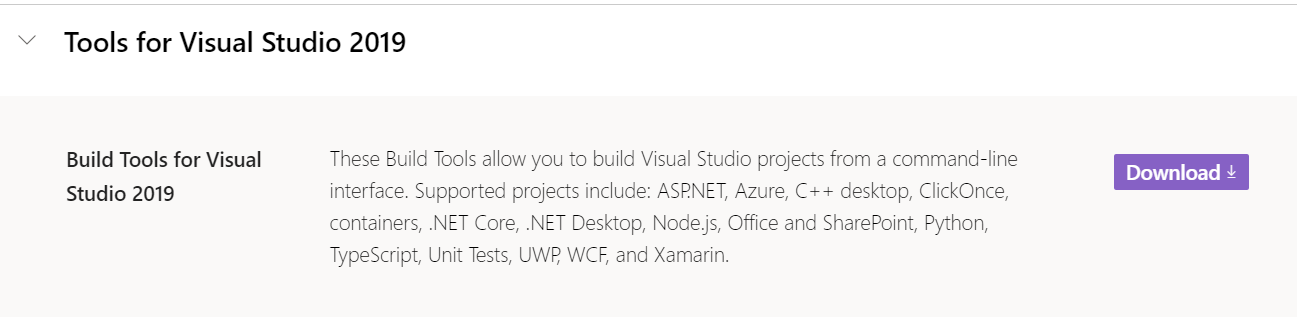
|  |  |
| --- | --- |
| Warning | **Warning**: This is my personal note during my learning PCF Control Development. I am sorry for any incorrect grammars or unclear explaination. This file will be updated along with my PCF learning journey. Information here may contain inaccurate or inadequate information.  If you find any places that need to be corrected or any questions related to subjects discussed here, please contact me at: [knguyen@procentrix.com](mailto:knguyen@procentrix.com).  For better navivate between topics, turn on Navigation using View – Nagivation. |
| **Last update** | **2021-03-01, 2021-03-05** |

## 1. Installation

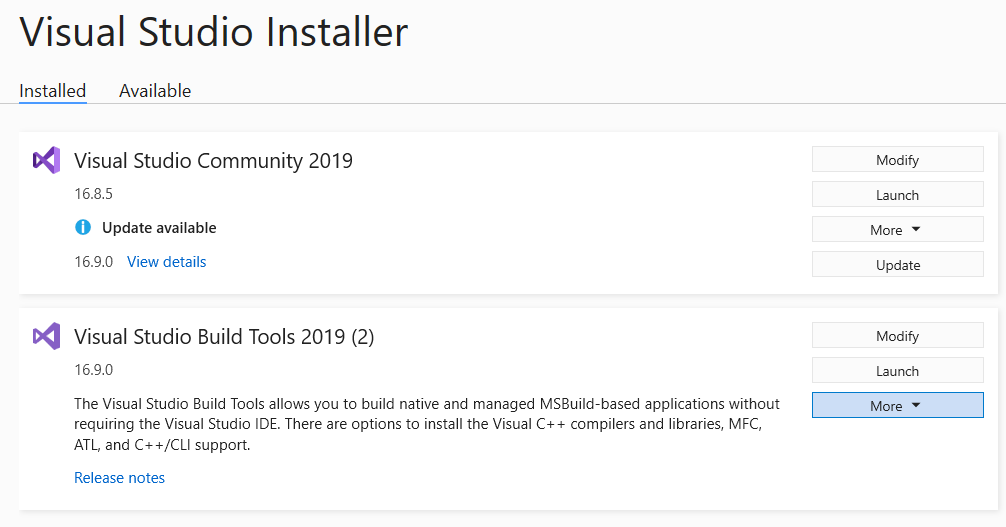
### 1.1 Install MS-BUILD tools

Download Ms-Build here: [Download Visual Studio 2019 for Windows & Mac (microsoft.com)](https://visualstudio.microsoft.com/downloads/?q=build+tools)

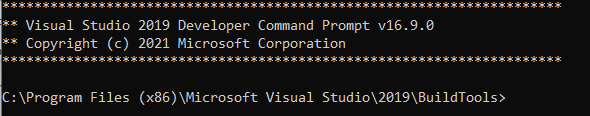
From above URL, download the downloader (which allow you to download Visual Studio, Ms-Build and other tools). Scrool down to the bottom and click download.



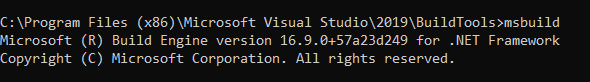
**After you install your downloader’s screen look like this :**



This is where Micrsoft store MS-BUILD

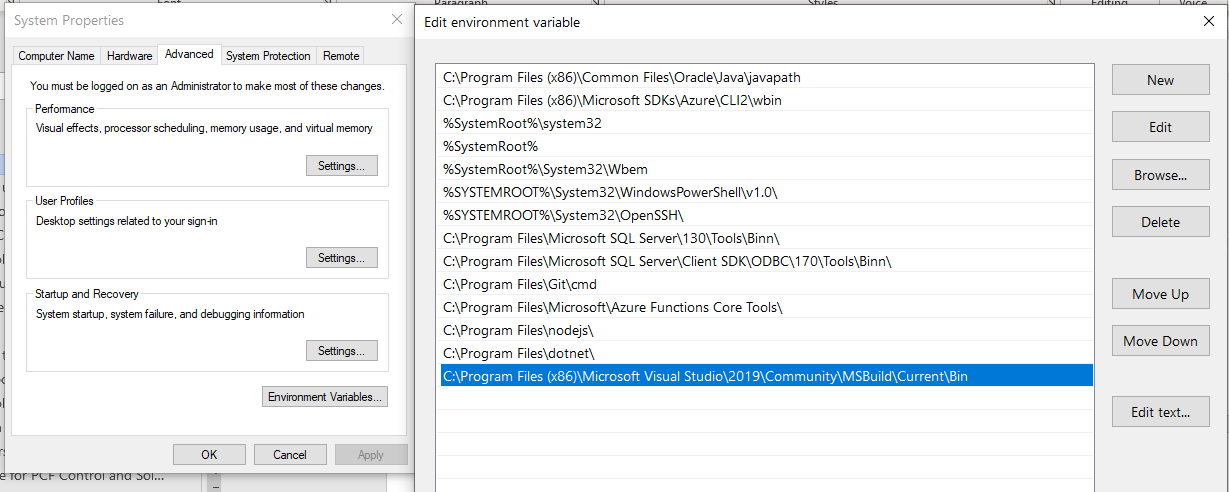


Check MS-BUILD version, type msbuild :



If you have another version of MS-BUILD intalled, you might need to modify the Environment Variable, or your call to this tool may go to the older version.

Update your system environment PATH variable to the latest version of MS-BUILD.



Close all terminal windows and reopen after you update the PATH system variable so the changes you made is taking effective.

### 1.2 Install PAC tool

Download the PAC command line tool here:

<https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#pcf>

If you already install, update it to latest version:

pac install latest

### Most common commands used by PAC

Click the links associated to each subject to read more about that command.

|  |  |
| --- | --- |
| Command | Description |
| [admin](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#admin) | Commands for environment lifecycle features. |
| [auth](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#auth) | Commands to [authenticate to Dataverse](https://docs.microsoft.com/en-us/powerapps/developer/component-framework/import-custom-controls#connecting-to-your-environment). |
| [org](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#org) | Commands for working with Dataverse environment. |
| [package](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#package) | Commands for working with [Solution Packages](https://docs.microsoft.com/en-us/power-platform/alm/package-deployer-tool). |
| [pcf](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#pcf) | Commands to work with [Power Apps component framework](https://docs.microsoft.com/en-us/powerapps/developer/component-framework/overview). |
| [plugin](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#plugin) | Command to create a [plug-in](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/plug-ins) project. |
| [solution](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#solution) | Commands for working with [Microsoft Dataverse solution projects](https://docs.microsoft.com/en-us/powerapps/maker/data-platform/solutions-overview). |
| [telemetry](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#telemetry) | Manages the telemetry settings. |

## 2. PowerApp CLI command line tool:

Link: <https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#pcf>

### 2.1 pac pcf - command

This section discusses commands used to to work with Power Apps component framework. It has the following parameters:

|  |  |  |
| --- | --- | --- |
| PARAMETERS | | |
| Property Name | **Description** | **Example** |
| init | Initializes the code component project. It has the following parameters - namespace: Namespace of the code component. - name: Name of the code component. - template: available values are: field or dataset.  choose field if your component is bound to a CRM column  choose dataset if your component binds to a CRM table. | pac pcf init --namespace <SampleNameSpace> --name SampleComponent --template field |
| push | Pushes the code component to the Dataverse instance with all the latest changes. It has the following parameter: - publisher-prefix: Publisher prefix of the organization.  After this step, your component will be in the environment as a solution with name : PowerAppTools\_<prefix>. | pac pcf push --publisher-prefix dev |
| version | Updates the component manifest file with the specified patch version. It has the following parameters: - patchversion: Patch version of the code component. patchversion will only take value of the third part of the version tuple: Major.Minor.Patch. - path: Absolute or relative path of the component manifest file. - allmanifests: Updates the patch version for all the component manifest files. - updatetarget: Updates the specified manifest file. It has two values, build and project. - strategy: Updates patch version for the manifest files using specified strategy values. It has the following values: - gittags: Use git tags to decide if a particular component’s patch version needs to be updated. filetracking: Use .csv file to decide if a particular component’s patch version needs to be updated. - manifest: Increments the patch version by 1 for all the components. | pac pcf version --patchversion 1.0.0.0 --path c:\Users\Downloads\SampleComponent –allmanifests  pac pcf version --strategy gittags |

### 2.2 pac solution - command

Commands for working with [Dataverse solution projects](https://docs.microsoft.com/en-us/powerapps/maker/data-platform/solutions-overview). It has the following parameters:

|  |  |  |
| --- | --- | --- |
|  | PARAMETERS |  |
| Property Name | **Description** | **Example** |
| init | Initializes the solution project. It has the following parameters: - publisher-name: Publisher name of the organization. - publisher-prefix: Publisher prefix of the organization. | pac solution init --publisher-name developer --publisher-prefix dev |
| add-reference | Sets the reference path to the component project folder by passing the path parameter. | pac solution add-reference --path c:\Users\Downloads\SampleComponent |
| clone | Creates a solution project based up on the existing solution project. It has the following parameters: -name: The name of the solution to be exported. -targetversion: The version that the exported solution supports. -include: Settings that should be included in the solution being exported. It has the following values: autonumbering, calendar, customization, emailtracking, externalapplications, general, isvconfig, marketing, outlooksynchronization, relationshiproles, sales | pac solution clone -–name sampleSolution --version 1.0.0.2 --include general |
| import | Imports a Dataverse solution to an environment. It requires that you are connected to an environment [Auth commands](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#auth) and has the following parameters: -activate-plugins: Activates plug-ins and workflows in the environment after the import (alias: -ap). -async: Imports the solution asynchronously (alias: -a). -force-overwrite: Forces an overwrite of unmanaged customizations (alias: -f). -import-as-holding: Imports the solution as a holding solution (alias: -h). -max-async-wait-time: Maximum asynchronous wait time in minutes. Default value is 60 mintues (alias: -wt). -path: Path to solution zip file. If not specified, assumes the current folder (alias: -p). -publish-changes: Publishes changes after successful import (alias: -pc). -skip-dependency-check: Skips dependency check against dependencies flagged as product update (alias: -s). | pac solution import --path c:\Users\Documents\Solution.zip |
| export | Exports a Dataverse solution from an environment. It requires that you are connected to an environment [Auth commands](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#auth) and has the following parameters: -path: Complete file name where the exported solution zip file will be saved. - name: Name of the solution that needs to be exported. - managed: Defines whether the solution should be exported as a managed solution or not. -targetversion: The version that the exported solution supports. -include: Settings that should be included in the solution being exported. | pac solution export --path c:\Users\Documents\Solution.zip -- name SampleComponentSolution --managed true --targetversion 10.0.03 --include general |
| list | List all Solutions from a Dataverse environment. It requires that you are connected to an environment [Auth commands](https://docs.microsoft.com/en-us/powerapps/developer/data-platform/powerapps-cli#auth). This command has no parameters: | pac solution list |

## 3. Working with PCF control

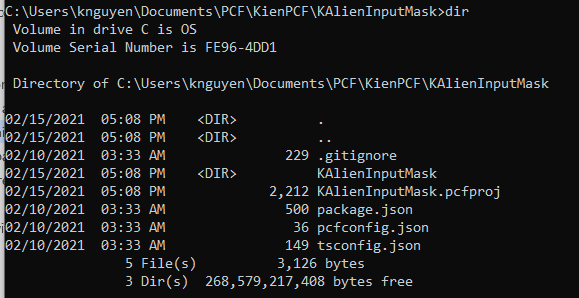
### 3.1 Initialize a new component

#### Step 1: Initialize PCF component

To initialize the PCF control, run PAC command with following options

|  |  |
| --- | --- |
| OPTION | DESCRIPTION |
| --namespace | the namespace for the component (alias: -ns) |
| --name | the name for the component (alias: -n) |
| --template | choose a template for the component (alias: -t), available option are :  field : if the control bound to a field  dataset : if the control bound to the whole table |

pac pcf init --namespace KntControls --name KAlienInputMask --template field



Notice that package.json, pcfconfig.json and tsconfig.json and KAlientInputMask.pcfproj are generated for you.

Here is package.json file:

{

**"name"**: "pcf-project",

**"version"**: "1.0.0",

**"description"**: "Project containing your PowerApps Component Framework (PCF) control.",

**"scripts"**: {

**"build"**: "pcf-scripts build",

**"clean"**: "pcf-scripts clean",

**"rebuild"**: "pcf-scripts rebuild",

**"start"**: "pcf-scripts start"

  },

**"dependencies"**: {

**"@types/node"**: "^10.12.18",

**"@types/powerapps-component-framework"**: "^1.2.0"

  },

**"devDependencies"**: {

**"pcf-scripts"**: "^1",

**"pcf-start"**: "^1"

  }

}

About package.json file.

**version**: is your control version. The version pattern has 3 parts. Will be discussed in section 5.3.

**dependencies** attribute lists dependencies used by your control. Here, notice that your control needs two dependencies which are node-js and powerapps-component-framework. In the next step, when we install node library, NPM will bring this dependency into our working environment.

#### Step 2: Install node library

npm install

To be able to run test and start to develop code for you PCF, NodeJS must be installed in your system. If not, go to section 1: **Install PAC Utility,** or you can install Node JS with NPM here : <https://www.npmjs.com/get-npm>

NPM look for the information inside package.json that was created by previous step, so it knows which dependencies are needed to be installed. Hence. you must have package.json ready for Node to know what to be installed. In case of PCF, the command pac pcf init that discuss in section 3.1 create package.json for you.

#### Step 3 : Update component manifest

##### <control> attribute:

Control property describes your PCF control. Change the version, display-name-key, and description-key properties that are found in the KAlienInputMask node of ControlManifest.Input.xml file to more meaningful values.

* version: is your PCF version. It has pattern MAJOR>.<MINOR>.<PATCH>. It is important to always adjust your PCF version OR your solution version every time you upload your solution zip file to Dynamic environment, otherwise your old code will not be replaced / upgraded. [Read more](#_5.3_PCF_Version).

The PCF version is not import if you push you code using pac push command. [Read more](#_5.3_PCF_Version).

* display-name-key: unique key used as control name.
* description-key: text that describe what you control is, this provide a developer who use your control to know about what your control does.

More info about other attributes, refer to [Microsoft Docs](https://docs.microsoft.com/en-us/powerapps/developer/component-framework/manifest-schema-reference/) .

Example control

  <**control** namespace="KntControls"

    constructor="KAlienInputMask"

    version="0.0.3"

    display-name-key="KntControls.KAlienInputMask"

    description-key="Allows applying mask on the input for alien number"

    control-type="standard">

##### <property> attribute

This attibute of the manifest file describe the properties that your PCF control may have. For sample, an input text may have property value to hold its input value, text length for max characters user is allowed to enter.

When a developer adop your PCF to use, the unbound properties allowes them to configure your PCF.

A sample property attribute:

    <**property** name="errorMessage"

      display-name-key="error-message"

      description-key="Error message that displays when the field fails requirements"

      of-type="SingleLine.Text"

      usage="input"

      default-value = "the input must have 'A' following by 7 to 9 digits."

      required="true" />

* usage : can have two available options: input or bound. The bound is used for property that is bound to CRM field, input is used for all others.

Example of bound property

    <**property** name="entityField"

      display-name-key="input\_field"

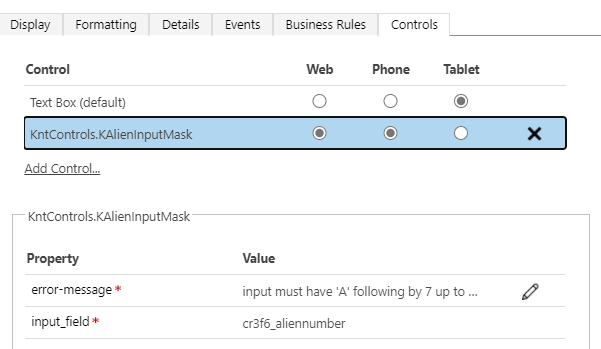
      description-key="The entity field that input mask will be applied (bound) to"

      of-type="SingleLine.Text"

      usage="bound"

      required="true" />

Notice two above properties show up on the PCF settings windows



##### <property> of-type Enum

Don’t know why using property with of-type= “TwoOptions” does not work. For sample here is a property with two options Yes/No: it does not work. When developers open the control setting popup, the drop-down “bind to static options”: has no value in it.

<**property** name="saveToField"

      display-name-key="save to field?"

      description-key="Save binding colum or just display?"

      of-type="TwoOptions"

      usage="input"

      required="true"

      default-value="0">

        <**value** name="no" display-name-key="No"

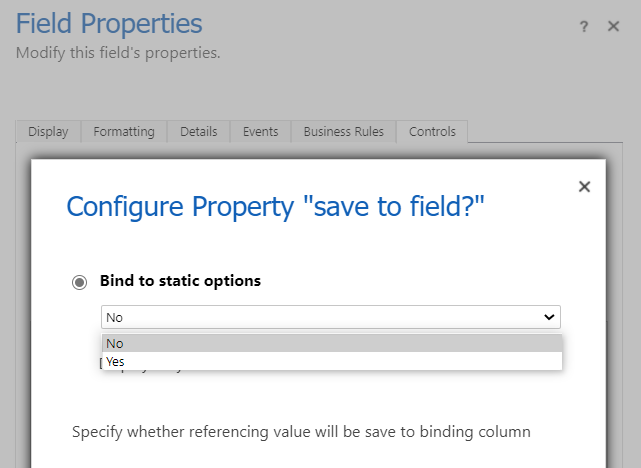
description-key="Display only">0</**value**>

        <**value** name="yes" display-name-key="Yes"

description-key="Save to the field.">1</**value**>

</**property**>

However, when changing of-type =“Enum”, it works perfectly:



### 3.2 Develop PCF control

#### StandardControl interface

The most of your control code is in the file index.ts. PCF controls need to implement ComponentFramework.StandardControl interface which need to implement 4 of its methods.

**interface** StandardControl<TInputs, TOutputs> {

    init(context, nofifyOutputChange, state, container , ) **void**;

    updateView(context): **void**;

    destroy(): **void**;

   getOutputs?(): TOutputs;

}

init() method is for initialize the PCF controls. This is where we declare internal property, set their default value, store some important object provided by the platform for later use such as context object or container object.

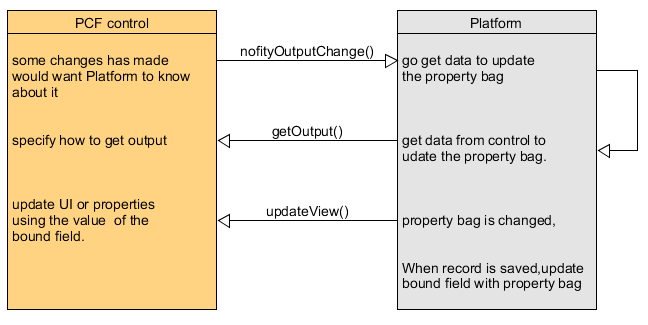
notifyOutputChange is a callback function that we must call whenever the PCF status has any changes that we need to notify the framework so it can update property bag.

For example, the PCF input text is changed, and we want to update that changes to the field that is bound to it so we call this method. The platform updates the changes to the property bag. When user click save a record, the value of property bag is written to the the bound field.

updateView() : this methos is called when the value of the field that is bound to your control has changed.

For example, when you PCF control first loaded, the value of the field that associated with the control is also loaded first time to the property bag. Platform calls updateView() so you can assign the field’s value to your control internal property or display it on the form.

getOutput() : for PCF control that is bound to a CRM field, this method tell platform how and what it can used to update the underline CRM field.



**Note important**:

In above diagram, notice that the platform only writes property bag to the field when the record is saved. Hence, do not call notifyOutputChange() in your updateView(), because doing that will cause a circular loop which bring back the old value of the field back to the control so new value stored in the control will be erased.

(Here is how: User types new input to the control, the control calls notifyOutputChage() → Platform uses getOutput to get the value. Bag is updated → Platform auto call updateView() to tell control that bag is changed → updates its input with value of field (old value)

destroy() for clean up code. What should be clean up ? So far , even listerner, timeout function that used in the control should be clean up.

#### Handling event

#### Access Platform Utilities via context object

## 4. Working with Solution

### 4.1 Initialize solution with: pac init solution

Initializes the solution project which packing the application into distribute-able solution package. It has the following parameters:  
- publisher-name: Publisher name of the organization.  
- publisher-prefix: Publisher prefix of the organization.

**pac solution init --publisher-name** <you-publisher-name> **--publisher-prefix** <your-prefix>

example: **pac** **solution** **int** **--publisher-name** Procentrix **--publisher-prefix** pcx

### 4.2 Adding reference:

This step sets the reference path to the component project folder by passing the **path** parameter. The value of path is the directory point to the PCF component project file \*.pcfproj . Note: stay inside the solution folder to execute this command

pac solution add-reference --path c:\Users\Downloads\SampleComponent

### 4.2 Use clone

Creates a solution project based up on the existing solution project. It has the following parameters:  
-name: The name of the solution to be exported.  
-targetversion: The version that the exported solution supports.  
-include: Settings that should be included in the solution being exported.  
It has the following values: autonumbering, calendar, customization, emailtracking, externalapplications, general, isvconfig, marketing, outlooksynchronization, relationshiproles, sales

pac solution clone -–name sampleSolution --version 1.0.0.2 --include general

## 5. Upload the PCF to Dataverse environment

To upload a PCF to a Dataverse environment, we have two ways:

1. use “pac pcf push” to upload a debug version of the PCF. This will upload a debug version of the PCF to a solution called “PowerAppsTools\_<prefix>” (unmanaged)
2. make a “.cdsproj” for the solution (“pac solution init”), add a reference to the “.pcfproj” (using pac solution add-reference) and generate the Solution.zip using “msbuild”. This solution can be then imported to any Dataverse instance.

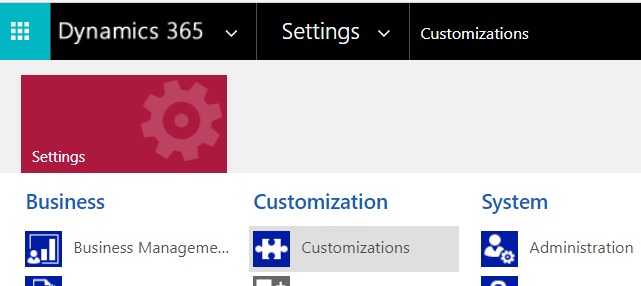
For the way 1 (pac pcf push), the version of the pcf **doesn’t need to be incremented**. This is intended to be used for development reasons only. This will skip the version checks, and just update the content.

The normal deployment should be made using way 2 (with a .cdsproj). When you import a solution.zip generated this way, the manifest (PCF) **version need to be incremented**. Otherwise you won’t get an error, but the content of the PCF won’t be upgraded. But there are some cases where, even if the manifest version was changed, the PCF won’t change.

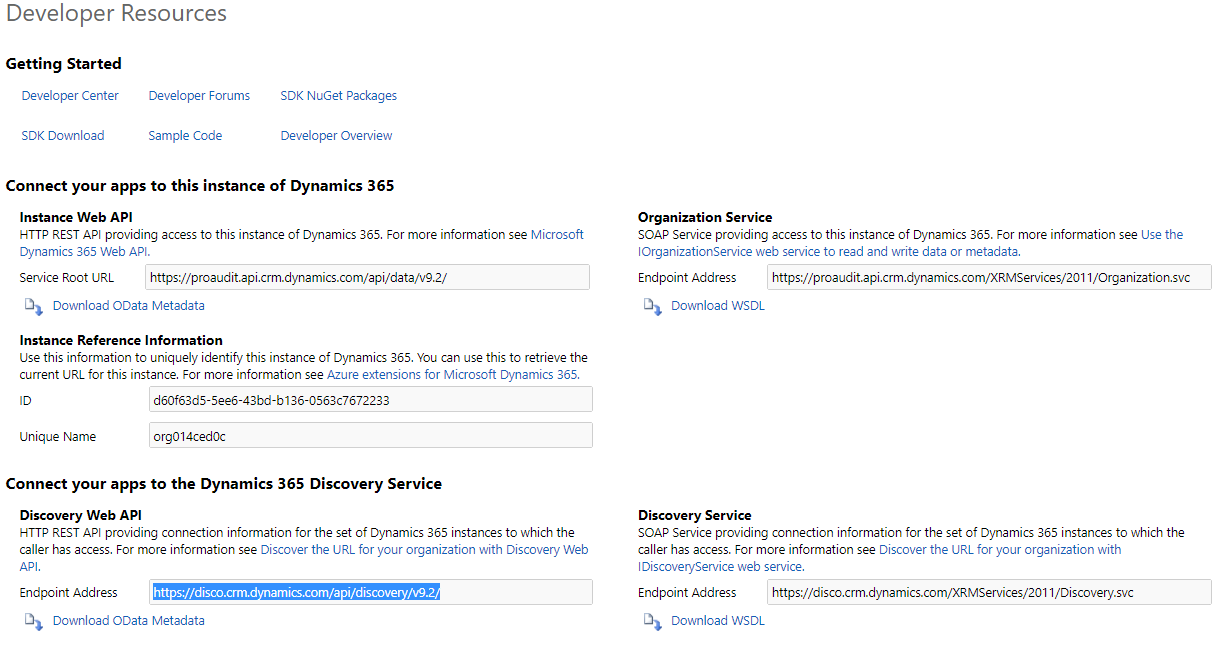
### 5.1 ­Setup pac pcf push

#### Step 1: get the endpoint:

In your PowerApp environment, select **Advance Settings**, then select **Customization**,



Then click **Developer Resources**:



Copy the Endpoint Address in the section **Organization Service**:

In my case, the **End Point** is: https://proaudit.api.crm.dynamics.com/

#### Step 2: Create Authenticate Profile

In command line

pac auth create --url <your-end-point-url>

After pressing Enter, the popup screen ask you to sign in using your credential. After you sign in, your authentication is created.

#### Step 3: Import the solution into your org and publish

Run the command bellow and wait for the publishing to complete. The push command uploades your component to the configured environment. This can be used over and over uring development to quickly see your compolent in the live form.

pac pcf push --publisher-prefix knt

### 5.2 Build the Solution ZIP file

Two steps need to be done:

**Step 1: Intialze your solution**.

Make your solution folder. Best practice is making a new folder, sibling with your project folder. Stay inside your solution folder then enter this command:

**pac solution init --publisher-name** <you-publisher-name> **--publisher-prefix** <your-prefix>

**Step 2: Add path reference to your solution project**.

Tell your solution project, where your PCF project file (\*.proj) is located.

Stay inside solution folder, type this command,

pac --path <path-to-your-pcf-proj file>

**Step 3: Build and pack your solution**.

Stay inside the solution folder to execute this command. Note: always build your PCF control before packing your solution using these commands. Adjust your PCF version as needed. Always change version of solution so your PCF code is updated and will be reflected in your solution. (Read more: problem with PCF version, section 4.3)

msbuild /t:restore

msbuild

Your solution file is built and put in Debug folder.

**Control Manage or Unmanage Solution**

You can specify managed or unmanage solution my specify in Solution.xml.

<!-- Solution Package Type: Unmanaged(0)/Managed(1)/Both(2)-->

    <**Managed**>2</**Managed**>

You can also specify in solution \*.cdproj file

<**PropertyGroup**>

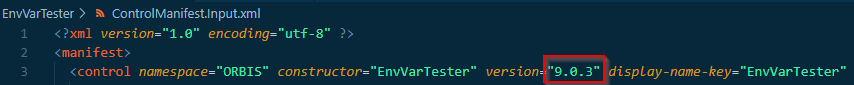
    <**SolutionPackageType**>Both</**SolutionPackageType**>

</**PropertyGroup**>

### 5.3 PCF Version Problem

One of the most common question asked in the Power Apps Pro Dev Community Forum is about the PCF version. In some cases, the PCF doesn’t seem to get updated, which creates a lot of frustration.

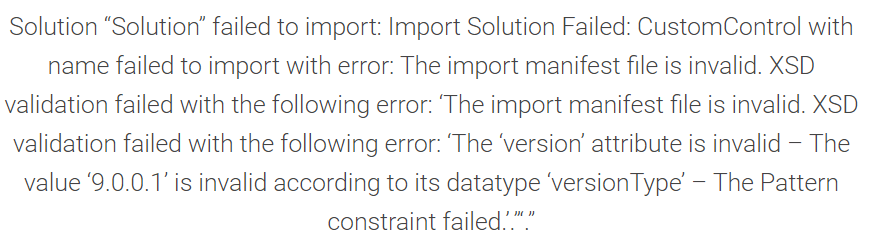
What is the PCF version? The PCF version is specified in the PCF manifest, as an attribute for the “control” node.



The SDK says that the version must follow the rules of “Semantic Versioning”. This means that it has three parts: <MAJOR>.<MINOR>.<PATCH>.

So following version works: 1.01, 1.1.5, 2.1.20202021, but this does not work : 2.0.3-beta. If you try, you will get error during the build.

Further, if you use version format such as 9.0.1.3, you can build the solution, but you will have error when you import the solution:

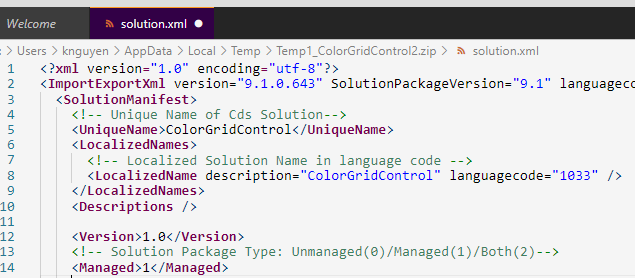


**Remember**: the solution version and PCF control version are two different things.

### 5.4 Solution Version

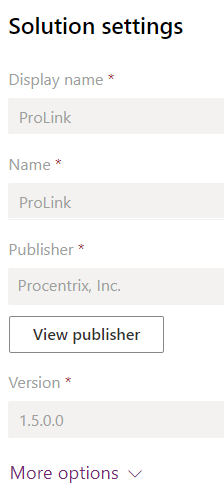
Remember: the solution version is not related to the PCF version.

To manipulate the solution version, open the file “src\Other\Solution.xml” , focus on the Version section



Solution verison is slightly different comparing with PCF version. Solution version can have 4 parts:

<MAJOR>.<MINOR>.<PATCH>.<BUILD>.



So solution version and PCF version are not related. In next section we discuss best practice to make them work together.

### 5.5 Best practice for PCF Control and Solution Version

As discussed above we have two way to upload PCF control to a Dataverse:

1. using pcf push for testing, and
2. import solution zip file for deployment.

The normal deployment should be made using way 2 (with a .cdsproj). When you import a solution.zip generated this way, the manifest (PCF) version needs to be incremented. Otherwise you won’t get an error, but the content of the PCF won’t be upgraded.

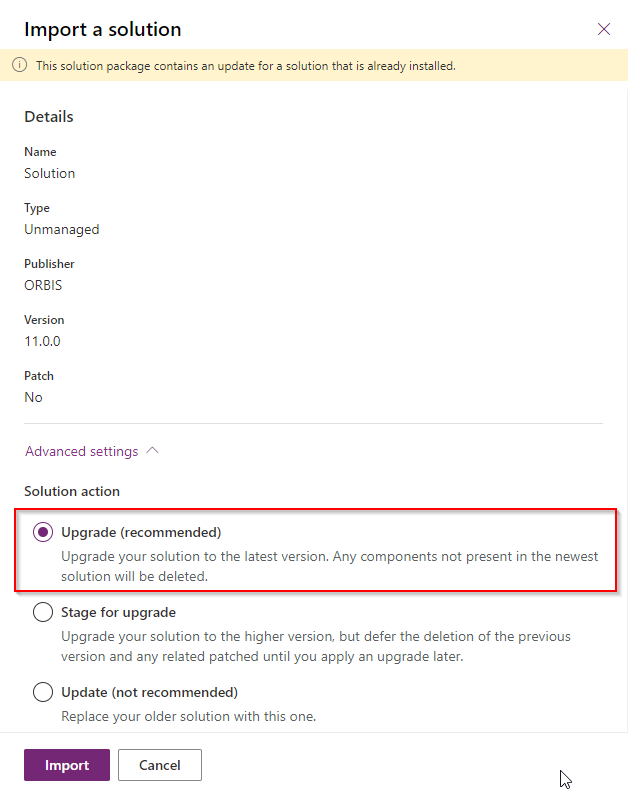
But there are some cases where, even if the manifest version was changed, the PCF won’t change.

<MAJOR>.<MINOR>.<PATCH>.<BUILD>.

#### Best Practice:

* **Incrementing the last part of the PCF Version (Patch) is always safe**. The changes will be reflected, the PCF content will be upgraded.
* Changing one of the first two numbers (**Major or Minor**) alone, the PCF won’t be upgraded… unless you increment the solution version too. It doesn’t matter what part of the Solution version you change.

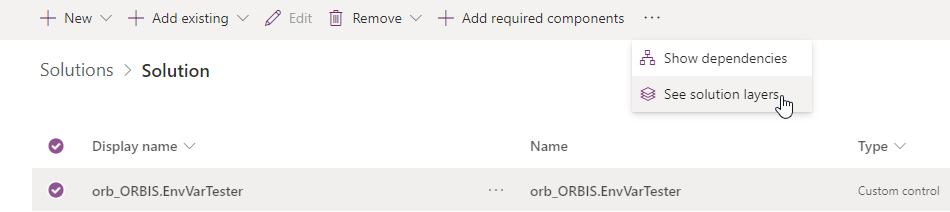
When import the update solution, choose **UPGRADE**, don’t use **~~UPDATE~~**.



#### Mixed Solution upload vs. Pac Push upload?

What if once we imported solution zip file then one day want to pac push the update so we can debug? Will it work? it depends.

* it does work when the solution.zip was unmanaged.
* if the **solution.zip was managed**, we see both versions inside the “Solutions Layers”



If the situation the solution is managed, the managed solution version takeover unmanaged.

Even if we customize the PCF on the form only now, we will get to choose the managed version of the PCF. Any change to the unmanaged layer, so as any change using “pac pcf push” won’t be seen on the form.

So if you want to go back to “pac pcf push” you need to remove the PCF from the form, **uninstall the managed solution**, and start over with “pac pcf push” and register the PCF again on the form.

**Note**: Using “pac pcf push” might help to develop, but the better way to debug PCFs is using the [Fiddler AutoResponder](https://dianabirkelbach.wordpress.com/2020/11/17/debug-your-pcfs-during-development-using-fiddler-autoresponder/).

## 6. Debug with PCF

## 7. Using jQuery in PCF

### 6.1 Install jQuery to PCF environment

jQuery provides lot of convernience methods to query and work with DOM elements. However, you should use PCF provided components and utilities instead of using jQuery. Two reasons:

1. Duplicate loading: other developers may also load jQuery to the environment which cause multiple versions being loaded that may causes conflict that hard to debug.
2. PCF does not recommend you to directly interact with DOM element since platform upgrade may caues the code not to work.

Recommend : use it as last resource if no other way to work arround it.

Here is how to install jQuery in the development environment so TS can compile properly. To do that, standing n the same folder as your package.json file, run following command:

npm install @types/jquery --save-dev

Save jQuery as a dependency to your project

npm install jquery --save

To install jQuery in both environment (dev and prod) , so you can run test on your local (using npm start)

npm install jquery

Then in your index.ts or any where you need to use it, import jQuery as following

**import** **\*** **as** $ **from** 'jquery';

### 6.2 Query for element that ID has dot

Normally, we use jQuery to retrieve an input field like this

$(“#myInput”).val();

But Microsoft always use a element with ID property that has DOT (.) in it. Two methods to work arround.

Method 1: Using \\.

$("#id-235746ed-1dcc-4536-980e-46b90c06cf8c-1-pro\_name8-pro\_name\\.fieldControl-text-box-text").val()

"kgrid"

Method 2: Using “[id=’...’]”

$("[id='id-235746ed-1dcc-4536-980e-46b90c06cf8c-1-pro\_name8-pro\_name.fieldControl-text-box-text']").val()

"kgrid"

## 10. Fluent UI with PCF control

Fluent UI are set of React controls that you can use for you PCF.

When you build your PCF, you are responsible to create the HTML DOM element associate to your PCF and styling them using CSS, assigning event handler / listerner to them as you need. That is a lot lof work.

Do not re-invent the wheel. We should use existing library as much as we can.

The reason you should use Fluent UI are:

* You want your PCF to have a consistent Microsoft style of look and feel (styling, behaviors to certain events)
* You want your control has characteristic, properties, even handlers that inherited from existing React Fluent UI controls.
* Fluent UI provides a control with rich properties, and behaviors, methods ready for us to use out of the box.

### 7.1 Installation

Initialize you PCF normally with PAC command. Then after that, install extra dependencies

//always run npm installation initializing you PCF control with pac command

npm install

//install React and its definitions and components

npm install react

npm install @types/react

npm install react-dom

npm install @fluentui/react

### 7.2 Use it in your code

We must import React before using it in our code.

**import** **\*** **as** React **from** 'react';

**import** **\*** **as** ReactDom **from** 'react-dom'; //import this where component will be render

If your control extends existing React components, you might need to import it before using. For example, here we import React’s Fabric and DetailsList component.

**import** {Fabric, DetailsList} **from** '@fluentui/react';

### 7.3 Using Icons

If your controls use any font-base fluent ui icon, it is not loaded by default. You must load it yourself. To make the icons available, you may initialize them as follows.

Note that initializeIcons() should only be called once per app and must be called before rendering any components. This is typically done in the app's top-level file just before the main ReactDOM.render() call. Read more here: [Using icons · microsoft/fluentui Wiki (github.com)](https://github.com/microsoft/fluentui/wiki/Using-icons)

**import** { initializeIcons } **from** '@fluentui/react/lib/Icons';

initializeIcons(/\* optional base url \*/);

### Trouble shoot PCF with Fluent UI

Cannot use JSX file unless the '--jsx' flag is provided. PCF does not see the code in JSX.

Go to the tsconfig.json and change that line from react-jsx to react:

{

"compilerOptions": {

"jsx": "react"

}

}

Restart your IDE. Sometimes tsconfig.json changes aren't immediately picked up.