

Power Component Framework (PCF)



Cognitive Convergence

<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com

Cognitive Convergence is Subject Matter Expert in Office 365, Dynamics 365, SharePoint, Project Server, Power Platform: Power Apps-Power BI-Power Automate-Power Virtual Agents. Our Microsoft Dynamics 365 Consulting, Development, Customization, Integration services and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability and providing best customer service.

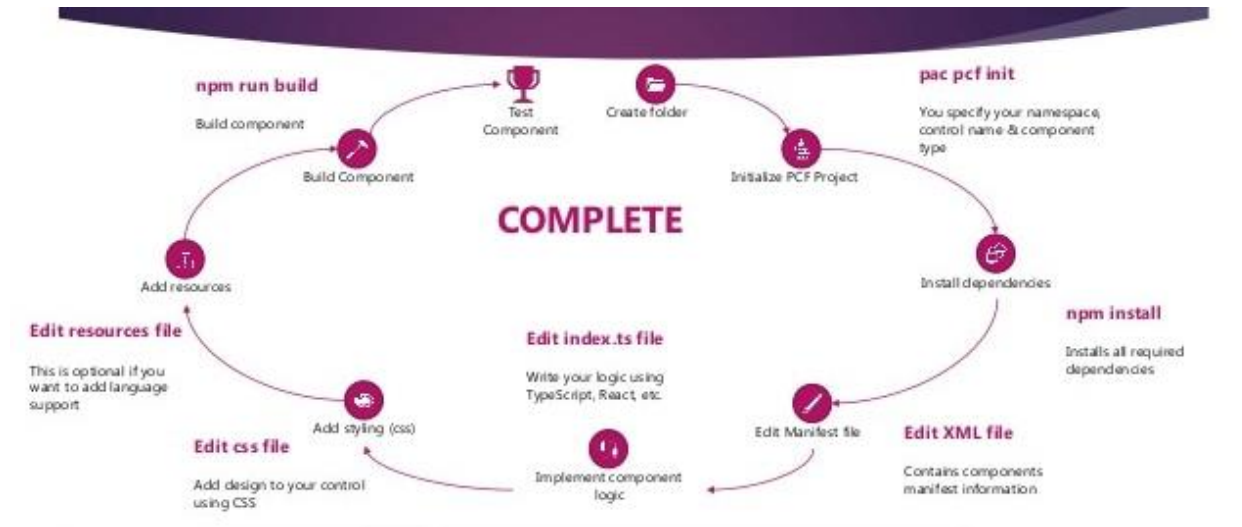
CONTENTS

Code Components	3
<i>Manifest</i>	3
<i>Component implementation</i>	4
<i>Resources</i>	4
Code Components for canvas Apps	5
<i>Prerequisites</i>	5
<i>Enable PCF Feature</i>	5
<i>Add components to canvas app</i>	6
Code Components for Model Driven Apps	7
<i>Need of Code Components</i>	7
PowerApps Component Framework (PCF).....	10
PCF Development Architecture.....	13
<i>Creating a new Code Component</i>	13
<i>Building Code Component</i>	14
<i>Debugging Code Component</i>	14
<i>Packaging Code Component</i>	14
Custom PCF Project	14
<i>Prerequisites</i>	14
<i>Manifest control implementation</i>	15

<i>Manifest property</i>	16
<i>Manifest resource</i>	17
<i>Implementation of code logic</i>	17
<i>Linear input control PCF added in Model Driven App</i>	18
<i>Web API</i>	18
<i>DataSet</i>	18
<i>Context</i>	20
<i>Filtering</i>	22
<i>Linking</i>	22
<i>EntityRecord</i>	23
<i>EntityReference</i>	23
<i>Challenges</i>	24
<i>Conclusion</i>	25

CODE COMPONENTS

Code components are a type of solution components, which means they can be included in a solution file and installed in different environments.



Code components can be added by including them in a solution and then import it into Dataverse. Once the components are in Dataverse, system administrators and system customizers can configure fields, sub grids, views, and dashboard sub grids to use in place of default components. Code components can be added to both **model-driven and canvas apps**.

Code components consist of three elements:

1. Manifest
2. Component Implementation
3. Resources

Manifest

Manifest is the metadata file that defines a component. It is an XML document that describes:

- The name of the component.
- The kind of data that can be configured, either a field or a data-set.
- Any properties that can be configured in the application when the component is added.
- A list of resource files that the component needs.
- The name of the Typescript function in the component implementation library that returns an object that applies the required component interface.

When a user configures a code component, the data in the manifest file filters out the available components so that only valid components for the context are available for configuration. The properties defined in the manifest file for a component are rendered as configuration fields so that the user configuring the component can specify the values. These property values are then available to the component at runtime.

Component implementation

Implementing the component is one of the key steps when you are developing code components using Power Apps component framework. Developers can implement a component using TypeScript. Each code component must have a `index.ts` file that includes the definition of a function, which returns an object that implements the methods described in the code component interface. This file is autogenerated via CLI tooling with main stub methods.

The object implements the following methods:

- `init` (Required)
- `updateView` (Required)
- `getOutputs` (Optional)
- `destroy` (Required)

These methods control the lifecycle of the code component.

Resources

Each code component should have a resource file to construct its visualization. You can define a resource file in the manifest. The resource node in the manifest file refers to the resources that the component requires to implement its visualization



CODE COMPONENTS FOR CANVAS APPS

We can use Power Apps component framework to create code components that can be used in their canvas apps. Professional developers can use Power Apps component framework to create, import, and add code components to canvas apps by using Microsoft Power Platform CLI. Certain APIs might not be available in canvas apps.

When you open a canvas app that contains code components in Power Apps Studio, a warning message about potentially unsafe code appears. Code components in the Power Apps Studio environment have access to security tokens; hence only components from trusted sources should be opened. Administrators and system customizers should review and validate all code components before importing those components in an environment and making them available for makers to use in their apps. The Default publisher is shown when you import code components by using an unmanaged solution or when you have used `pac pcf push` to install your code component.

Prerequisites

- A Power Apps license is required.
- System administrator privileges are required to enable the Power Apps component framework feature in the environment.

Enable PCF Feature

To add code components to an app, we enable the Power Apps component framework feature in each environment. By default, the Power Apps component feature is enabled for model-driven apps.

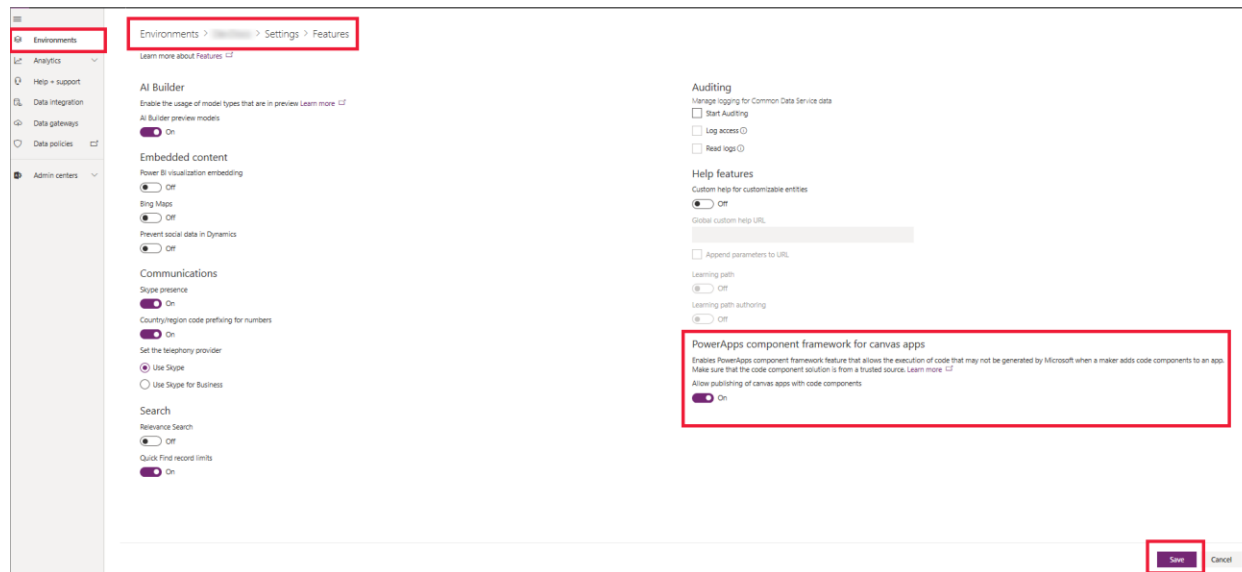
Our support for code components for Canvas and Model-driven apps will help you to optimize your business performance.

Cognitive Convergence

<http://www.cognitiveconvergence.com>

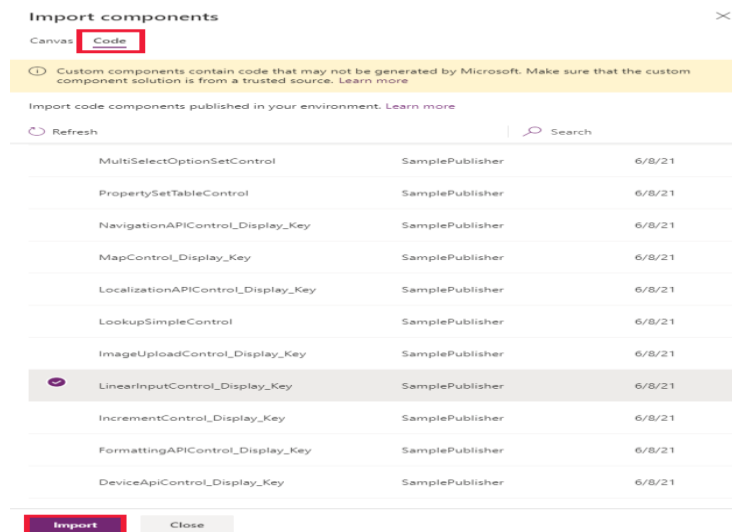
+1 4242530744

shahzad@cognitiveconvergence.com



Add components to canvas app

The code component, when created is added in the canvas app.

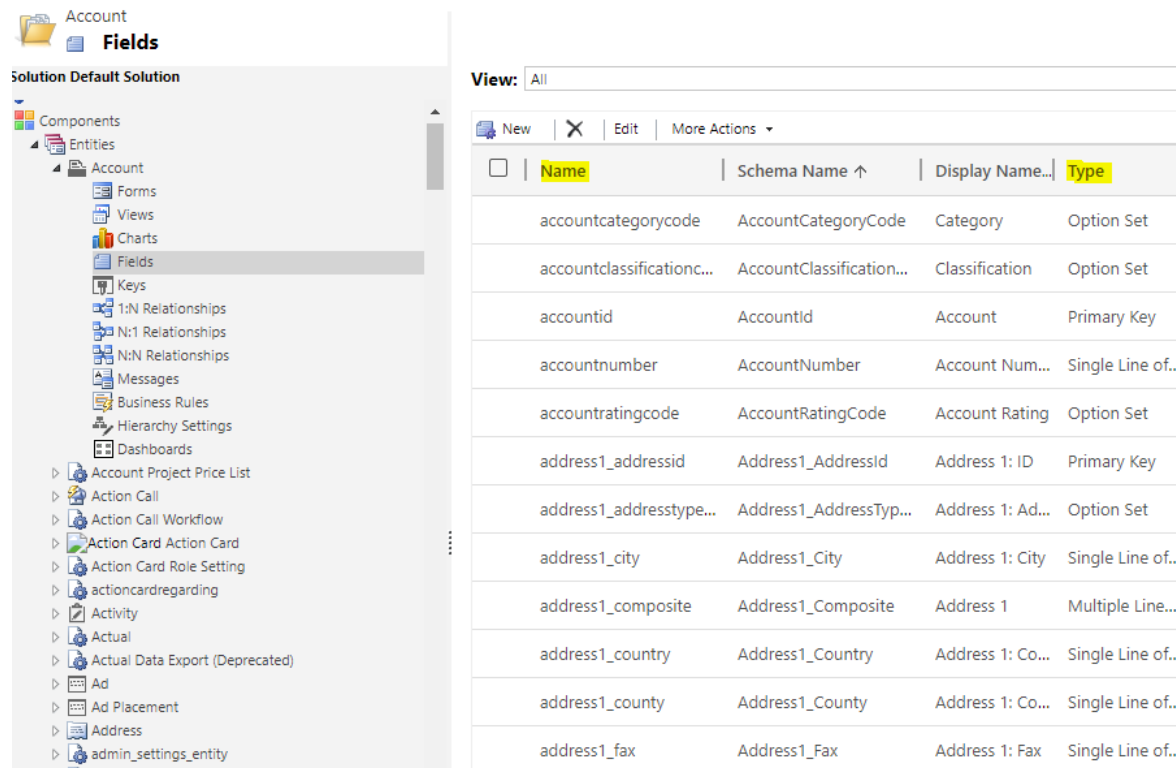


CODE COMPONENTS FOR MODEL DRIVEN APPS

Power Apps component framework gives developers the ability to extend the visualizations in model-driven apps. Professional developers can create, debug, import, and add code components to model-driven apps using Microsoft Power Platform CLI. We can add code components to columns, grids, and sub grids in model-driven apps.

Need of Code Components

In PowerApps and Dynamics 365, fields have types, such as Single Line of Text, Whole Number, Currency etc. For example, if we open our Account entity, we can see there are many fields, and these fields have different types:



Name	Schema Name	Display Name	Type
accountcategorycode	AccountCategoryCode	Category	Option Set
accountclassificationc...	AccountClassification...	Classification	Option Set
accountid	AccountId	Account	Primary Key
accountnumber	AccountNumber	Account Num...	Single Line of...
accountratingcode	AccountRatingCode	Account Rating	Option Set
address1_addressid	Address1_AddressId	Address 1: ID	Primary Key
address1_addresstype...	Address1_AddressTyp...	Address 1: Ad...	Option Set
address1_city	Address1_City	Address 1: City	Single Line of...
address1_composite	Address1_Composite	Address 1	Multiple Line...
address1_country	Address1_Country	Address 1: Co...	Single Line of...
address1_county	Address1_County	Address 1: Co...	Single Line of...
address1_fax	Address1_Fax	Address 1: Fax	Single Line of...

We can open a field and see the field type, e.g. Account Number is a Single Line of Text field:

Field

Working on solution: Default Solution

Account Number of Account

Common

- Information
- Business Rules

General

Schema

Display Name * Account Number Field Requirement * Optional

Name * accountnumber Searchable Yes

Field Security ☐ Enable ☒ Disable

⚠ Enabling field security? [What you need to know](#)

Auditing * ☒ Enable ☐ Disable

Description

Type an ID number or code for the account to quickly search and identify the account in system views.

Appears in global filter in interactive experience ☐ Sortable in interactive experience dashboard ☐

For information about how to interact with entities and fields programmatically, see the [Microsoft Dynamics 365 SDK](#)

Type

Data Type * Single Line of Text

Field Type * Simple

Format * Text

Maximum Length * 20

IME Mode * auto

If we open an Account form, and add the Account Number field, ongoing to the properties we can see the Control used is set to Text Box:

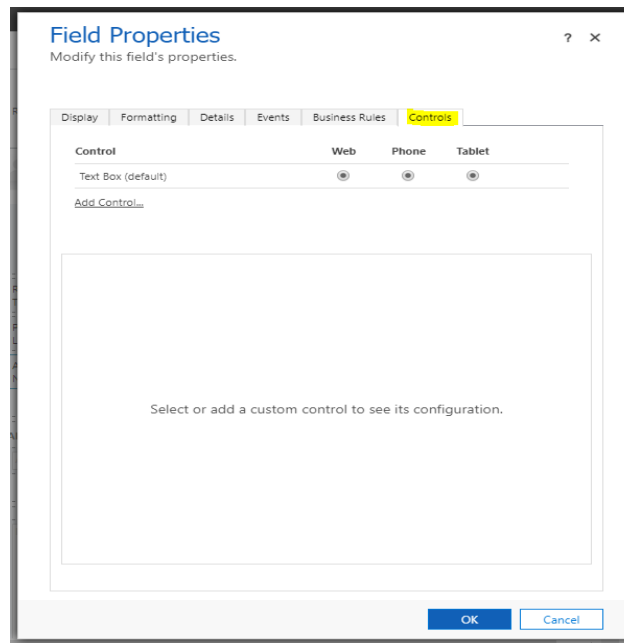
We help to build the code component controls based on your specific needs.

Cognitive Convergence

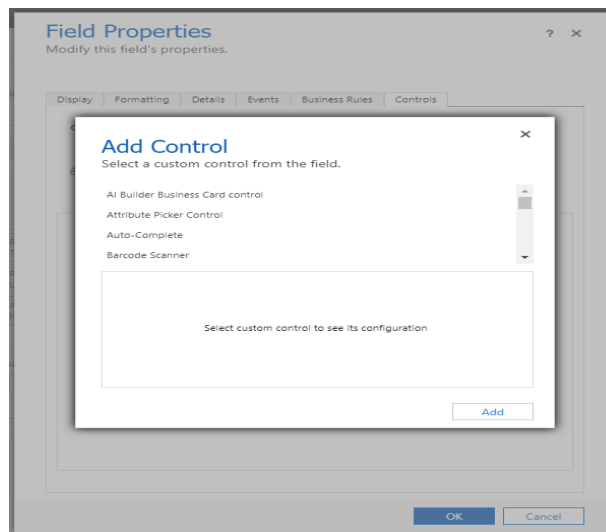
<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com



Clicking Add Control, we can change the control to be a different type, such as Auto Complete and Barcode Scanner:



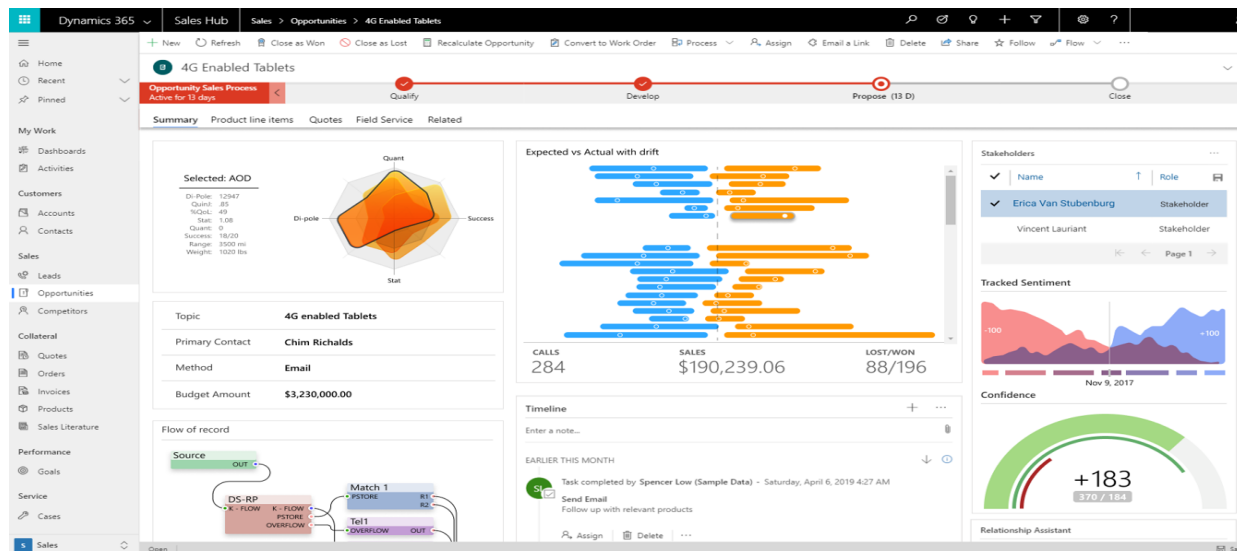
This is great, but what if the type of control we require doesn't exist, and we would like to add our own? This is where the PCF comes in. The PowerApps Component Framework allows us to build our own controls, which opens PowerApps controls to interesting use cases.

POWERAPPS COMPONENT FRAMEWORK (PCF)

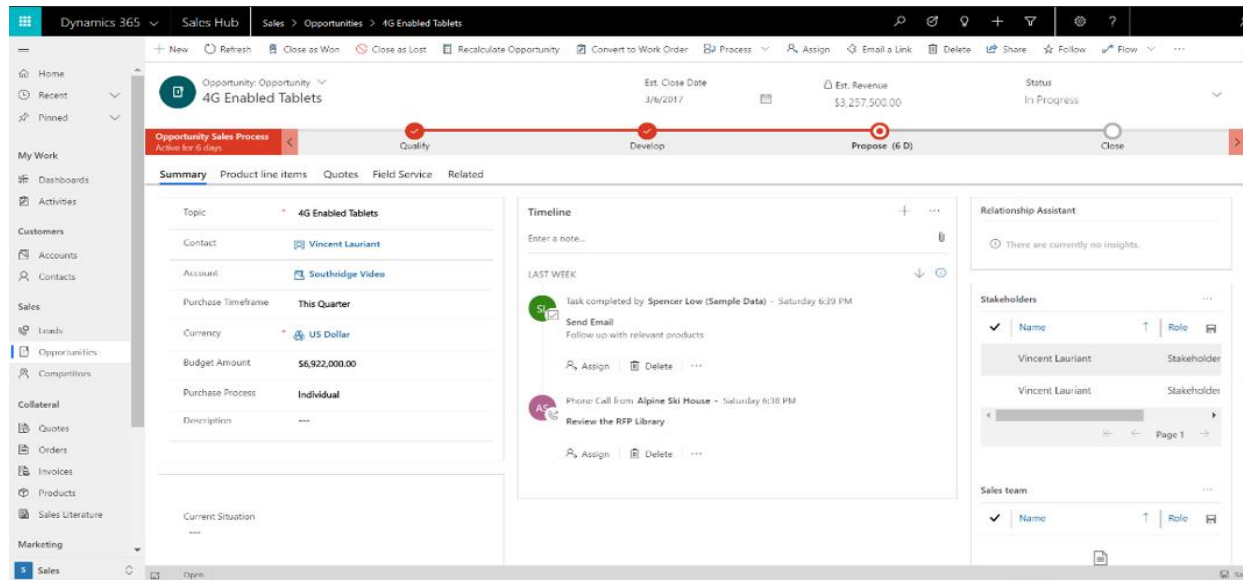
Power Apps component framework empowers professional developers and app makers to create code components for model-driven and canvas apps to provide enhanced user experience for the users to work with data on forms, views, and dashboards. A few sample implementations:

- Replace a field that displays a numeric text value with a dial or slider code component.
- Transform a list into an entirely different visual experience bound to the data set like a Calendar or Map.

For example, the existing screen might render like the following image.



However, if you reconfigured your app to use custom Power Apps components, your app might look something like the following image.



Power Apps component framework works only on Unified Interface and not on the web client and it doesn't work for on-premises instances.

Commonly added Components

It might be said that if you can build it in HTML+JavaScript, you can build it as a Power Apps component. However, common types of components that you might want to use in a Power Apps application might be:

- A custom control for a field on a form
- A custom grid (or sub grid) to display data in a tabular format
- A component that displays content from external services



Power Apps Component Framework

With access to a rich set of framework APIs,
our consultancy will support you for
modern web practices.

Cognitive Convergence

<http://www.cognitiveconvergence.com>

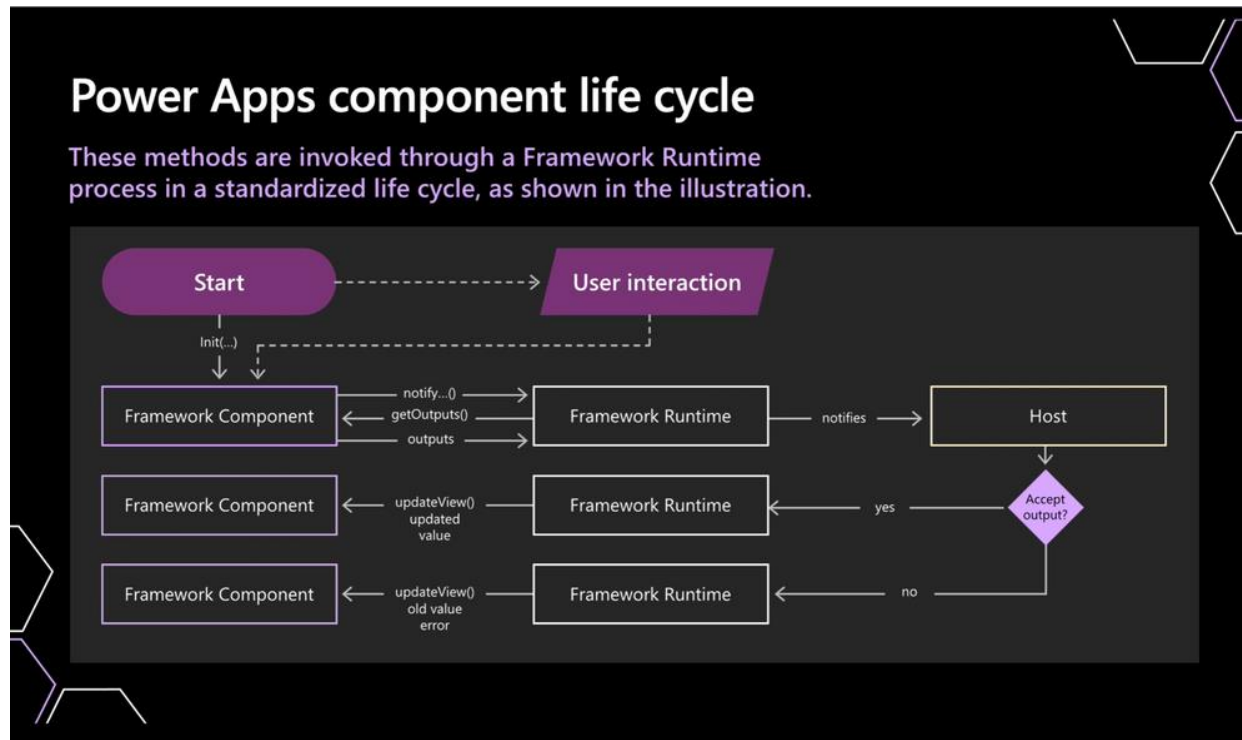
+1 4242530744

shahzad@cognitiveconvergence.com

PCF DEVELOPMENT ARCHITECTURE

PCF Development requires Node.js and PowerApps CLI to be installed and used.

After installation, PCF code component is created in Visual Studio



Creating a new Code Component

1. Creating a new folder on local machine, for example,
C:\Users\your name\Documents\My_code_Component using the command

```
mkdir <Specify the folder name>
```

command

```
cd <specify your new folder path>
```

2. Newly created folder using the

3. New component project by passing some basic parameters using the command:

```
pac pcf init --namespace <specify your namespace here>  
--name <Name of the code component> --template <component type>
```

4. Retrieving all the required project dependencies, for example,

```
npm install
```

Building Code Component

After implementing the required artifacts for the component like manifest, component logic, and styling and then build the component project by using following command

```
npm run build
```

Debugging Code Component

After implementing the code component logic, starting the debugging process.

```
npm start
```

Packaging Code Component

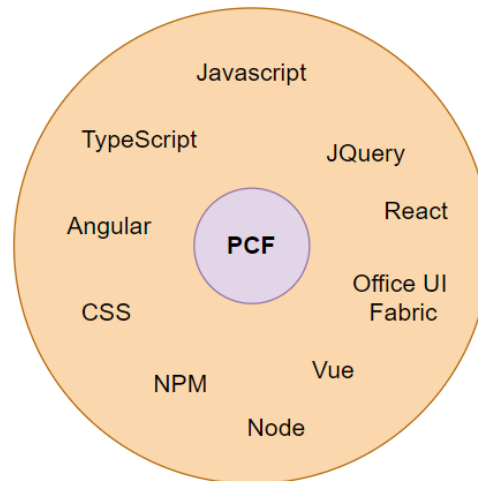
After completing the implementation of code component, we need to export it as a component to be installed in Dataverse. This includes creating solution project, adding reference to the code component, generating zip file from the solution project and running msbuild to build the solution.

CUSTOM PCF PROJECT

Prerequisites

- Visual Studio Code (VSCode) (Ensure the Add to PATH option is select)
- node.js (LTS version is recommended)
- Microsoft Power Platform CLI (Use either the Visual Studio Code extension or the MSI installer)

- One of the following:
 - Visual Studio 2019 for Windows & Mac. Select at minimum the workload .NET build tools.
 - Build Tools for Visual Studio 2019. Select at minimum the workload .NET build tools.



Manifest control implementation

The control node defines the namespace, version, and display name of the code component. Each property of the control node as shown here:

namespace: Namespace of the code component.

Constructor: Constructor of the code component.

Version: Version of the component. Whenever you update the component, you need to update the version to see the latest changes in the runtime.

display-name-key: Name of the code component that is displayed on the UI.

description-name-key: Description of the code component that is displayed on the UI.

control-type: The code component type. Only *standard* types of code components are supported.


```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <control namespace="SampleNamespace" constructor="LinearInputControl" version="1.1.0" display-name-key="controlValue_Display_Key" description-key="controlValue_Display_Key" />
</manifest>
```

Manifest property

The property node defines the properties of the code component like defining the data type of the column. The property node is specified as the child element under the control element.

name: Name of the property.

display-name-key: Display name of the property that is displayed on the UI.

description-name-key: Description of the property that is displayed on the UI.

of-type-group: The of-type-group is used when you want to have more than two data type columns. Add the of-type-group element as a sibling to the property element in the manifest. The of-type-group specifies the component value and can contain whole, currency, floating point, or decimal values.

usage: Has two properties, *bound* and *input*. Bound properties are bound only to the value of the column. Input properties are either bound to a column or allow a static value.

required: Defines whether the property is required.

```
<property name="controlValue" display-name-key="controlValue_Display_Key" description-key="controlValue_Display_Key" />
```

Manifest resource


The resources node defines the visualization of the code component. It contains all the resources that build the visualization and styling of the code component. The code is specified as a child element under the resources element.

- **code:** Refers to the path where all the resource files are located.

```
<resources>
  <code path="index.ts" order="1" />
  <css path="css/LinearInputControl.css" order="1" />
</resources>
```

Implementation of code logic

We worked on Linear input control and after the implementation of code logic, we added some style in the code component using CSS. The implementation is done in C# .NET



Our consultancy will help you to effectively use Power Apps code components in your business to drive user engagement.


Cognitive Convergence

<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com

Linear input control PCF added in Model Driven App

Annual Revenue	<div><div></div><div>\$565.00</div></div>
Number of Employees	---
Owner	*  Michelle Carter

Web API

Provides properties and methods to use Web API to create and manage records. Available for Model-driven apps.

Methods

Method	Description
createRecord	Creates an entity record.
deleteRecord	Deletes an entity record.
retrieveMultipleRecords	Retrieves a collection of entity records.
retrieveRecord	Retrieves an entity record.
updateRecord	Updates an entity record.

DataSet

Provides properties and methods to work with data on grids and views. Available for Model-driven apps and Canvas apps

Properties

addColumn()

Adds a column to the column set

Remarks

This method accepts two parameters.

Name	Type	Required	Description
name	string	Yes	Column name to be added to the dataset.
entityAlias	string	No	Entity alias for which the column name needs to be added.

columns

The set of columns available in this dataset.

Type: Column[]

error

Whether an error occurred in data retrieval.

Type: boolean

errorMessage

The error message associated with the last encountered error, if applicable.

Type: string

filtering

The column filtering for the current query.

Type: Filtering

linking

Defines the linked entity information.

Type: Linking

loading

Indicates whether the dataset is loading or not.

Type: boolean

paging

Pagination status and actions.

Type: Paging

records

Map of IDs to the full record object.

We build a complete functional PCF control,
from setting up environment to deploying
the control.

Cognitive Convergence

<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com

Type: EntityRecord

sortedRecordIds

IDs of the records in the dataset, order by the query response result.

Type: string[]

sorting

The sorting status for the current query.

Type: SortStatus

Methods

Method	Description
clearSelectedRecordIds	Clears the selected record ids list.
getSelectedRecordIds	Retrieves all the selected record ids.
getTargetEntityType	Returns the target entity type name.
getTitle	Retrieves the view display name used by the data-set property.
getViewId	Returns the Id of view used by data-set parameter.
openDatasetItem	Open data-set item for a given EntityReference. It checks if there is a command with command button id Mscrm.OpenRecordItem. If exists, it executes the command, otherwise it just navigates to the associated form of the EntityReference.
refresh	Refreshes the data-set based on filters, sorting, linking, new column.
setSelectedRecordIds	Set the ids of the selected records.

Context

Provides all the properties and methods available in the Power Apps component framework. Available for Model-driven apps and canvas apps.

client

Provides access to the methods to determine which client is being used, whether the client is connected to server, and what kind of device is being used.

Type: Client

device

Provides methods to use native device capabilities.

Type: Device

factory

Provides properties and methods to work with Popup services.

Type: Factory

formatting

Provides properties and methods to work with formatting.

Type: Formatting

mode

Provides access to methods to get the information about the current state of the code component.

Type: Mode

navigation

Provides navigation-related methods.

Type: Navigation

parameters

The data provided to the component. Structure defined by the component's manifest, corresponding to parameter and data-set nodes.

Type: TInputs

resources

The resource interface of **context.resource** provides access to the methods to get all the information about the resource files defined in the manifest.

updatedProperties

An array of strings with values that have changed since the last time it was passed and parameters. `updatesProperties` is currently only supported for model-driven apps and always returns empty string for canvas apps.

Type: string[]

userSettings

Provides information about the current user settings.

Type: UserSettings

utils

Provides a container for useful methods.

Type: Utility

webAPI

Provides properties and methods to use Web API to create and manage records.

Type: WebApi

Filtering

Provides properties and methods for filtering in a data-set. Available for Model-driven apps.

Method	Description
clearFilter	Clears the filter associated with the data-set.
getFilter	Returns the top-most filter associated with the data-set.
setFilter	Sets the top-most filter associated with the data-set.

Linking

Provides properties and methods to determine which entity is linked and to get all the entities linked. Available for Model-driven apps.

Method	Description
addLinkedEntity	Adds a new linked entity relationship with the existing query primary entity.
getLinkedEntities	Returns all the linked entities information.

EntityRecord

Base interface for data-set record result. Supports value retrieval by column name. Available for Model-driven apps.

Methods	Description
getFormattedValue	Gets the current formatted value of the record column.
getRecordId	Gets the record ID.
getValue	Gets the raw value of the record's column.
getNamedreference	Gets the object that encapsulates an EntityReference as a plain object.

EntityReference

An object that encapsulates an Entity reference as a plain object suitable for storing in the state tree. Available for Model-driven apps.

Properties

etn

The entity type name. Read-only.

Type: string

id

The record id. Read-only.

Type: object

The id object contains the following property:

Name	Type	Description
guid	string	00000000-0000-0000-0000-000000000000

name

The name of the entity reference. Read-only.

Type: string

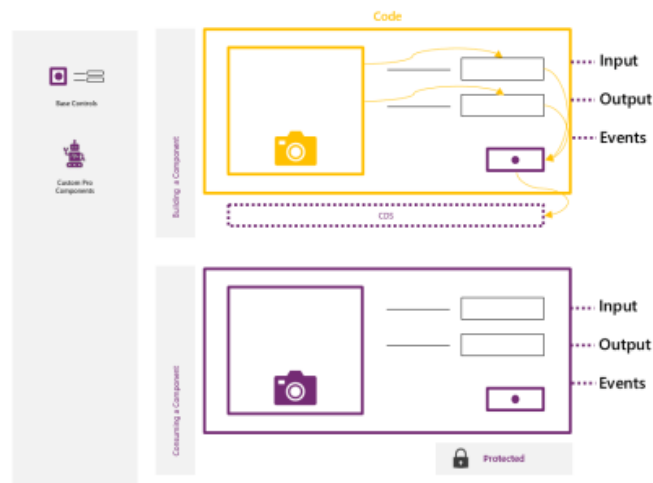
We optimize your business performance through reusability, bundling of solution and seamless server access via Web API.

Cognitive Convergence

<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com



PowerApps component framework can be used to develop leaf or base controls by Pro Developers through code. **Code can also be used to connect base controls.**

Custom Pro Developer components use input and output properties, and can connect only to CDS

Custom **Pro Developer** components are **consumed as black box** components to Makers, and can't be customized through maker experiences, except for developer designated properties.

CHALLENGES

Like most development scenarios, implementing PCF has challenges that need to be overcome. Here are a few of the challenges to keep in mind when embarking on PCF control development:

1. Build Error

The name of the task in the project file can be the same as the name of the task class. The task class can be public and implements the `Microsoft.Build.Framework.ITask` interface.

2. Publisher Prefix

If a component is created using the CLI version lower than 0.4.3, you will encounter an error while trying to reimport the solution file into Common Data Service.

3. Web Resource Size

There can be an error while importing solution and it leads to import failure. The reason of the error can be "Web resource content size is too big".

CONCLUSION

PowerApps component framework (PCF) can be used to create custom components in model-driven apps to provide an enhanced user experience for the users to view and work with data in forms, views and dashboards.

Cognitive Convergence is currently helping a handful of clients developing PCF controls, so we can attest that it is a strong addition to the PowerApps platform for building custom components in PowerApps. There are some risks and challenges developing PCF controls and the easiest way to mitigate these risks is to hire an independent, technology-agnostic digital transformation consulting firm such as Cognitive Convergence to help validate your software decision, prepare for implementation readiness, help manage your organizational change program, and provide quality assurance over your system integrator.



Contact Us

Cognitive Convergence

<http://www.cognitiveconvergence.com>

+1 4242530744

shahzad@cognitiveconvergence.com