PowerApps Component Framework

Hands-on Lab

Released:

Terms of Use

© 2019 We Speak You Learn, LLC. All rights reserved. Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of We Speak You Learn, LLC.

This document reflects current views and assumptions as of the date of development and is subject to change. Actual and future results and trends may differ materially from any forward-looking statements. We Speak You Learn, LLC. assumes no responsibility for errors or omissions in the materials.

THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT.

This document reflects current views and assumptions as of the date of development and is subject to change. Actual and future results and trends may differ materially from any forward-looking statements. We Speak You Learn, LLC. assumes no responsibility for errors or omissions in the materials. THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. “About the Authors” is an important item as it establishes the credibility of the text by explaining how much experience the author team has in the field. Add as many “About the Authors” as needed from the Quick Parts Gallery. Change “Author” to a different title, such as “Subject Matter Expert” if that is more appropriate. Include SMEs who assisted with the development of the course. Add this to every Module.

Table of Contents

[Lab Overview 1](#_Toc22046378)

[Learning Objectives 2](#_Toc22046379)

[Exercise 1: Create PCF Control 3](#_Toc22046380)

[Task 1: Install Microsoft PowerApps CLI and Prerequisites 3](#_Toc22046381)

[Task 2: Setup Components Project 3](#_Toc22046382)

[Task 3: Open the CLI in VS Code and Run 11](#_Toc22046383)

[Task 4: Deploy 13](#_Toc22046384)

[Task 5: Use the PCF Control 15](#_Toc22046385)

## Lab Overview

In this hands-on lab you will be implementing a PowerApps Component Framework component to filter values on an option set.

Several paragraphs – or longer – describing an overview of the lab including a description of the lab and why certain topics are covered. This is also called the ‘LAB ABSTRACT’ that will be used for hand-off to conferences as part of the content hand-off process

## Learning Objectives

* Initialization of a PCF component using the CLI
* How to implement the component logic
* How to use a framework like React and Office UI Fabric with PCF
* How to start the PCF test harness
* How to use the CLI to push the component to a CDS Environment
* How to configure a model-driven form to use the PCF component

# Exercise 1: Create a PowerApps Component

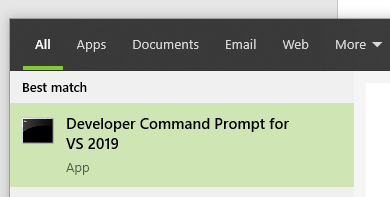
### Task 1: Install Microsoft PowerApps CLI and Prerequisites

Note: If you already have installed the PowerApps CLI, please issue a pac install latest command and skip to Task 2

1. Navigate to <https://docs.microsoft.com/en-us/powerapps/developer/component-framework/get-powerapps-cli> and follow the instructions. This lab is documented using Visual Studio Code for the editor

### Task 2: Setup Components Project

1. Start **Developer Command Prompt**.



1. Run the command shown below to change directory. Replace **[Computer User Name]** with your OS user name – or choose another starting location to build the project

cd C:\Users\[Computer User Name]\Documents

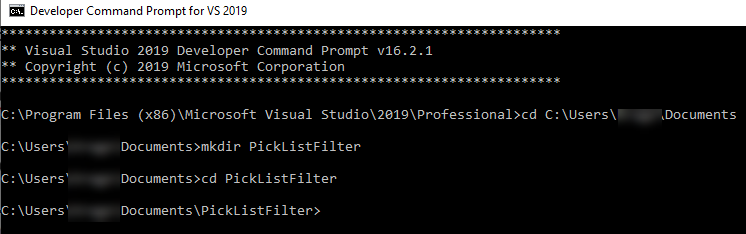
1. Create a new folder with the name PickListFilter.

mkdir PickListFilter

1. Change directory to the folder you just created.

cd PickListFilter

1. Your command prompt should now look like the image below.

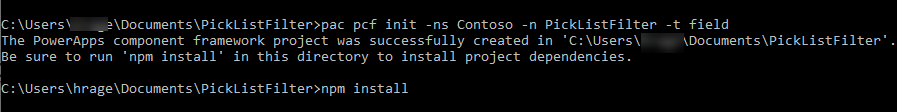


1. Initialize the component. This command will create a set of files that will implement a field component. You will customize these files as per your specific component as we continue.

pac pcf init -ns Contoso -n PickListFilter -t field

1. Install dependencies by running **npm install** command in the Terminal and wait for the dependencies to be installed.

npm install



1. Add React and Office UI Fabric to your project and wait for the installation to complete.

npm install react react-dom office-ui-fabric-react -save

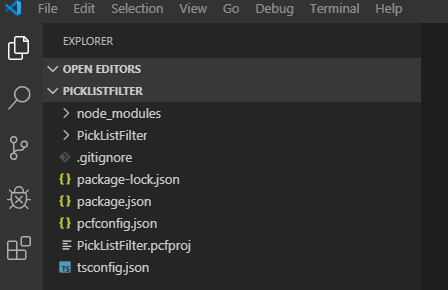
The following is some details about the three packages you just installed

|  |  |
| --- | --- |
| **Package** | **Details** |
| react | A JavaScript library for building user interfaces |
| react-dom | This package serves as the entry point to the DOM and server renderers for React. It is intended to be paired with the generic React package, which is shipped as react |
| office-ui-fabric | A responsive, mobile-first collection of styles and tools designed to make it quick and simple for you to create web experiences using the Office Design Language |

1. Open the project in **Visual Studio Code**. The “.” will cause the current folder to be the focus

code .

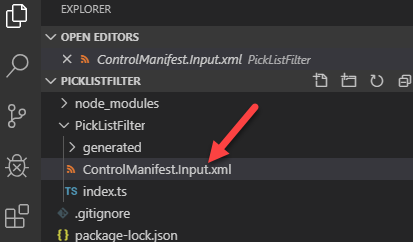
1. Your project structure should look like the image below.



# Exercise 2: Implement the component

### Task 1: Update the component properties

1. Expand the **PicklListFilter** folder and click to open the **ControlManifest.Input.xml** file.



1. Locate the **Property** element and replace it with the xml fragment below.

<property name="picklistField" display-name-key="Picklist Field" description-key="Picklist Field to filter" of-type="OptionSet" usage="bound" required="true" />

    <property name="filtering" display-name-key="Picklist Field" description-key="Picklist Field to filter" of-type="SingleLine.Text" usage="input" required="true" />

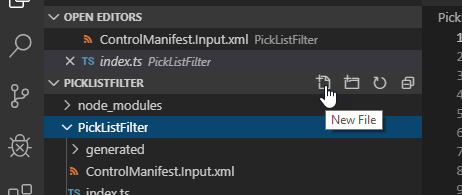


The following is a description of each property

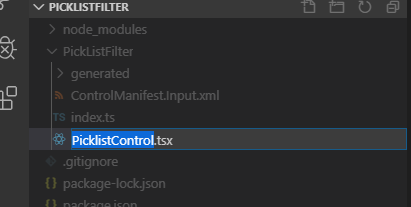
|  |  |
| --- | --- |
| Property | Description |
| picklistField | This will be bound to the Option Set field that you want the component to provide a visualization for |
| filtering | This will bind to a static value that will contain a comma separated list of option set values that the control should include in the dropdown list rendered by the component |

### Task 2: Setup the React Control

1. Select the **PickListFilter** folder and click **+ New File**.



1. Name the new file **PicklistControl.tsx**. This file is going to contain the React control that will render the visual for our PCF



1. Open the **PickListControl** file you just created and add the below imports. The imports allow us to reference items from both the React and Office UI Fabric namespaces.

import \* as React from 'react';

import { Dropdown, IDropdownOption } from "office-ui-fabric-react/lib/Dropdown";

1. Add the following code bellow the import statements.

export interface IControlProps {

    value: number | undefined,

    options: IDropdownOption[],

    onChange: (newValue: number | undefined) => void

}

1. Add the picklist control export below the interface you just added. This class has the mark up for the Dropdown control React will render in the app.

export class PicklistControl extends React.Component<IControlProps> {

    constructor(props: IControlProps, context: any) {

        super(props);

    }

    private \_onChange = (event: React.FormEvent<HTMLDivElement>, item: IDropdownOption | undefined): void => {

        var selectedKey = item && item.key as number;

        this.props.onChange(selectedKey);

    };

    render() {

        return (

            <Dropdown

                selectedKey={this.props.value}

                onChange={this.\_onChange}

                placeholder="--Select--"

                options={this.props.options}

                styles={{ dropdown: { width: "100%" } }}

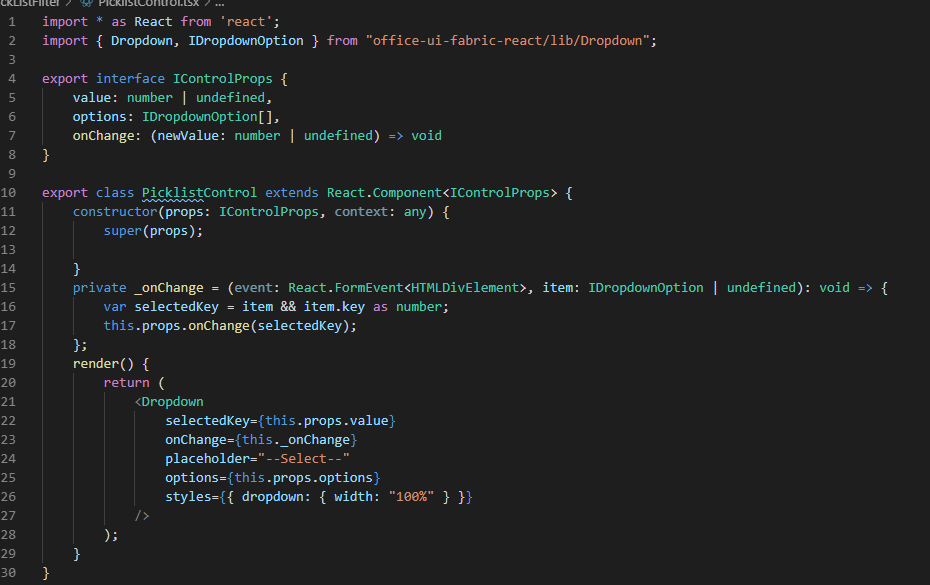
            />

        );

    }

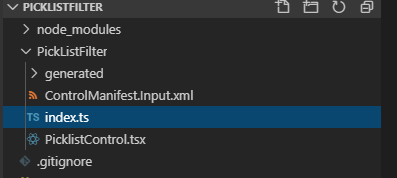
}

1. Your **PicklistControl.tsx** file should now look like the image below.



### Task 3: Modify Index.ts to render the control

1. Click to open the **index.ts** file.



1. Add the below **imports** to **index.ts** after the existing import.

import \* as React from "react";

import \* as ReactDOM from "react-dom";

import { PicklistControl } from "./PicklistControl";

import { IDropdownOption } from "office-ui-fabric-react/lib/Dropdown";

import { initializeIcons } from '@uifabric/icons';



1. Add the below constant above the class and below the import statements. This simply gives us a constant to reference instead of a hard coded magic string.

const picklistFieldName = "picklistField";

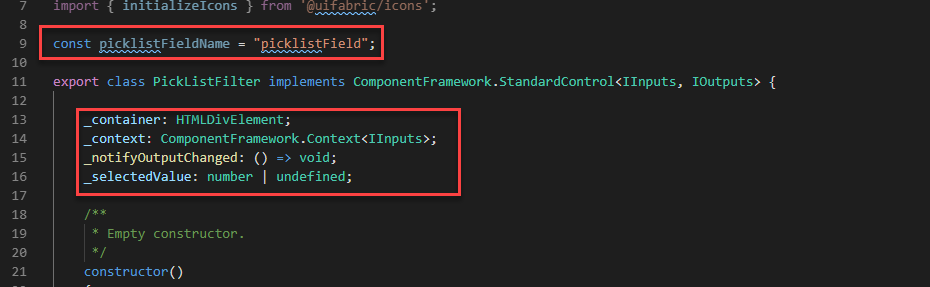
1. Add the below properties inside the class and before the constructor. These are class scoped variables.

    \_container: HTMLDivElement;

    \_context: ComponentFramework.Context<IInputs>;

    \_notifyOutputChanged: () => void;

    \_selectedValue: number | undefined;



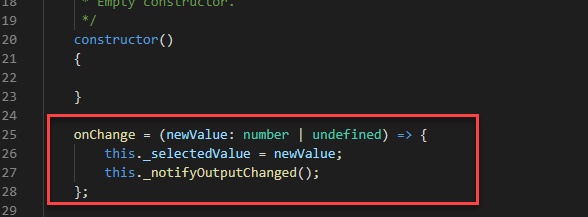
1. Add **onChange** function to the **index.ts** file just before the last }

    onChange = (newValue: number | undefined) => {

        this.\_selectedValue = newValue;

        this.\_notifyOutputChanged();

    };



1. Add the function that will render the control to the **index.ts** file. This function will be called to add the Dropdown control to the ReactDOM.

    private renderControl(context: ComponentFramework.Context<IInputs>) {

        // Get options metadata

        var metadata = context.parameters.picklistField.attributes;

        if (metadata && context.parameters.filtering.raw) {

            // Get the configuration input parameter - comma separated list of optionset values

            var filterInput = context.parameters.filtering.raw.

                split(",").map((a) => { return parseInt(a) });

            // Create filtered options provided by the metadata

            // Do this with every render in case they change

            var options = metadata.Options

                .filter((o) => filterInput.includes(o.Value))

                .map((v) => {

                    return {

                        key: v.Value,

                        text: v.Label

                    } as IDropdownOption

                });

            ReactDOM.render(

                React.createElement(PicklistControl, {

                    value: this.\_selectedValue,

                    options: options,

                    onChange: this.onChange

                }),

                this.\_container

            );

        }

    }



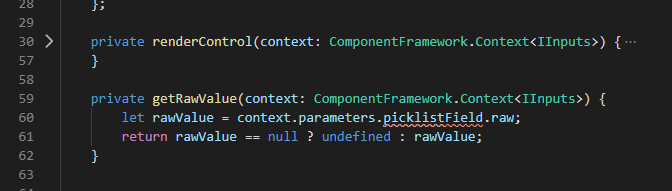
1. Add the getRawValue function below to the index.ts file. This will get the current value of the input parameter that was bound in the app. Note: until you build you will have an error indication under the picklistField property name.

private getRawValue(context: ComponentFramework.Context<IInputs>) {

        let rawValue = context.parameters.picklistField.raw;

        return rawValue == null ? undefined : rawValue;

    }



1. Add the snippet below inside the **init** function. The init function is used to initialize the control instance. You could also kick off remote server calls and other initialization actions here. If this was a Dataset component, you would do this in updateView

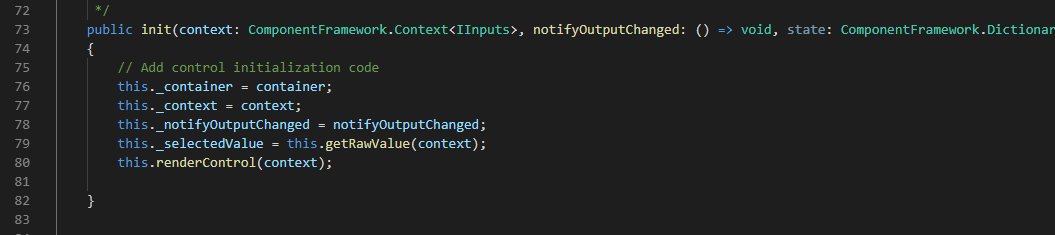
        this.\_container = container;

        this.\_context = context;

        this.\_notifyOutputChanged = notifyOutputChanged;

        this.\_selectedValue = this.getRawValue(context);

        this.renderControl(context);



1. Locate the **updateView** function and add the snippet below. The updateView function is called when any value in the property bag has changed. This includes field values, data-sets, global values such as container height and width, offline status, control metadata values such as label, visible, etc.

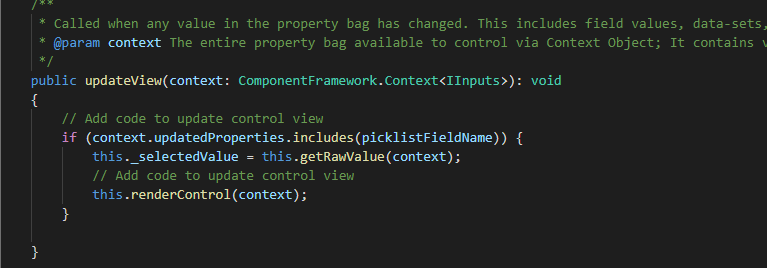
if (context.updatedProperties.includes(picklistFieldName)) {

            this.\_selectedValue = this.getRawValue(context);

            // Add code to update control view

            this.renderControl(context);

        }



1. Locate **getOutputs** and replace the **return {}** line with the snippet below. The getOutputs function is called by the framework prior to a control receiving new data

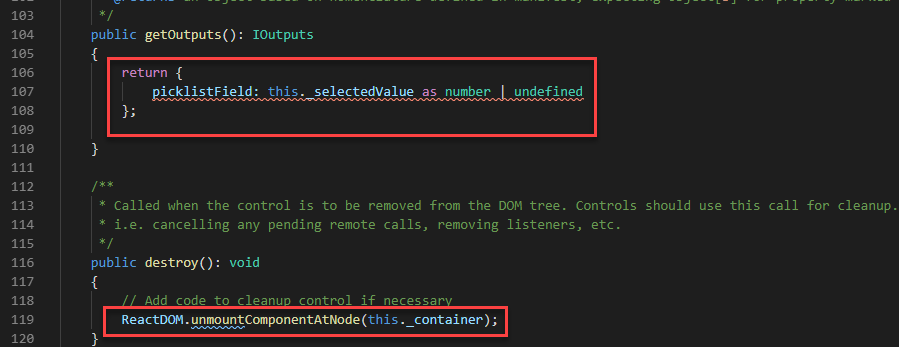
return {

            picklistField: this.\_selectedValue as number | undefined

        };

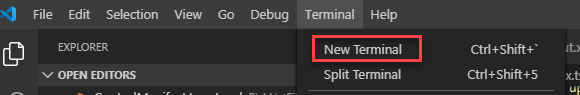
1. Locate the **destroy** function and add the snippet below.

ReactDOM.unmountComponentAtNode(this.\_container);



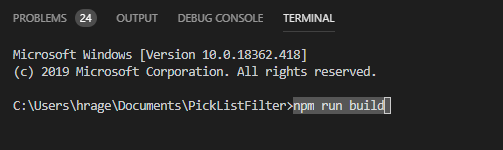
### Task 4: Open the CLI in VS Code and Run

1. Click **Terminal** and select **New Terminal**.

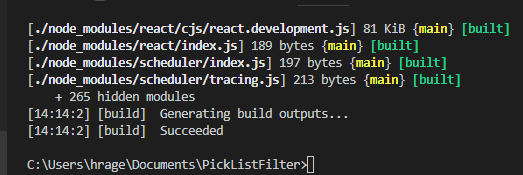
****

1. Run the **build** command.

npm run build



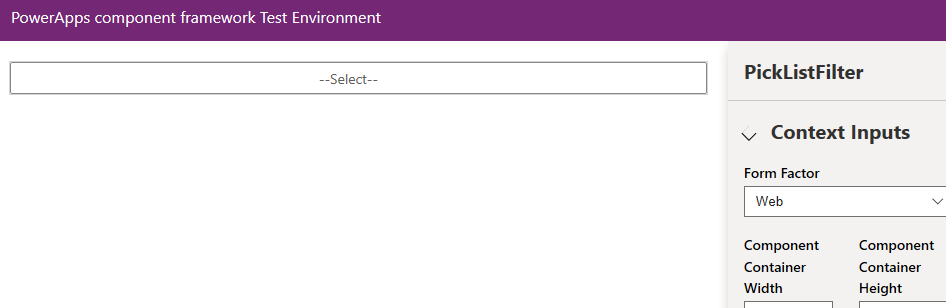
1. The build should succeed. Make sure you resolve any problems before you proceed.



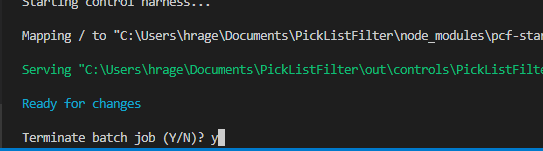
1. Run the **start** command.

npm start

1. The test environment should render the control, but you can’t interact with it. The reason is the control you are building depends on binding to live data of a record. Some components you will build you will be able to use this to quickly test the component.



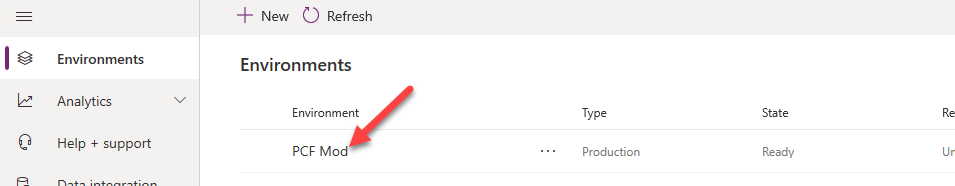
1. Close the Test Environment browser window or tab.
2. Go back to the terminal in Visual Studio code.
3. Click on the **Terminal** and press the **[CONTROL]** key and **c**
4. Type **y** and enter **[ENTER]**.



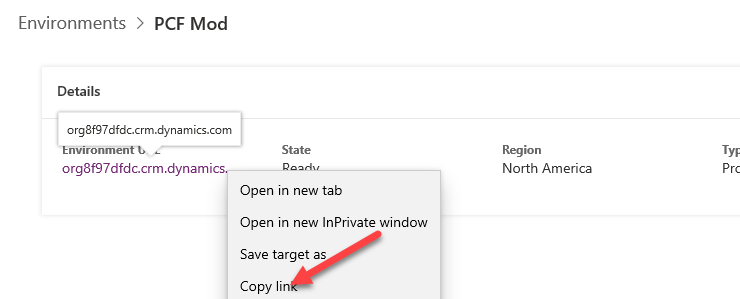
# Exercise 3: Deploy the component

### Task 1: Setup the CLI to allow deploy

1. Navigate to make <https://admin.powerplatform.microsoft.com/> select Environments and click to open the environment you are deploying the PCF control to.



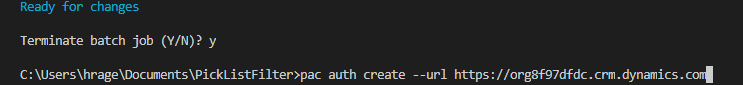
1. Right click on the **Environment URL** and select **Copy Link**.



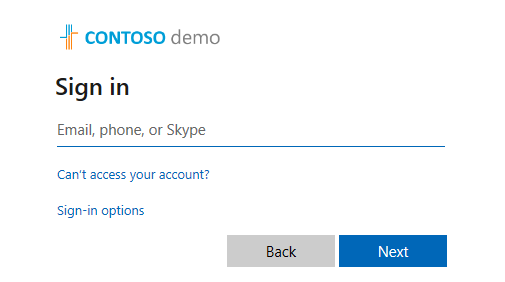
1. Go back to the Visual Studio terminal and type the command below.

pac auth create --url

1. Add the **URL** you copied in front of **url**.
2. Your command should now look like the image below. press [ENTER]



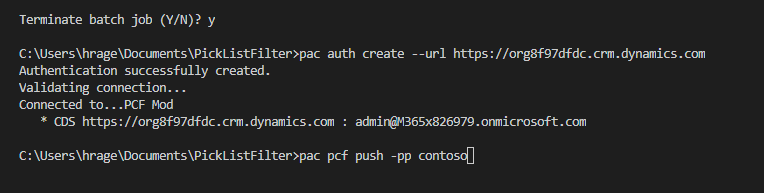
1. Login with your admin user credentials.

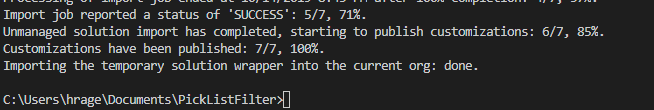


### Task 2: Push the component to the environment

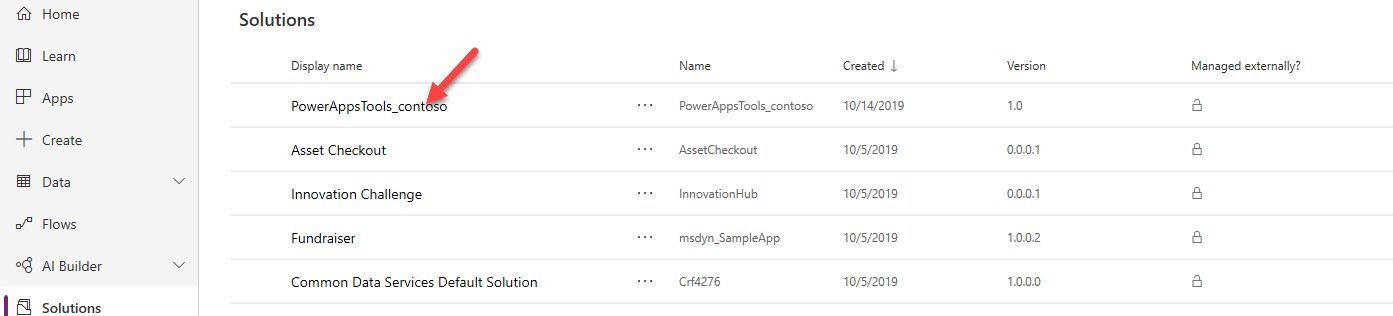
1. Run the command below and wait for the publishing to complete. This command will package the component in a temporary solution with a publisher prefix as specified in the “pp” parameter

pac pcf push -pp contoso

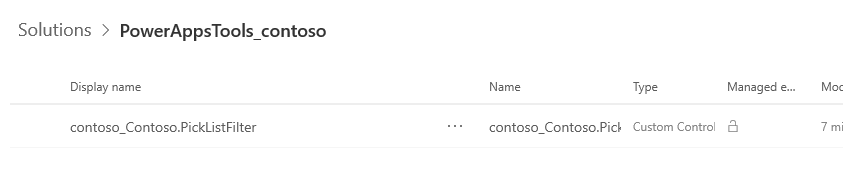




1. Navigate to <https://make.powerapps.com/> and select the correct environment.
2. Select **Solutions**.
3. You should see the solution **PowerApps** **Tools** created. Click to open the solution.



1. You should see the Custom Control you created.

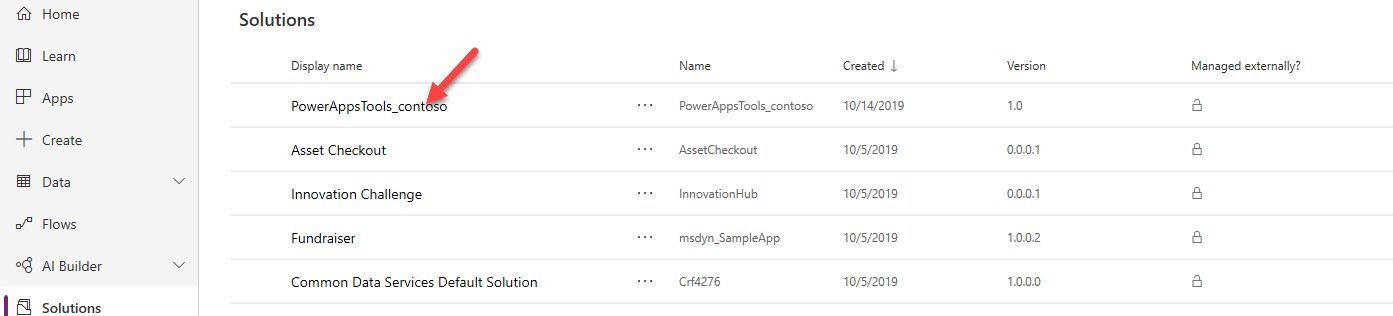


# Exercise 4: Use the component

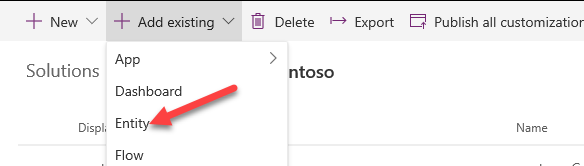
### Task 1: Configure a form to use the component

1. Navigate to <https://make.powerapps.com/> and select the correct environment.
2. Select **Solutions**.
3. You should see the solution **PowerApps** **Tools** created. Click to open the solution.

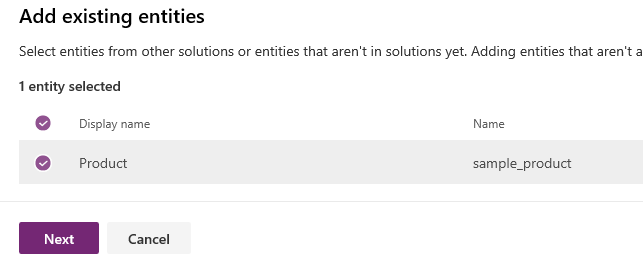
Note: While we are using the temporary solution to configure testing the component – this is only done for the lab, generally you don’t want to modify this solution and should do any customization in your own solution.



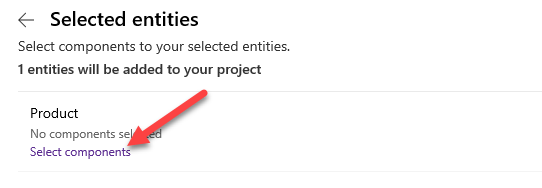
1. Click **+ Add Existing** and select **Entity**.



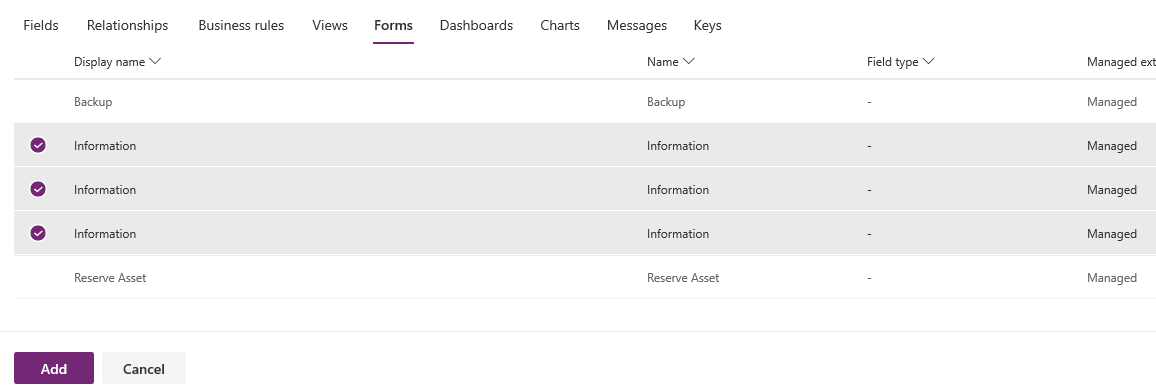
1. Select **Product** and click **Next**.



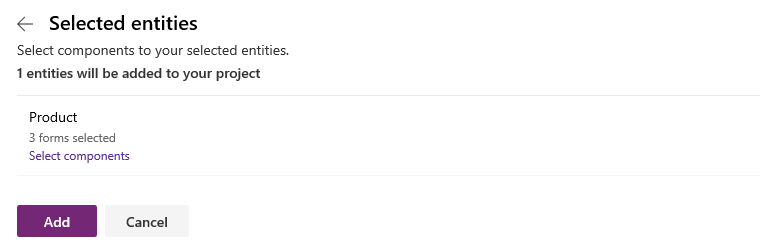
1. Click **Select Components**.



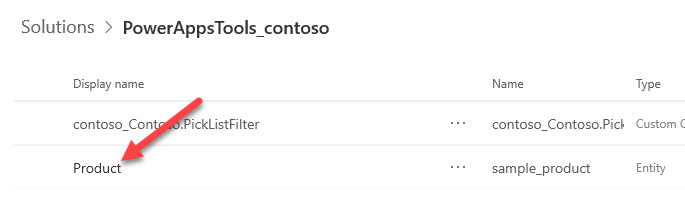
1. Select the **Forms** tab, select all the **Information** forms and click **Add**. Note: we really just want the main form, but currently there isn’t a way to tell which form that is of the 3



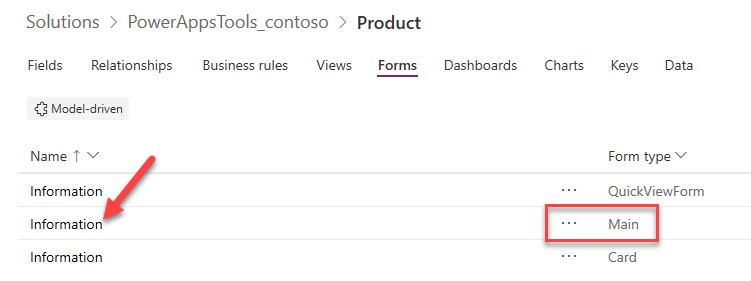
1. Click **Add** again.



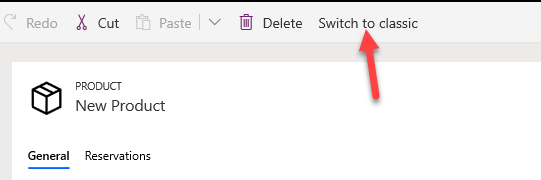
1. Click to open the **Product** entity you just added to your solution.



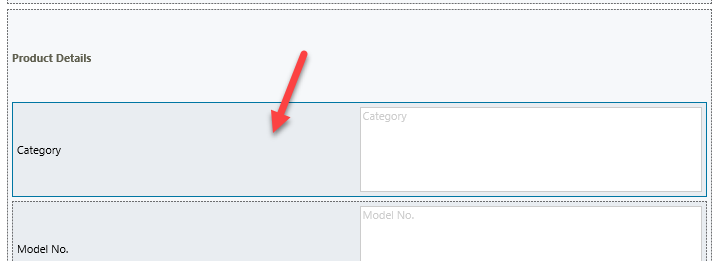
1. Select the **Forms** tab and click to open the **Main** form.



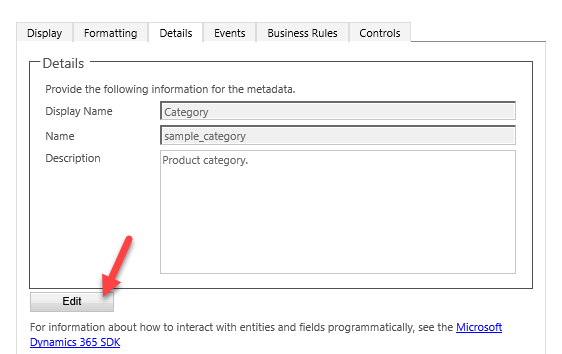
1. Click **Switch to Classic**. Note: We are switching to classic because the new editor doesn’t let you configure custom components yet.



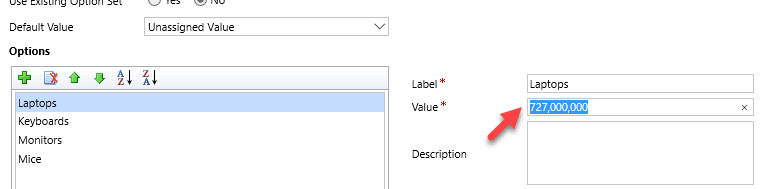
1. Double click the **Category** field. Note: Make sure you select Category and not Category ID



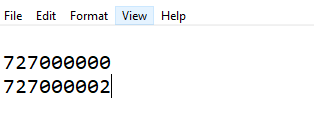
1. Select the **Details** tab and click **Edit**.



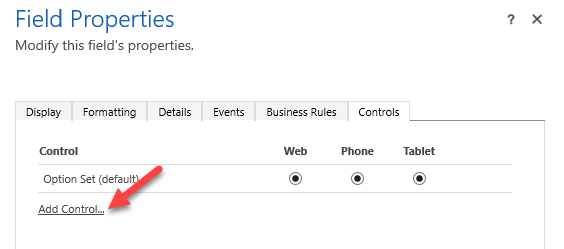
1. Scroll down to the **Options** and select **Laptops**. Notice that there are a total of 4 items in this Option Set. We will be filtering it down to 2 items with our PCF component.
2. Copy the **Value** and keep it in a notepad.



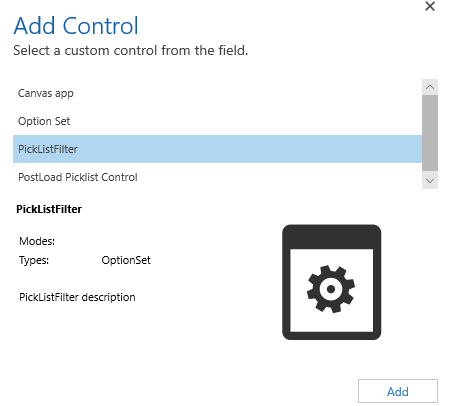
1. Select **Monitors** and copy the **Value** to a notepad.
2. You should now have two values in your notepad. Remove the commas.



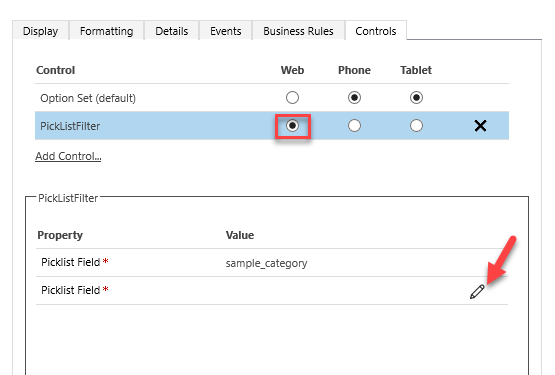
1. Go back to the field editor and close it.
2. Select the **Controls** tab and click **Add Control**.



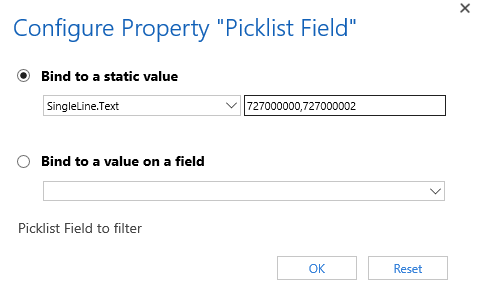
1. Select the **PickListFilter** you created and click **Add**.



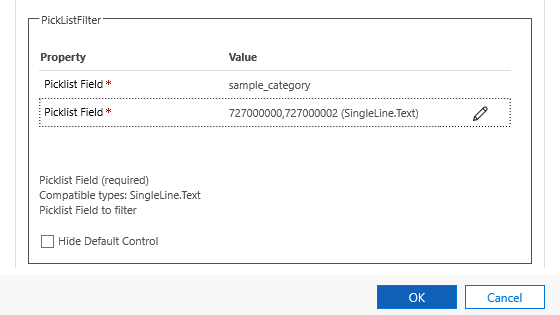
1. Select **Web** and click **Edit. You can also select Phone and Tablet if you like.**  Notice since you started from the Category Field, the Picklist Field is already bound to that for you.



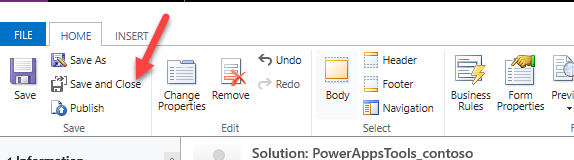
1. Select **Bind to Static Value**
2. Paste the two values you copied separated by a comma and click **OK**.



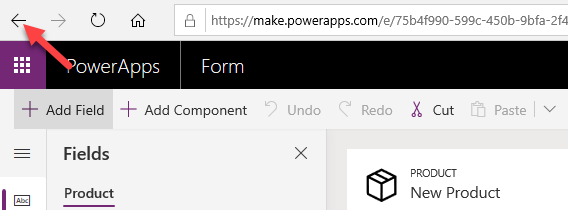
1. Click **OK** again.



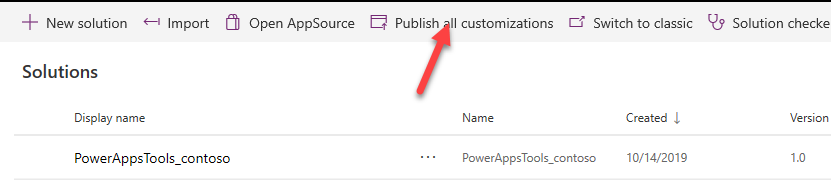
1. Click **Save and Close**.



1. Click on the browser back button.

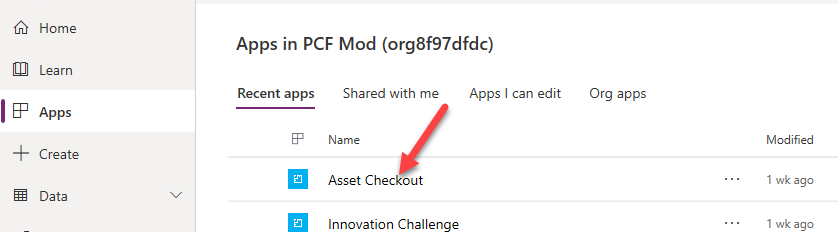


1. Select **Solutions** and click **Publish All Customizations**.

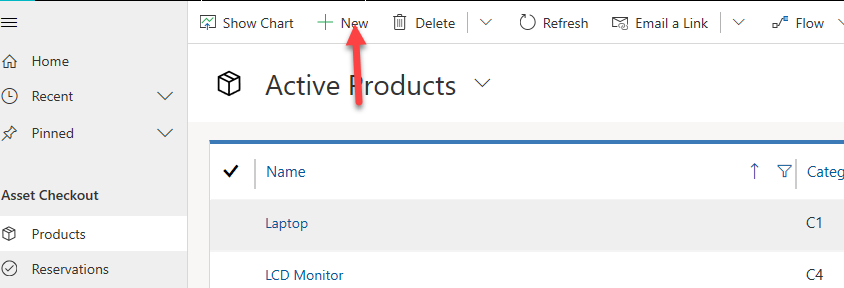


### Task 2: Test the component

1. Select **Apps** and click to open the **Asset Checkout** application.



1. Select **Products** and click **+ New**.



1. The **Category** option-set should now have only the two options you provided.

