are selected in the list view. This is the default implementation.When using the method **tryGetCommand**, you get a Command object, which is a representation of the command that shows in the UI. You can modify its values, such as **title**, or **visible**, to modify the UI element. SPFx uses this information when re-rendering the commands. These objects keep the state from the last render, so if a command is set to **visible** = **false**, it remains invisible until it is set back to **visible** = **true**.typescript @override public onListViewUpdated(event: IListViewCommandSetListViewUpdatedParameters): void { const compareOneCommand: Command = this.tryGetCommand('COMMAND\_1'); if (compareOneCommand) { // This command should be hidden unless exactly one row is selected. compareOneCommand.visible = event.selectedRows.length === 1; } }

The **OnExecute()** method defines what happens when a command is executed (for example, the menu item is selected). In the default implementation, different messages are shown based on which button was selected.

@override  
public onExecute(event: IListViewCommandSetExecuteEventParameters): void {  
 switch (event.itemId) {  
 case 'COMMAND\_1':  
 Dialog.alert(`${this.properties.sampleTextOne}`);  
 break;  
 case 'COMMAND\_2':  
 Dialog.alert(`${this.properties.sampleTextTwo}`);  
 break;  
 default:  
 throw new Error('Unknown command');  
 }  
}

## Task 3: Test your extension

You cannot currently use the local Workbench to test SharePoint Framework Extensions. You’ll need to test and develop them directly against a live SharePoint Online site. You don’t have to deploy your customization to the app catalog to do this, which makes the debugging experience simple and efficient.Go to any SharePoint list in your SharePoint Online site by using the modern experience or create a new list. Copy the URL off the list to clipboard.Because our ListView Command Set is hosted from localhost and is running, you can use specific debug query parameters to execute the code in the list view.Open **serve.json** file from **config** folder. Update the **pageUrl** attributes to match a URL of the list where you want to test the solution. After edits your serve.json should look somewhat like:powershell { "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json", "port": 4321, "https": true, "serveConfigurations": { "default": { "pageUrl": "https://contoso.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx", "customActions": { "bf232d1d-279c-465e-a6e4-359cb4957377": { "location": "ClientSideExtension.ListViewCommandSet.CommandBar", "properties": { "sampleTextOne": "One item is selected in the list", "sampleTextTwo": "This command is always visible." } } } }, "helloWorld": { "pageUrl": "https://sppnp.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx", "customActions": { "bf232d1d-279c-465e-a6e4-359cb4957377": { "location": "ClientSideExtension.ListViewCommandSet.CommandBar", "properties": { "sampleTextOne": "One item is selected in the list", "sampleTextTwo": "This command is always visible." } } } } } }

1. From the Visual Studio Code ribbon, select **Terminal** > **New Terminal**.
2. Run the project by executing the following command: gulp serve
3. When prompted, select the **Load debug scripts** button.
4. Stop the local web server by pressing CTRL+C in the console/terminal window.

## Review

In this exercise, you learned how to create and deploy a command set extension. Command set extensions are used for adding additional commands into the command bar at the top of a page.