Dataverse

These are some of the terms you may run up against with Dataverse:

* Entity: Refers to a table in Dataverse. Table and entity are often used interchangeably for data access.
* Virtual entity: Aka virtual tables – allows representing an external data set as a Dataverse entity.
* Record: a row in a table storing business data.

Dataverse has two major types of tables:

* Standard tables: created by the platform, e.g. the Account table
* Custom tables: created by the maker and suited for a specific scenario.

Tables appear in Power Apps as one of these different types:

* Standard: Several standard tables, also known as out-of-box tables, are included with a Power Platform environment, that includes Microsoft Dataverse. Account, business unit, contact, task, and user tables are examples of standard tables in Dataverse. Most of the standard tables included with Dataverse can be customized. Tables that are imported as part of a managed solution and set as customizable also appear as standard tables. Any user with appropriate privileges can customize these tables where the table property has customizable set to true.
* Activity: Are a special kind of table and are best for rows that have an activity-based element, which can include a subject, start time, stop time, due date, and duration. Dataverse already comes with several out-of-the-box activity tables, such as appointment, task, email, and phone call. Mre information: Activity tables
* Virtual: Are when you need the table to be populated with data from an external source outside of Dataverse.
* Elastic: Are for when the table will store a very large dataset in excess of tens of millions of rows.

Power App Cards

Power Apps cards are micro-apps with enterprise data and workflows and interactive, lightweight UI elements that other applications can use as content. Because they're part of the Power Apps ecosystem, cards can add business logic through Power Fx and integration with business data through Power Platform connectors. Using cards, you can quickly build and share rich, actionable apps without any coding or IT expertise.

To get started, sign in to Power Apps. Select Cards, and then select + Create a card. If the item isn’t in the side panel pane, select …More and then select the item you want.

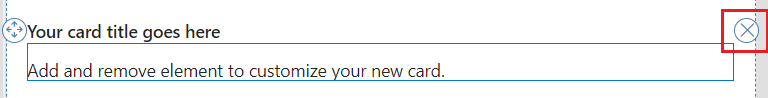
**Create a simple card**

**Create a card**

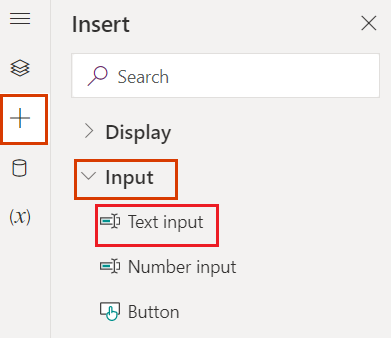
1. Sign in to Power Apps and select your environment.
2. In the left pane, select Cards. If the item isn’t in the side panel pane, select …More and then select the item you want.
3. Select + Create a card.
4. Under Card name, type HelloWorldCard, and then select Create.

**Ask for text input**

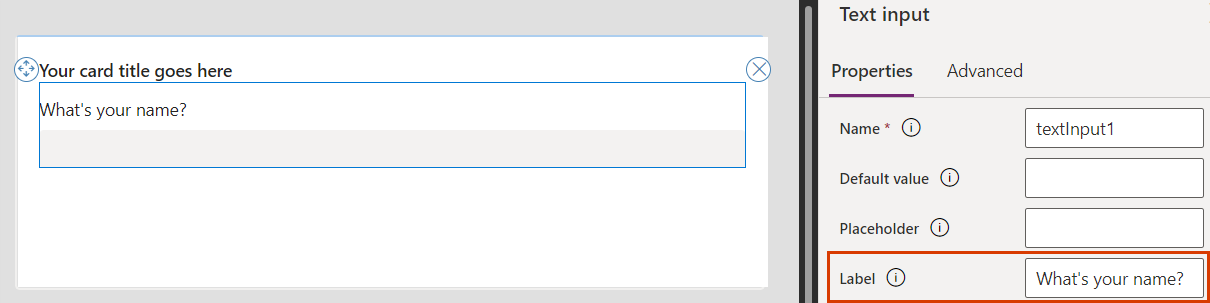
1. Select the text Add and remove element to customize your new card., and then select the Remove icon (X) to delete it.



1. In the left pane, select Insert.
2. In the tool pane, select Input to expand the category, and then select Text input.



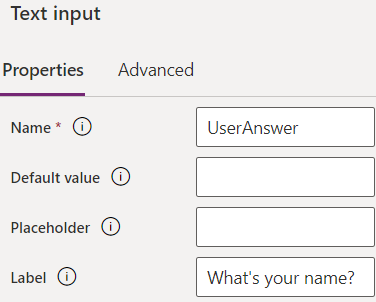
1. In the text input properties pane, set Label to What's your name?.



**Assign variables**

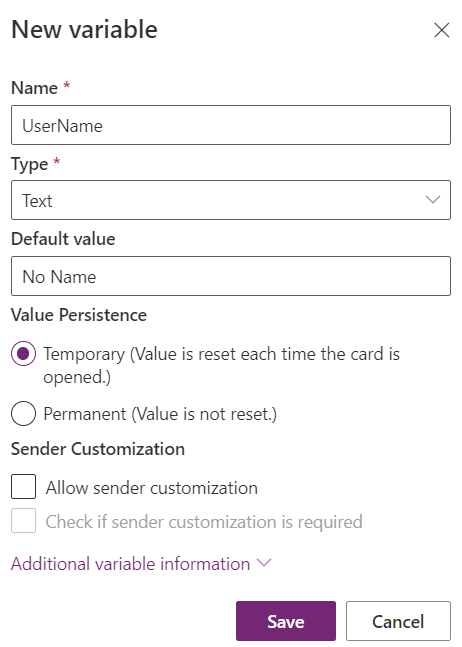
1. In the text input properties pane, set Name to UserAnswer.

We're giving the text input control the name of a variable that we're going to associate with the user's input.



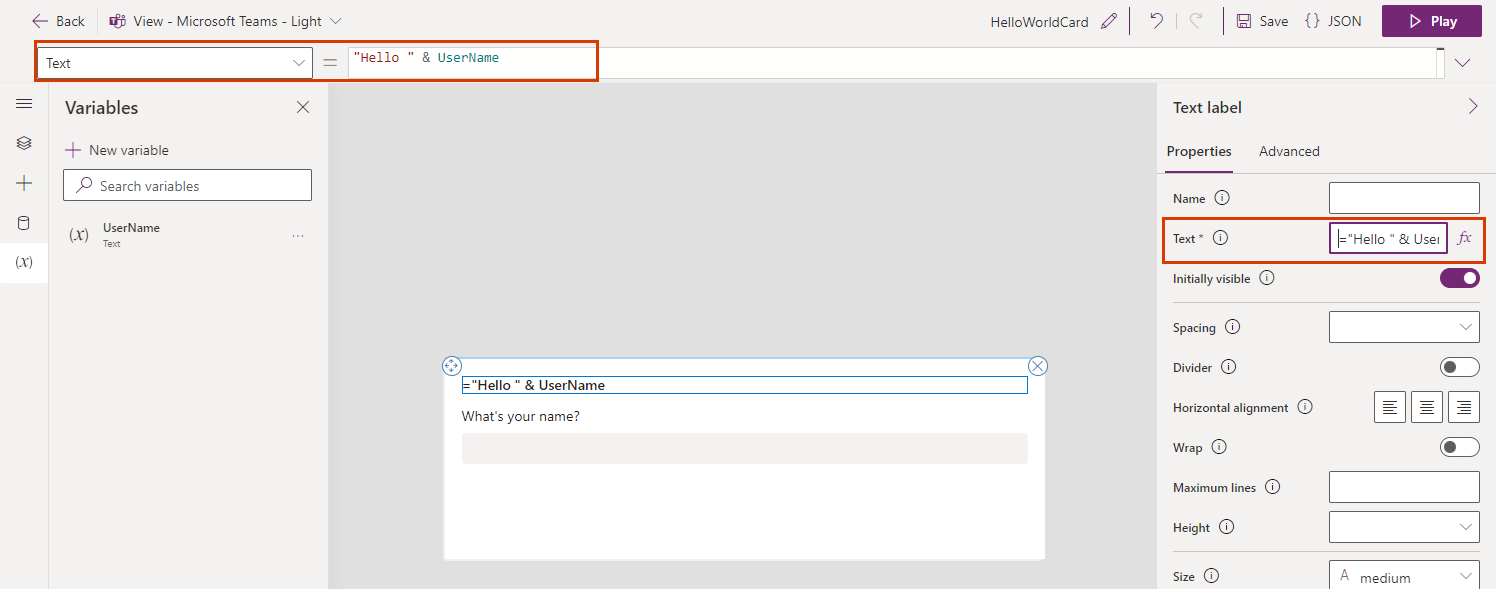
We need another variable to add the user's input to the card title, so let's create one now.

1. In the left pane, select Variables.
2. Select + New variable.
3. In the New variable window, enter UserName under Name and enter No Name under Default value. Leave the other values as they are.
4. Select Save.



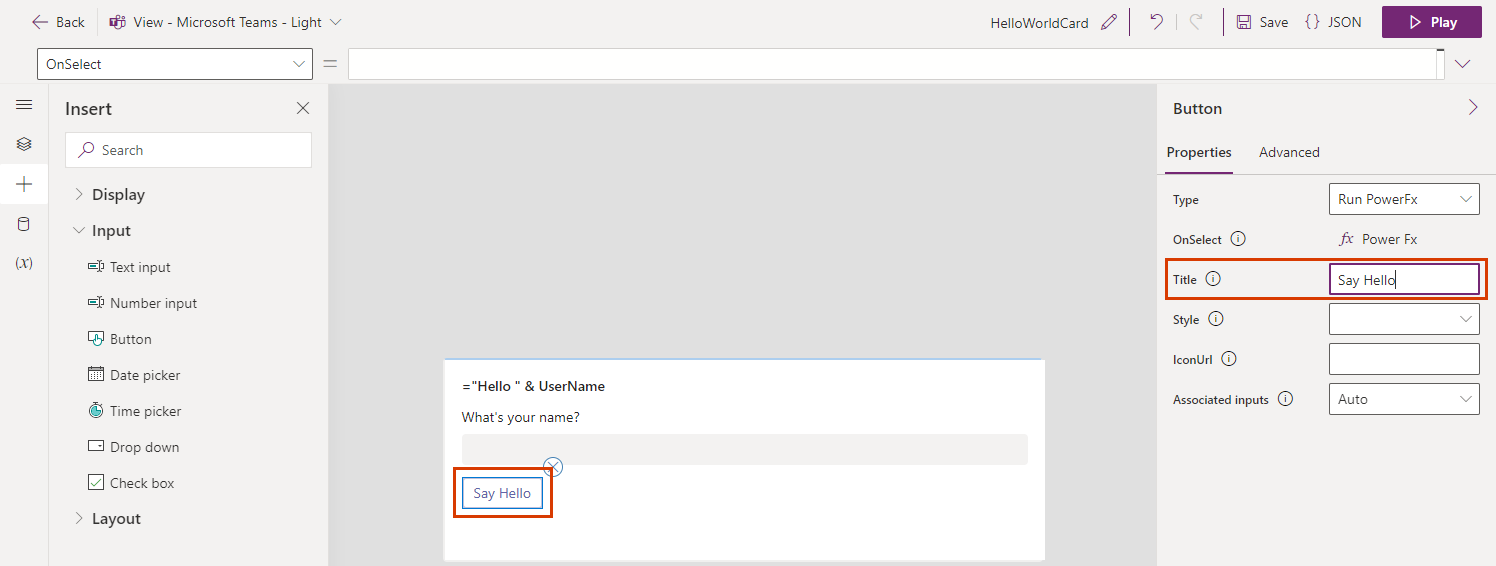
1. Select the card title, which is a text label control. In the control's properties pane, set Text to ="Hello " & UserName.

This expression changes the card title to the string "Hello " followed by the value of the variable you created. You can enter the expression in the formula bar or the properties pane.



**Add a Power Fx button**

1. In the left pane, select Insert.
2. In the tool pane, select Input to expand the category, and then select Button.
3. In the button properties pane, set Title to Say Hello.



1. In the properties pane, select PowerFx to place your cursor in the formula bar.
2. Type the following Power Fx expression in the formula bar: Set(UserName, UserAnswer)

This expression assigns the value of the user's input, UserAnswer, to the UserName variable you referred to in the card title, when the button is selected. Another way to read the expression is, "Set the value of the variable UserName equal to the value of UserAnswer." Since the expression is bound to the button's OnSelect property, it runs when the user selects the button.



**Test the card**

1. You should always save your changes before you play a card. Select Save, and then select Play.
2. Test your card a few times with different inputs. Make sure your input replaces the default value "No Name" in the card title each time.

**Create a shopping list card**

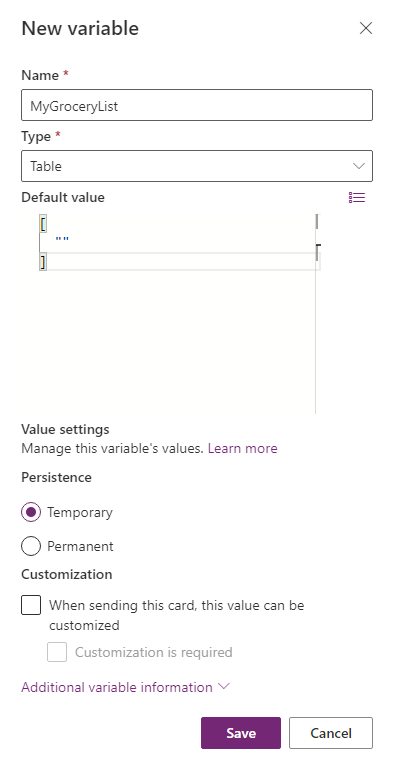
**Create a card**

1. Sign in to Power Apps and select your environment.
2. In the left pane, select Cards. If the item isn’t in the side panel pane, select …More and then select the item you want.
3. Select + Create a card.
4. Under Card name, type SimpleShoppingCard, and then select Create.
5. Select the text Your card title goes here. In the text label properties pane, set Text to Shopping List.
6. Select the text Add and remove element to customize your new card. In the text label properties pane, set Text to Use the box below to add items to the list.

**Add a variable**

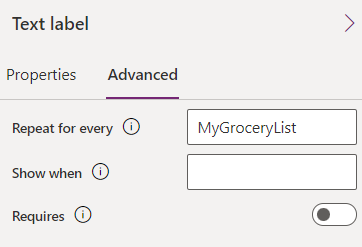
The shopping list will hold individual items in separate lines of text. It looks like a table with a single column, with one grocery item in each row. That suggests you'll need to create a table variable to store your list.

1. In the left pane, select Variables.
2. Select + New variable.
3. In the New variable window, enter MyGroceryList under Name. Set Type to Table.
4. Select the curly brackets to the right of Default value and enter "" between the brackets. This indicates that our table holds text values in a column implicitly called Value
5. Select Save.



**Add a list to the card**

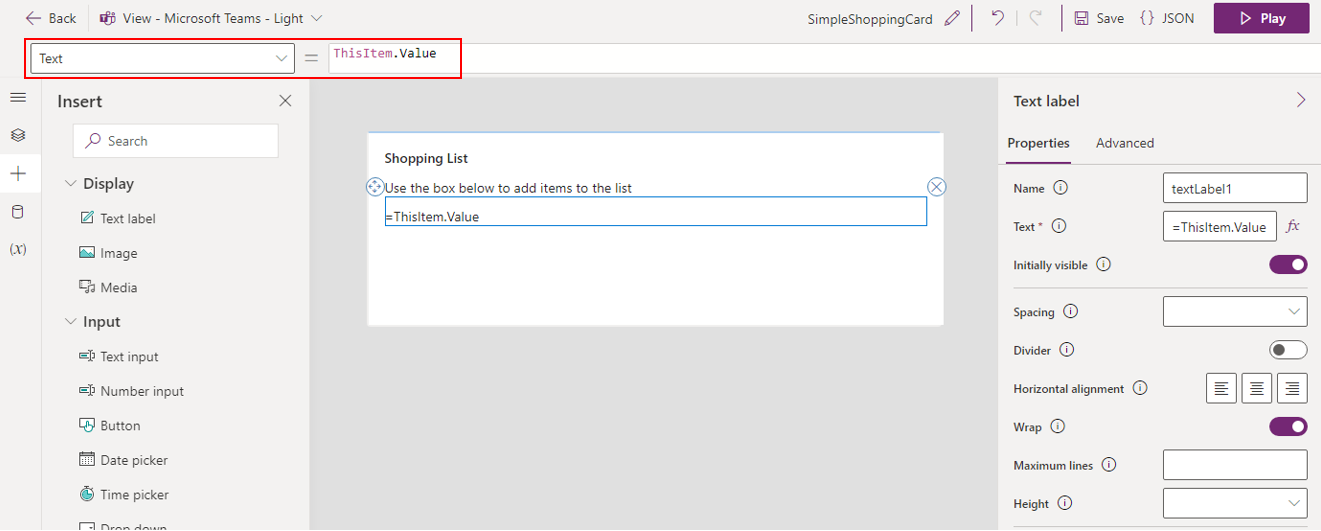
1. In the left pane, select Insert.
2. In the tool pane, select Display to expand the category, and then select Text label.
3. In the text label properties pane, select the Advanced tab.
4. Set Repeat for every to MyGroceryList.



Setting a text label's Repeat for every property repeats the text label for every item in the specified table. In this example, the table, MyGroceryList, is the variable you created. In other words, a separate text label is created in the card for every item in the grocery list.

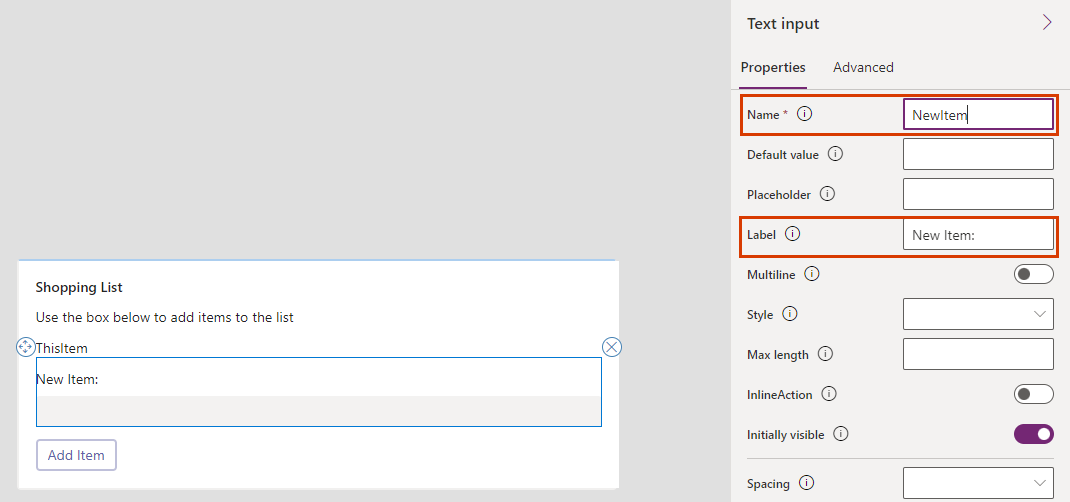
1. Select the Properties tab. Set Text to ThisItem.Value.

Assigning the system-defined variable ThisItem.Value to the text label displays the value of the current item in the MyGroceryList array as the label text. Array is another term for a table variable. You can enter the variable name in the formula bar or the properties pane.



**Add an input box**

1. The list you created is empty, so in the final step, you'll give the user the ability to add items.
2. In the left pane, select Insert.
3. In the tool pane, select Input to expand the category, and then select Text input.
4. In the text input properties pane, set Name to NewItem and set Label to New Item:.



The Name property lets you refer to the control in a Power Fx expression. It has to be one word, with no spaces or special characters. The Label property is displayed in the card. The name and label of a control don't have to be similar, as they are here, but using similar names makes it easier to keep track of controls.

**Add a Power Fx button**

1. In the left pane, select Insert.
2. In the tool pane, select Input to expand the category, and then select Button.
3. In the button properties pane, set Title to Add Item.
4. In the properties pane, select PowerFx to place your cursor in the formula bar.
5. Type the following Power Fx expression in the formula bar: Collect(MyGroceryList, {Value: NewItem})

This expression uses the Power Fx function Collect to append the value of the user's input, NewItem, to the MyGroceryList table variable in the implicit Value column. Since the expression is bound to the button's OnSelect property, it runs when the user selects the button.

Screenshot of the Power Fx expression in the formula bar, assigned to the button's OnSelect property.

**Test the card**

1. You should always save your changes before you play a card. Select Save, and then select Play.
2. Test your card by adding a few items to the grocery list.

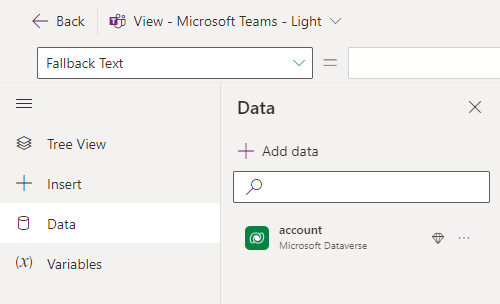
**Create a card with data from Dataverse**

**Create a card**

1. Sign in to Power Apps and select Cards. If the Cards tab is not visible, select More and pin the Cards tab.
2. Name the card DataverseCard and then select Create.

**Connect the card to the Dataverse account table**

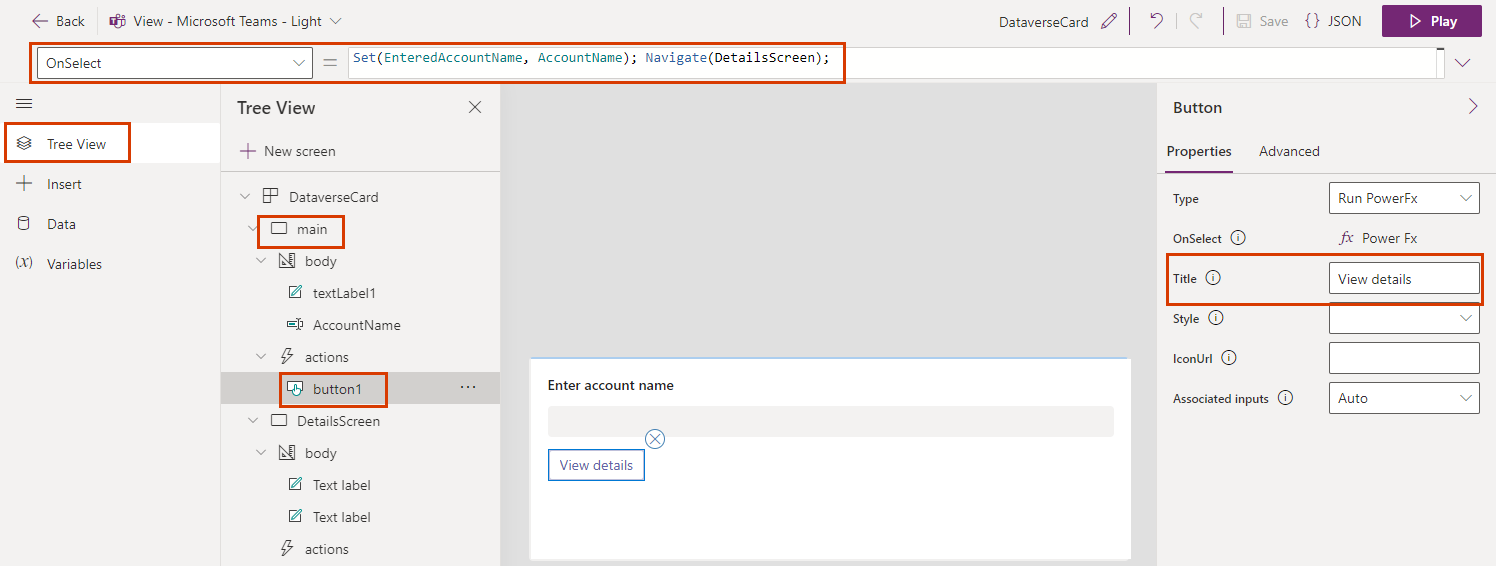
1. Select Data > + Add data.
2. Search for Dataverse and then select the Microsoft Dataverse connector.
3. Select the account table, and then select Select.



**Ask for the account name**

1. Insert a text label control and set its Text property to Enter account name.
2. Insert a text input control and set its Name property to AccountName.
3. Insert a button and set its Title property to View details.
4. Select Variables > + New variable.
5. Set Name to EnteredAccountName. Leave all other values as they are. Select Save.
6. Open the Tree View and select + New screen.
7. Name the screen DetailsScreen and then select Create.
8. In the Tree View, select the main screen.
9. Select the button and set its OnSelect property to Set(EnteredAccountName, AccountName); Navigate(DetailsScreen);

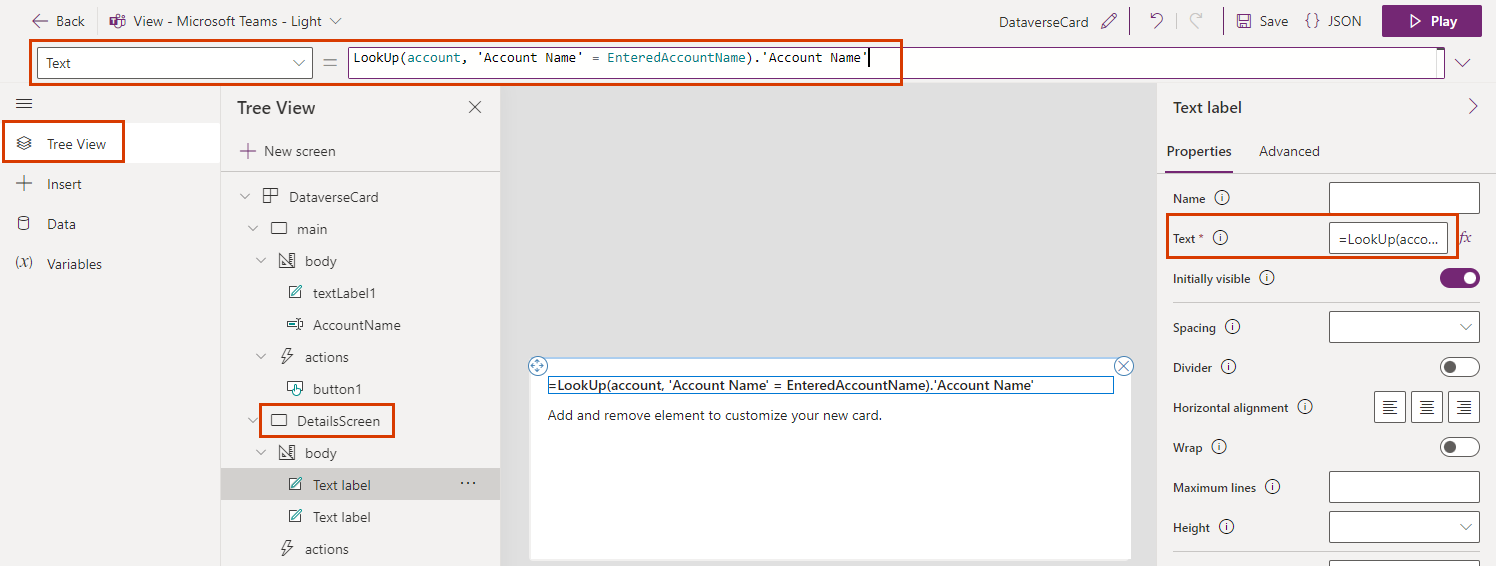
This expression has two parts, separated by a semicolon (;). The first part assigns the value of the user's input, AccountName, to the EnteredAccountName variable. The second part opens the screen named DetailsScreen. Since the expression is bound to the button's OnSelect property, it runs when the user selects the button.



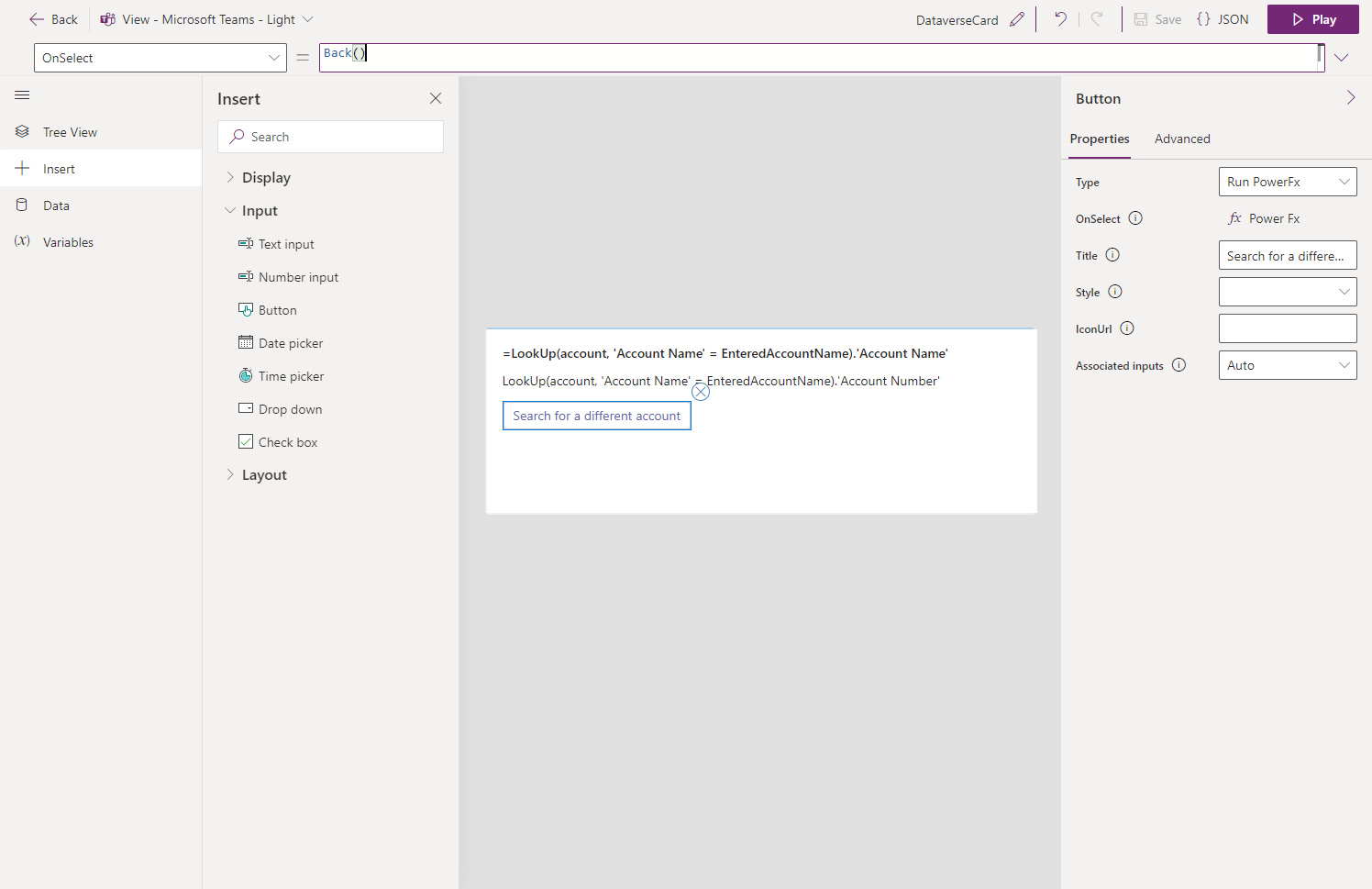
**Show account details**

1. In the Tree View, select the DetailsScreen screen.
2. Select the card title and set its Text property to LookUp(account, 'Account Name' = EnteredAccountName).'Account Name'.

This expression changes the card title to a string incorporating the account name. You can enter the expression in the formula bar or the properties pane.

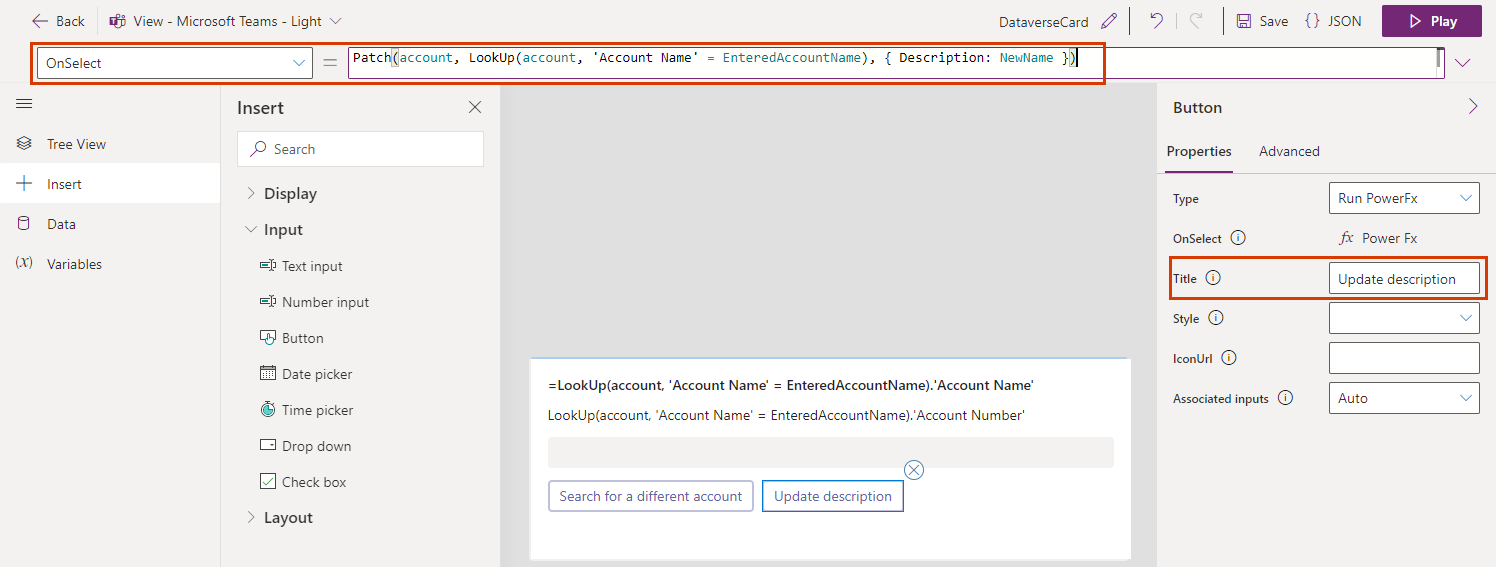


1. Select the second text label and set its Text property to LookUp(account, 'Account Name' = EnteredAccountName).'Account Number'.
2. Insert a button. Set its Title property to Search for a different account and its OnSelect property to Back().



**Change the account description**

1. Insert a text input control and set its Name property to NewName.
2. Insert a button. Set its Title property to Update name and its OnSelect property to Patch(account, LookUp(account, 'Account Name' = EnteredAccountName), { Description: NewName }).



**List all accounts**

1. In the Tree View, select the main screen.
2. Insert a text label. Set its Text property to ThisItem.'Account Name' and set its Repeat for every advanced property to account.

**Test the card**

1. You should permanently save your changes before you play a card. Select Save, and then select Play.
2. Test all the controls on both screens on your card. The first screen should list all the accounts and allow you to type the name of an account to open a screen with more details. The second screen should show the account name and number and enable you to change the account name.

Power Apps Containers

Containers make it easier to design the layout of an app. When a control (e.g. a button) is placed inside of a container it becomes automatically positioned. It removes the error-prone positioning of drag-and-drop and the tedious nature of manually writing X & Y position properties for individual controls.

There are 3 types of containers:

* Vertical Container: this container stacks controls up-and-down
* Horizontal Container: this container positions controls side-by-side from left-to-right
* “Container”: this container allows controls to be positioned on top of one another.

A screenshot of a computer

Description automatically generated

Vertical Container & Horizontal Container Properties

1. Direction Property

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Justify Property

A screen shot of a computer

Description automatically generated

A yellow rectangular boxes with black text

Description automatically generated

1. Align Property

A screenshot of a cell phone

Description automatically generated

This title bar uses End for its align property to position its label and icons along the bottom of the container. Leaving space at the top is desirable for iPhones which have a camera notch there.

A red rectangle with white text

Description automatically generated

1. Gap Property

A screenshot of a computer game

Description automatically generated

A screenshot of a computer screen

Description automatically generated

1. Overflow Property

Overflow defines what happens when the contents of the container exceed the size of the container. There are two options: Hide and Scroll. Hide does not show any content beyond the limits of the container.

A yellow square with black text

Description automatically generated

Scroll allows the user to browse past the limits of the container using a scrollbar.

A screenshot of a computer

Description automatically generated

1. Wrap Property

Wrap positions controls on the next line when the container is smaller than its contents. It is useful when used with responsive apps that must be displayed on devices with various sizes.

A screenshot of a computer

Description automatically generated

Data Validation

You can perform the PowerApps validation on submit, but as we mentioned before we can validate PowerApps when the user leaves the current field.

We will work on a SharePoint list with these columns:

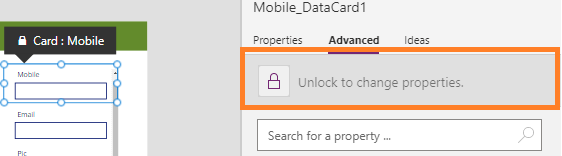
* Employee Name -Requierd field
* Employee Id – must be a number
* Email Address -must match email form.
* Age- must be more than 18
* Phone- must match specific expression
* Required Field Validation In PowerApps

A screenshot of a computer

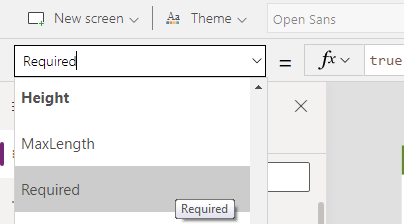
Description automatically generated

**For implementing PowerApps validation on submit while working with cards**

1. Select the card
2. From the advanced properities choose to unlock to change properties



1. On the Required property of the card change it to be true



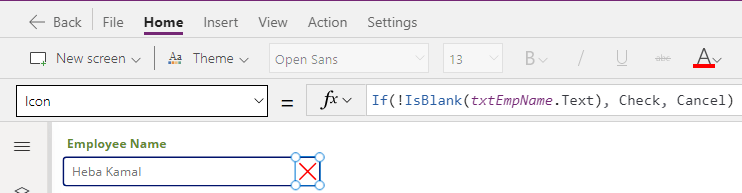
this is the powerapps required field validation on submit.

**For implementing PowerApps validation before submit**

In my example, I will insert an icon to show the user if there are any errors or not and I will write the error in the tooltip of the icon.

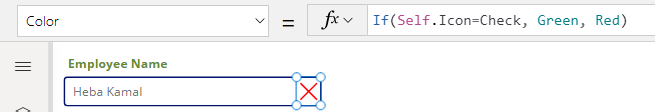
1. Insert an Icon in the employee name textbox
2. In the icon property write this formula

If(!IsBlank(txtEmpName.Text), Icon.Check, Icon.Cancel)



1. In the color property write this formula

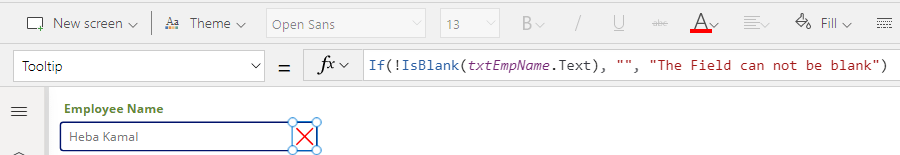
If(Self.Icon=Icon.Check, Green, Red)

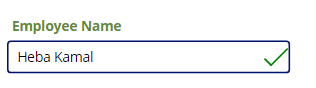


1. In the tooltip property write this formula

If(!IsBlank(txtEmpName.Text), "", "The Field can not be blank")

You can apply this step on a label insteed of the tooltip as we mentioned before

Now when the field is black the icon will be the X icon with the color red otherwise the icon will be the check icon with green color.



This is the difference between powerapps validation on submit and powerapps validation before submit.

**PowerApps Number Validation**

To validate a textbox to Accept only numbers:

1. Select the textbox
2. Make the Format property to be number
3. Also you can use the IsNumeric function to check if the value entered in the textbox is number or not.

IsNumeric( txtEmpID.text)

**PowerApps Validate Mobile Number**

1. Select the mobile textbox
2. Make the Format property to be number to allow only digits
3. Insert an icon and use the same method we used in the Employee Name

In the Icon property set the formula to allow only 10 digits

If(

IsMatch(

txtmobile.Text,

Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit

),

Icon.Check,

Icon.Cancel

)

and in the color property set the formula to be

If(Self.Icon=Icon.Check, Green, Red)

you can add an error message to the user either on a lable or on the tooltip property of the icon as shown bellow

If(

IsMatch(

txtmobile.Text,

Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit & Digit

),

"",

"the mobile must be 10 digit"

)

**Zip Code Field Validation**

The PowerApps Zip Code field Validation is the same as we have done in the previous example except for the number of the digits

**Validating Age Field**

In this PowerApps validation example, we’ll look at how to validate a number field in PowerApps to be > or < specific number.

let’s check if the age field is more than 21 years old.

1. Select the Age textbox
2. Make the Format property to be number to allow only digits
3. Insert an icon and use the same method we used in the Employee Name
4. In the Icon property set the formula to be

If(Value(txtAge.Text) >= 21, Icon.Check, Icon.Cancel)

and in the color property set the formula to be

If(Self.Icon=Icon.Check, Green, Red)

you can add an error message to the user either on a lable or on the tooltip property of the icon as shown below

If(Value(txtAge.Text) >= 21, "", "Age must be >=21")

**PowerApps Email Validation**

Another example of PowerApps validation before submit is validating Email address field. An email address must be in the format example@ example.com.

There is a predefined matching pattern already exists for email addresses that you can use

1. Select the Email Address textbox
2. Insert an Icon as we used before in the employee name .

In the Icon property set the formula to be

If(

IsMatch(txtEmail.Text, Match.Email),

Icon.Check,

Icon.Cancel

)

1. In the color property write this formula

If(Self.Icon=Icon.Check, Green, Red)

you can add the validation error message to the user either on a lable or on the tooltip property of the icon as shown bellow

If(

IsMatch(

txtEmail.Text,

Match.Email

),

"",

"the Email should be in the formate example@example.com"

)

**Data Validation With Pattern Matching**

1. What if we want to validate PowerApps textbox to be in a specific format, For example, we want to formate the phone number to be n the format ###-###-#### then the formula of the Icon property should be

If(

IsMatch(

txtPhoner.Text,

Match.Digit&Match.Digit&Match.Digit&"-"&

Match.Digit&Match.Digit&Match.Digit&"-"&

Match.Digit&Match.Digit&Match.Digit&Match.Digit

),

Icon.Check,

Icon.Cancel

)

1. and in the color property set the formula to be

If(Self.Icon=Icon.Check, Green, Red)

1. you can add the validation error message to the user either on a lable or on the tooltip property of the icon as shown bellow

If(

IsMatch(

txtphone.Text,

Match.Digit&Match.Digit&Match.Digit&"-"&

Match.Digit&Match.Digit&Match.Digit&"-"&

Match.Digit&Match.Digit&Match.Digit&Match.Digit

),

"",

"the Phone should be in the format ###-###-#### "

)

**PowerApps Validate Date**

1. Are you trying to validate the Date field in PowerApps? This PowerApps validation example shows how to validate a date field in PowerApps. The user must choose a weekday from Monday to Friday and must be a day in the future.
2. Insert an icon and in the icon property write the followin formula

If(

Weekday(joinDate.SelectedDate, StartOfWeek.Monday) <= 5

And joinDate.SelectedDate > Today(),

Icon.Check, Icon.Cancel

)

1. and in the color property set the formula to be

If(Self.Icon=Icon.Check, Green, Red)

1. you can add the validation error message to the user either on a lable or on the tooltip property of the icon as shown bellow

If(

Weekday(joinDate.SelectedDate, StartOfWeek.Monday) > 5,

"Choose a day from Monday to Friday",

joinDate.SelectedDate <= Today(),

"Must choose a date in the future",

"No date was selected"

)

**Validating Password In PowerApps**

1. Select the Password textbox
2. Insert an Icon as we used before in the employee name .
3. In the Icon property set the formula to be

Icon = If(

IsMatch(

txtPassword.Text,

"^(?=[^\d\_].\*?\d)\w(\w|[!@#$%]){7,20}

"

),

Check,

Icon.Lock

)

this validates a strong password which can contain eight, nine or 10 characters with the addition of at least one digit and at least one alphabetic character and no special charachters.

1. In the color property write this formula

If(Self.Icon=Icon.Check, Green, Red)

1. you can add the validation error message to the user either on a lable or on the tooltip property of the icon as shown bellow

If(

IsMatch(

txtEmail.Text,

Match.Email

),

"",

"the Email should be in the formate example@example.com"

)

Co

Creating Power Apps from Figma

1. go to figma.com and download and install the figma desktop app🡪go to community page🡪search create app from figma🡪select the Microsoft-owned toolkit.

A screenshot of a social media account

Description automatically generated

1. click the ‘open in figma’ button to start creating the app
2. once back in figma, click the Assets pane to view the controls
3. drag and drop a phone screen

A screenshot of a computer

Description automatically generated

1. before adding controls, detach the screen instance. Rclick the screen🡪detach instance
2. add a form screen over the initial screen then rclick🡪detach instance

A screen shot of a phone

Description automatically generated

1. set colors
2. add horizontal data card (‘Header’)
3. add vertical data card (‘Text Input’)
4. add vertical data card (‘radio button’)
5. add button (must be outside the form!)

A screenshot of a computer

Description automatically generated

1. go to power apps home 🡪create app from figma
2. set app name
3. the link to figma app/frame can be obtained from the figma app (rclick🡪copy/paste as link)
4. the personal access token can be obtained from your figma account🡪click on your account avatar🡪settings🡪personal access token🡪

A screenshot of a computer

Description automatically generated

1. you can then create the app and start building the workflow in power app.