

Power Apps Standards

Introduction

Welcome to the Power Apps Coding Standards and Development guidelines for Canvas Apps.

In this guide you will find coding rule, guidelines, Form Design, UI&UX Guidelines and best practices we used every day to create Power Apps Canvas apps.

Microsoft already share some best practises to develop power app, but this are some additional information.

Table Of Contents

1. Naming Conventions
 - Screen Names
 - Control Names
 - Variable Names
 - Collection Names
 - Datasource Table Names
2. Variable Types
 - Variable Scope
3. Commenting Code
 - Commenting Style
 - Line Comments vs. Block Comments
6. App Theming
 - Creating An App Theme
 - Theming Variables Sample Code

7. Responsive

8. Forms

- Form Responsiveness
- Form Layout
- Validation and error messages
- Alignment and spacing
- Protect against loss of Unsaved Data
- Use A single form to Create, Edit and View
- Re-use the same form
- Design empty state

9. Gallery Design & UX Guidelines

- Design Empty States

10. Error-Handling

- Enable Formula-Level Error Management
- Patch Function Error-Handling
- Power Apps Forms Error-Handling
- Power Automate Flow Error-Handling
- IfError Function
- Handling Unexpected Errors

11. Optimizing Performance

- Load Multiple Datasets Concurrently
- Write Formulas That Use Delegation
- Cache Data In Collections And Variables
- Limit The Size Of Collections
- “Batch Patch” Multiple Changes To A Datasource Table At Once
- Reduce Code In The App’s OnStart Property

- Minimize Number Of Controls On A Single Screen
- Enable DelayOutput For Text Input Controls
-

12. Improving Code Readability

- Apply Automatic Formatting
- Use The WITH Function
- Flatten Nested IF Functions
- Have A Consistent Naming Convention For Controls & Variables
- Join Text Strings And Variables
- Choose Consistent Logical Operators
- Remove IF Statements When The Result Is A True Or False Value
- Simplify Logical Comparisons When Evaluating A Boolean
- Substitute The Self Operator For The Current Control Name
- Alphabetize Patch & UpdateContext Function Arguments

13. Typography

- Basic
- Title
- Navigation
- Su-Content
- Button

Naming Conventions

Screen Names

A screen name should clearly describe its purpose in 2-3 words ending with word “Screen.” Use proper case. A screen-reader will speak the screen name to visually impaired users when the screen loads.

Good Examples	Bad Examples	Bad Reason
Appointments Screen	Appointments	Missing the word ‘Screen’
Order Form Screen	OrderFormScreen	Not friendly to a screen reader
Collect Signature Screen	scrCollectSignature	Not friendly to a screen reader

Control Names

A control name should show the control-type, the purpose and the screen. Use camel-case and underscores for spacing. For example, the control named `txt_OrderForm_FirstName` is a text input that captures first name on the app’s Order Form Screen.

Good Examples	Bad Examples	Bad Reason
<code>drp_NewEmployee_Department</code>	<code>drpDepartmentNewEmployee</code>	No spacing
<code>btn_OrderForm_Submit</code>	<code>btn_Submit_OrderForm</code>	Wrong order
<code>gal_Home_Appointments</code>	<code>gly_Home Appointments</code>	Non-standard control prefix

Variable Names

A variable name should show the scope of the variable and its purpose. Use camel-case with no spaces between each word. For example, the variable *gblUserEmail* is a global variable which holds the current user's email address.

Good Examples	Bad Examples	Bad Reason
gblUserCurrent	UserCurrent	No scope
locPacksInBoxQuantity	Loc_Packs_In_Box_Quantity	Improper capitalization and spacing
LocIsLoading	locBoolLoading	Do not use data types in variable names
varWorkdaysDuringVacation	varWorkdays	Not descriptive enough

Collection Names

A collection name should contain the original datasource and describe its purpose. Use camel-case with no spaces between each word. For example, the collection *colDvInvoices* is a collection of invoices from Dataverse.

Good Examples	Bad Examples	Bad Reason
colSpEmployees	colEmployees	No datasource
colDvSalesLeads	coldv_salesleads	Improper capitalization and spacing
colNavigationMenu	NavigationMenu	Do not use data types in variable names

A standard list of datasource abbreviations can be found below:

Original Datasource	Abbreviation
Dataverse	Dv
SharePoint	Sp
SQL	Sql
Salesforce	Sf
None (created in-app)	(none)

Datasource Table Names

A datasource created by the developer should have 1-3 words to describe its purpose. Use the singular form of the word and proper-case. Be as concise and clear about the purpose of the datasource as possible.

Good Examples	Bad Examples	Bad Reason
Employee	Emp	Abbreviation instead of full word
Construction Project	Projects	Too general, what type of projects?
Repair Order	RepairOrders	No spacing, plural

Variable Types

Variable Scope

A [variable's](#) scope determines where it can be referenced in the app. If the variable is required on multiple screens use a global variable. Otherwise, use a local or context variable instead. Choose the proper variable type by determining its scope.

Variable Type	Declaration Method	Scope
Global	Set Function	Variable is available across all app screens
Local	UpdateContext Function	Variable is only available on a single app screen
One-time	With Function	Variable is only available inside the WITH function

Commenting Code

Commenting Style

Use these commenting conventions to ensure a consistent style:

- Place comments on a separate line above the code section they are describing.
- Do not write in-line comments beside on the same line as a piece of code.
- Start comments with a capital letter
- End comment text with a period

```
// Validate the work order to ensure it will not be rejected upon submission.
Set(
    varValidateForm,
    Validate(
        'Work Orders',
        Defaults('Work Orders'),
        gblRecordWorkOrderCurrent
    )
);
```

Format text Remove formatting Find and replace

Line Comments vs. Block Comments

Power Apps has two comment styles: line comments and block comments. Line comments are made on a single-line and block comments can be made across multiple lines.

Comment Style	Syntax	Example
Line	// [comment goes here]	// Validate the work order to ensure it will not be rejected upon submission.
Block	/* [comments go here] */	/* Work Order Details Screen: – Uses a single form to create, edit and view a record to minimize the number of controls in the app. – Emails a signed PDF to the employee’s manager after the form is submitted so it can be stored as backup. */

Theming Guidelines

Creating An App Theme

In order to achieve a consistent style throughout all screens. Setup theming variables in the app's [OnStart property](#) and manually apply them to each control type.

Variable Name	Purpose
gblAppColors	Color palette for the app
gblAppFonts	Heading fonts, body fonts and sizes used in the app
gblAppIcons	SVG icons used in the app
gblAppDefaults	Default values for common control properties

Keep a copy of each styled control on a hidden screen. It is more efficient to re-use controls instead of setting up a new control with a style every time.

Theming Variables Sample Code

Use this code in the OnStart property of an app to define its theme.

```

// App Color Palette
Set(
  gblAppColors,
  {
    // Primary Colors
    Primary1: ColorValue("#30475E"),
    Primary2: ColorValue("#F05454"),
    Primary3: ColorValue("#222831"),
    Primary4: ColorValue("#DDDDDD"),

    // Accent Colors
    Black: ColorValue("#000000"),
    Cyan: ColorValue("#17A2B8"),
    Green: ColorValue("#28A745"),
    Orange: ColorValue("#FD7E14"),
    Red: ColorValue("#DC3545"),
    Teal: ColorValue("#20C997"),
    White: ColorValue("#FFFFFF"),
    Yellow: ColorValue("#FFC107"),

    // Neutral Colors
    GrayDark: ColorValue("#484644"),
    GrayMediumDark: ColorValue("#8A8886"),
    GrayMedium: ColorValue("#B3b0AD"),
    GrayMediumLight: ColorValue("#D2D0CE"),
    GrayLight: ColorValue("#F3F2F1")
  }
);

```

```

// App Fonts & Sizes
Set(
  gblAppFonts,
  {
    Heading: "Roboto, Open Sans",
    Body: "Lato",
    Size: {
      Tiny: 10,
      Regular: 13,
      Subtitle: 16,
      Title: 20,
      Huge: 28
    }
  }
)

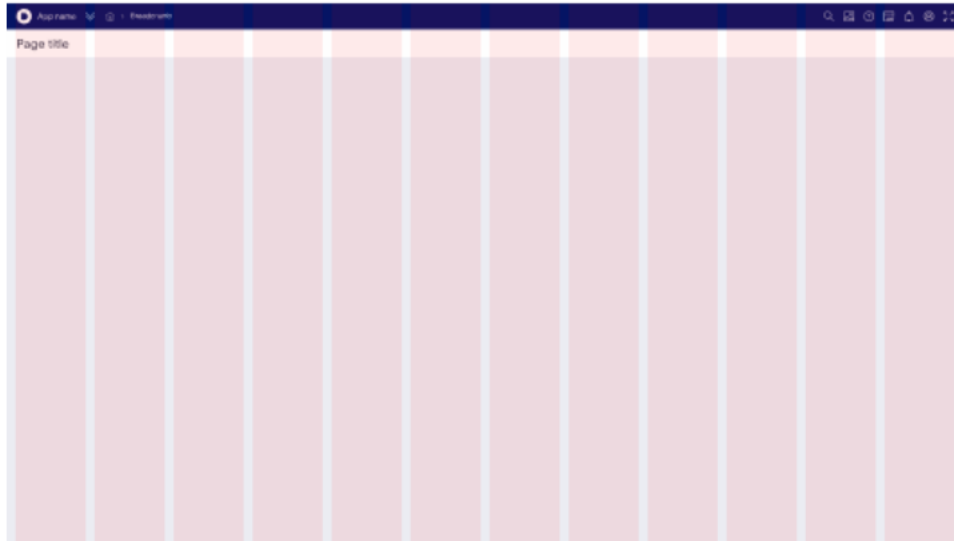
// App Icons
Set(
  gblAppIcons,
  {
    // SVG icon code is stored in an 'Import from Excel' table named AppIcons
    Checklist: LookUp(AppIcons, Name="Checklist", DataURI),
    Checkmark: LookUp(AppIcons, Name="Checkmark", DataURI)
  }
);

```

Responsive

Make sure to follow our responsive guidelines by using the appropriate grid for your resolution.

12 Column grid for resolution



Text Box, Dropdown, Calander:

Provide a hint text only if there is a special requirement for the text to be entered.

Height – 30

Width – Depend on Screen Resolution

The image shows three distinct input field styles. The first is a date picker containing the text '2/2/2024' and a blue calendar icon. The second is a dropdown menu, represented by a white rectangular box with a dark grey downward-pointing arrow on the right side. The third is a simple, empty white rectangular text box.

Checkbox

Use sentence case for checkbox captions. Refer to the [Capitalization](#) section in the Microsoft Writing Style Guide for the correct usage of sentence case.

☐ Checkbox label

Forms

A form is used to present data to the user, and to allow users to enter data in a logical way. With a form, properties and input fields can be easily organized intuitively in a structured manner to help guide users through content.

Form title

Personal info

Name

Email

Date of birth

Well details

Well name

Well name

Select

Size

☐ Small

☐ Large

☒ Medium

☐ Extra large

Size

☐ Red

☐ Blue

☒ Green

☒ Orange

Group title

Name

Email

Date of birth

Form Responsiveness

Always assign the responsive grid layout for your form.

By using the responsive grid layout, the form offers a responsive layout based on a 12-column grid.

Form Layout

Label on top of input field

Forms are built with the label on the top of the input. This option is better for scanning and accessibility.

While it offers better completion rates, it is less ideal for longer forms.

Label on top

A diagram showing a form with two input fields. The first field is labeled 'Email' and the second is labeled 'Password'. Both labels are positioned directly above their respective input boxes. A blue line points from the text 'Label on top' to the 'Email' label.

Units display

When displaying units in forms, we recommend displaying the units alongside the input label enclosed in parentheses. The parenthesis create a visual break in the text allowing users to scan the form faster while understanding the specific unit applied per control.

A diagram showing a form with three input fields. The first field is labeled 'Length (ft)', the second is labeled 'Width (ft)', and the third is labeled 'Depth (ft)'. Each label is positioned directly above its respective input box. Below each input box is the text 'Hint text'.

If the label is placed to the left of the input box, we recommend following the same unit guideline.

A diagram showing a form with three input fields. The first field is labeled 'Length (ft)', the second is labeled 'Width (ft)', and the third is labeled 'Depth (ft)'. Each label is positioned to the left of its respective input box. Below each input box is the text 'Hint text'.

Form title

Personal info

Name

Email

Date of birth

Group title

Name

Email

Date of birth

Group title

Name

Email

Date of birth

The diagram illustrates a form layout with three groups of elements. Each group is headed by a 'Group title' and contains three items, each with a 'Label' and a light blue rounded rectangle. Dashed blue lines connect the elements across the groups, showing a flow from left to right. The first group has a solid blue dot in the first rectangle. The second group has a calendar icon in the third rectangle. The third group has a solid blue arrow and a calendar icon in the third rectangle.

Inputs can also be grouped horizontally in a single column creating sub-columns within the form.

Label

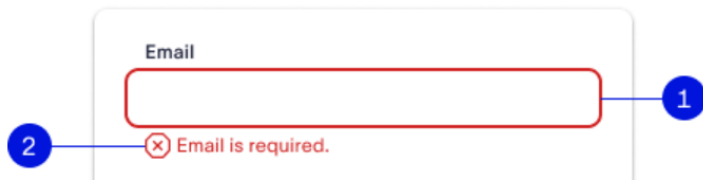
Well name

Type

Select ▼

Validation and error messages

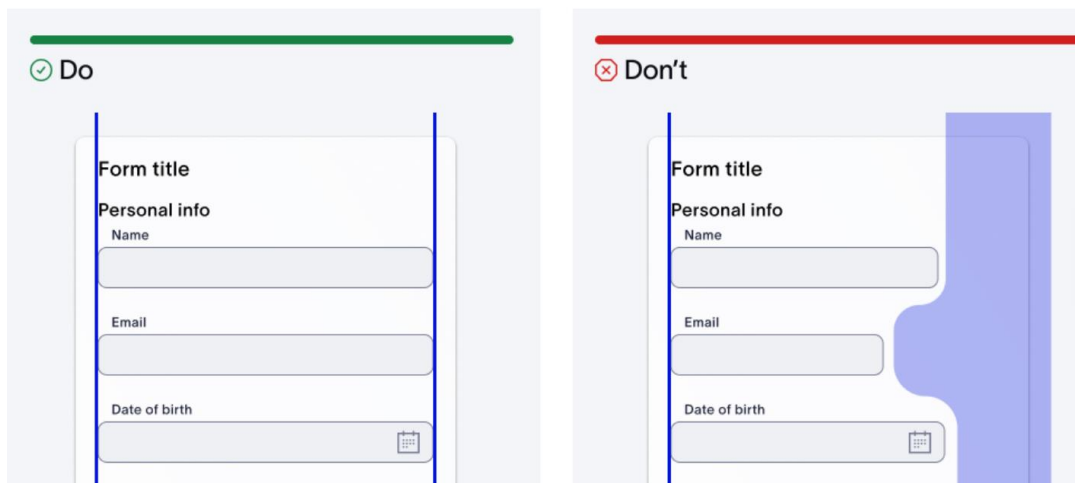
Use validation and error messages to indicate when a form or field submission fails or requires additional information to be shown.



Alignment & spacing

Forms are built on a 4px grid, every element has been designed to fit within this grid.

Forms should be aligned to the left and right, avoid inconsistent spacing in inputs as the white space demands more attention from the user, thereby resulting in slower form completion rates.

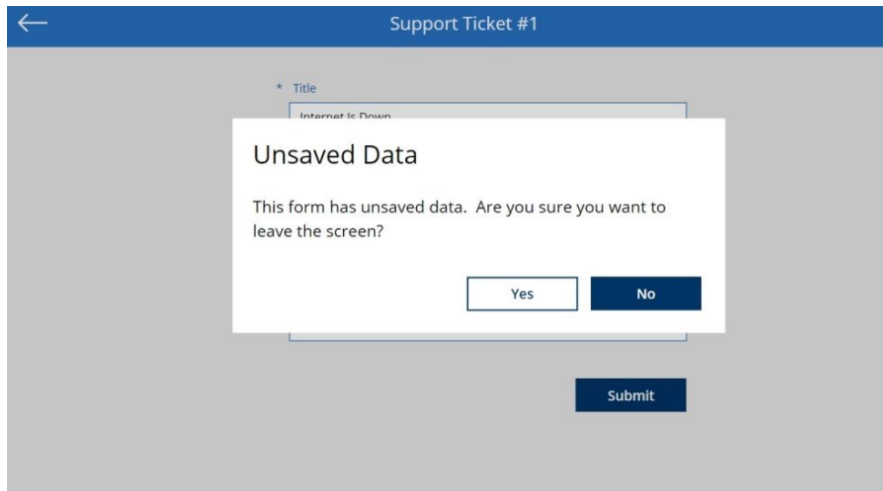


20px: Padding around form and container. Also, the spacing between a form title and next element. And the spacing between columns.



Protect Against Loss Of Unsaved Data

Prevent users from accidentally exiting a form and losing unsaved data. Before a user leaves the screen prompt them for confirmation. Display this message in a pop-up menu: “This form has unsaved data. Do you really want to close the form?” and present the choices OK or Cancel.



Use A Single Form To Create, Edit & Display Records

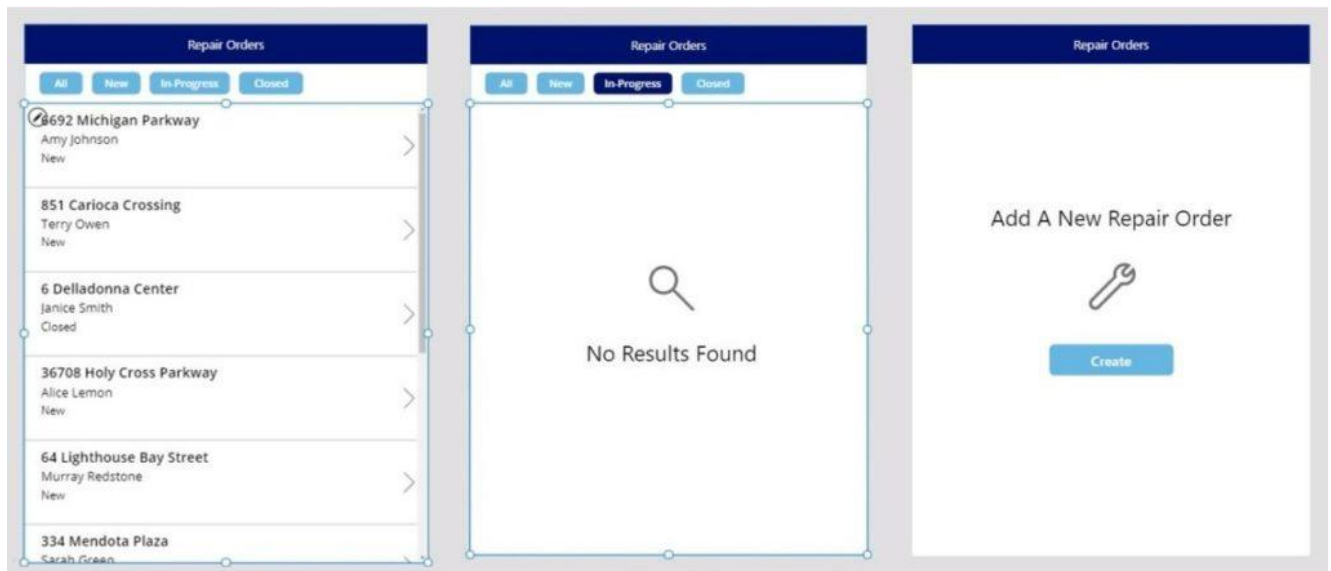
Re-use the same form to Create, Edit and View in view only mode. Having only one form reduces development & maintenance time and ensures consistency. Change the mode of a Power Apps form using the NewForm, EditForm and View form functions. Patch forms require additional code to change the DisplayMode of each individual control manually.

New Mode	Edit Mode	View Mode
<div><p>Restaurant Inspections</p><p>Restaurant Name</p><input type="text"/><p>Street Address</p><input type="text"/><p>Inspection Date</p><input type="text"/><p>Passed Or Failed</p><input type="text"/><p>Submit</p></div>	<div><p>Restaurant Inspections</p><p>Restaurant Name</p><input type="text" value="Clementine Cafe"/><p>Street Address</p><input type="text" value="120 North Haverbrook Drive"/><p>Inspection Date</p><input type="text" value="9/21/2021"/><p>Passed Or Failed</p><input type="text" value="Passed"/><p>Submit</p></div>	<div><p>Restaurant Inspections</p><p>Restaurant Name</p><p>Clementine Cafe</p><p>Street Address</p><p>120 North Haverbrook Drive</p><p>Inspection Date</p><p>9/21/2021</p><p>Passed Or Failed</p><p>Passed</p></div>

Gallery Design & UX Guidelines

Design Empty States

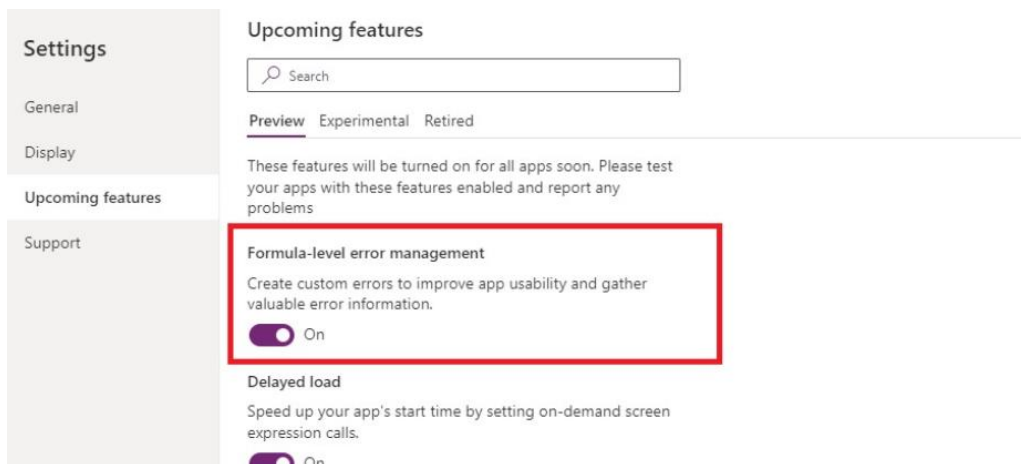
Include an *empty state* that appears when a gallery has no data. The empty state should tell the user why the gallery is empty and/or give directions on what actions to take next.



Error Handling Guidelines

Enable Formula-Level Error Management

Open Power Apps advanced settings and turn on formula-level error management. This setting enables use of the [IfError function](#), the [IsError function](#).



Patch Function Error-Handling

Check for errors anytime data is written to a datasource with the [Patch function](#) or [Collect Function](#). Even if the submitted record(s) are valid, good connectivity and the correct user permissions cannot be assumed. This is not necessary for local collections stored in memory.

```
// Create a new invoice
If(
    IsError(
        Patch(
            'Invoices',
            Defaults('Invoices'),
            {
                CustomerNumber: "C0001023",
                InvoiceDate: Date(2022, 6, 13)
                PaymentTerms: "Cash On Delivery",
                TotalAmount: 13423.75
            }
        )
    ),
    // On failure
    Notify(
        "Error: the invoice could not be created",
        NotificationType.Error
    ),
    // On success
    Navigate('Success Screen')
)
```

Power Apps Forms Error-Handling

Write any code should be executed after a Power App Form is submitted in its OnSuccess and OnFailuer Properties. If form submission is successful, use the OnSuccess property to control what happens next. Otherwise, use the OnFailure property to display an error message that tells the user what went wrong.

```
// OnSelect property of the form's submit button
SubmitForm(frm_Invoice);

// OnSuccess property of the form
Navigate('Success Screen');

// OnFailure property of the form
Notify(
    "Error: the invoice could not be created",
    NotificationType.Error
);
```

Do not write any code after the [SubmitForm function](#) used to submit the form. If form submission fails Power Apps will still move onto the next line of code. This can result in loss of data.

```
// OnSelect property of the form's submit button
SubmitForm(frm_Invoice);
Navigate('Success Screen');
```

Power Automate Flow Error-Handling

When a Power Automate flow is [triggered from Power Apps](#) its response must be checked for errors. Flows can fail due to poor connectivity. They can also return a failure response or a result with the incorrect schema. If Power Apps does not know the flow failed it will continue as normal.

```
// Get customer invoices
If(
    IsError(
        GetAllCustomerInvoices.Run("C0001023")
    ),
    // On failure
    Notify(
        "Error: could not retrieve customer invoices",
        NotificationType.Error
    ),
    // On success
    Navigate('Success Screen')
)
```

IfError Function

Use the [IfError function](#) to handle calculations that require a different value when an error occurs. In this example, if gblTasksTotal equals 0 the IfError function will return 0 instead of throwing a “divide by zero” error.

```
// calculate the percentage of tasks completed
IfError(
    gblTasksCompleted/gblTasksTotal,
    0
)
```

Handling Unexpected Errors

An app's OnError property is triggered when an unexpected error occurs. An [unexpected error](#) is any error that is not handled using the IfError or IsError function.

Use this code to quickly locate the source of unexpected errors and fix them. Do not leave this on in a production app since the error message is helpful for a developer but is confusing to a user. If you want to see unexpected errors in a production app use the [Trace function](#) and [Azure Application insights](#) to silently log the errors.

```
// unexpected error notification message
Notify(
  Concatenate(
    "Error: ",
    FirstError.Message,
    "Kind: ",
    FirstError.Kind,
    ", Observed: ",
    FirstError.Observed,
    ", Source: ",
    FirstError.Source
  ),
  NotificationType.Information
);
```

Performance Optimization Guidelines

Load Multiple Datasets Concurrently

Making connector calls sequentially is slow because the current connector call must be completed before the next one starts. The [Concurrent function](#) allows Power Apps to load data faster by simultaneously processing multiple connector calls at once. Only use the Concurrent function to retrieve data stored in cloud. There is no advantage to using concurrent when working with data already on the device (i.e. variables and collections).

```

// Sequential code execution (slower)
Set(
    gblUserProfile,
    Office365Users.GetUserProfileV2(User().Email)
);
ClearCollect(
    colActiveProjects,
    Filter(
        Projects,
        ProjectStatus.Value="Active"
    )
)

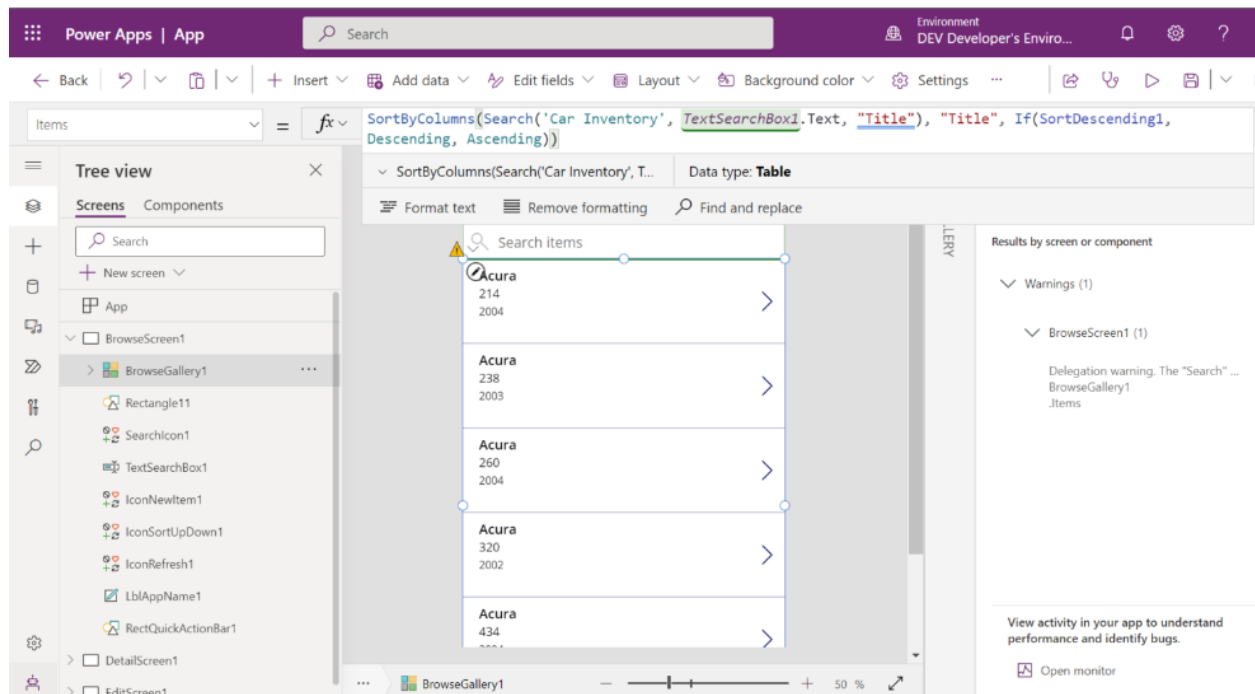
// Simultaneous code execution (faster)
Concurrent(
    // Thread #1
    Set(
        gblUserProfile,
        Office365Users.GetUserProfileV2(User().Email)
    ),
    // Thread #2
    ClearCollect(
        colActiveProjects,
        Filter(
            Projects,
            ProjectStatus.Value="Active"
        )
    )
)

```

Write Formulas That Use Delegation

Always write formulas that can be [delegated to the cloud datasource](#). Delegation is when data operations such as filter, lookup and search are performed in the cloud (i.e. SharePoint, Dataverse) instead of on the user's device. Data operations can be performed faster in the cloud because there are more computing resources than a laptop or mobile phone. Also, less data will be transmitted to the user's device because it has already been filtered by the datasource.

Refer to the official Power Apps documentation to determine which Power Fx functions can be delegated. The supported functions are different for [SharePoint](#), [Dataverse](#) & [SQL](#). A warning will appear in the [app checker](#) when a function cannot be delegated.



Dataverse views are not subject to Delegation rules. Use Dataverse views to write filter criteria that cannot be delegated using Power Apps formulas.

```
Filter(
    'Device Orders',
    'Device Orders (Views)'.Active Device Orders'
)
```

Cache Data In Collections And Variables

Store frequently used data in Collections and Variables. Data stored in memory can be accessed very quickly. A cloud datasource must receive a connector call, perform a query and send a response back to the device before data can be displayed on-screen.

```
// Store the currency exchange rates table in memory for quicker access
ClearCollect(
    colCurrencyExchangeRates,
    'Currency Exchange Rates',
)
```

Limit The Size Of Collections

Limit the size of collections to the least number of rows and columns that required by the app. Mobile devices have tight restrictions on memory usage. Collections are stored in the device's memory. If too much memory is in use the mobile operating system will kill the Power Apps process and the app will crash.

Use the [ShowColumns function](#) to select only specific columns and drop the rest from the collection. Enable [explicit column selection](#) to fetch only table columns used in the app when connecting to Dataverse.

```
// Selecting only desired columns from the accounts table
ClearCollect(
    colAccounts
    ShowColumns(
        Accounts,
        "name",
        "city",
        "state",
        "zipcode"
    )
)
```

“Batch Patch” Multiple Changes To A Datasource Table At Once

Quickly update multiple records in the same datasource table by using Batch Patch Technique. “Batch patch” enables record updates to be made simultaneously. The traditional ForAll + Patch method is slower because it makes the updates sequentially.

```
// Collection of records to update
ClearCollect(
    colUpdateEmployees,
    Table(
        {ID: 2, FullName: "Alice Henderson", Active: true},
        {ID: 4, FullName: "David Wright", Active: false},
        {ID: 5, FullName: "Mary Allen", Active: false}
    )
);
// Update records one-by-one (slower)
ForAll(
    colUpdateEmployees,
    Patch(
        Employees,
        Lookup(Employees, ID=colUpdateEmployees[@ID]),
        {
            FullName: colUpdateEmployees[@FullName],
            Active: colUpdateEmployees[@Active]
        }
    )
);
// Bulk update multiple records at once (faster)
Patch(
    Employees,
    ShowColumns(
        colUpdateEmployees,
        "ID",
        "FullName",
        "Active"
    )
);
```

Reduce Code In The App's OnStart Property

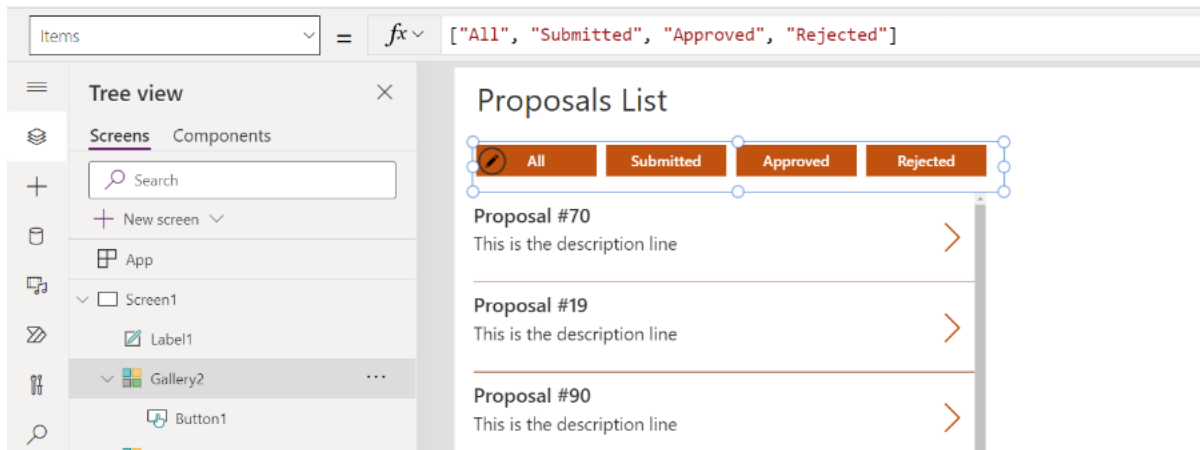
The more code that is in the app's [OnStart property](#), the longer an app will take to start. Improve app startup time by initializing global variables in the [OnVisible property](#) of the app's first screen. If possible, further defer setting variables until the screen they are needed.

Time to first screen metrics can be found in the app's Analytics page. Go to the maker portal, click on the three dots beside the app, select Analytics (preview), then choose Performance.

Minimize Number Of Controls On A Single Screen

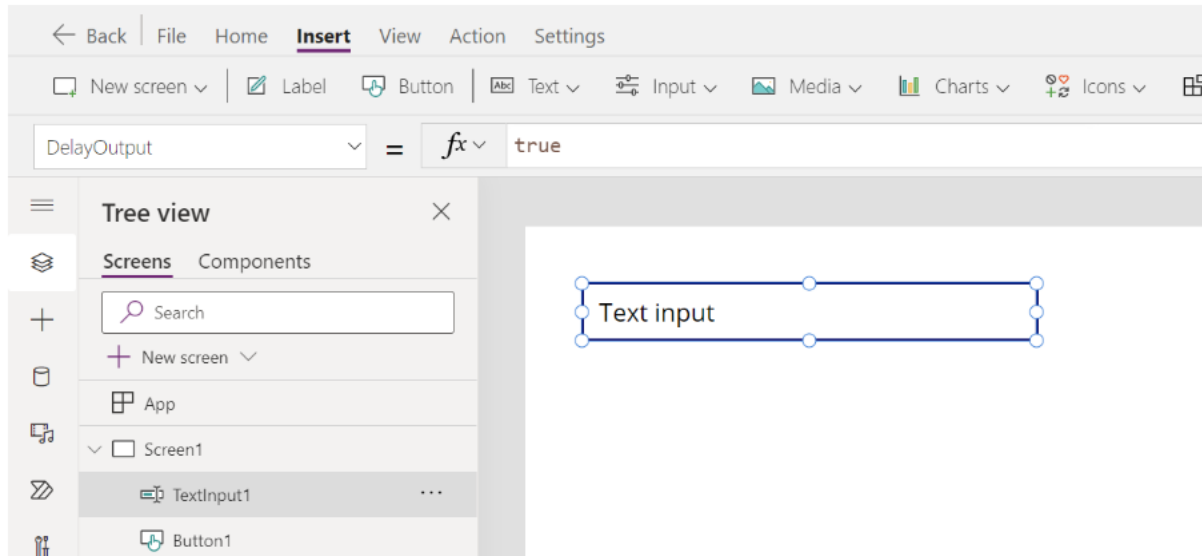
Every control added to a screen increases memory usage when the screen loads. Try to achieve a screen design with the fewest controls possible. A screen with fewer controls on it is faster to render and consumes less memory. For screens with too many controls, consider dividing their functionality across multiple screens.

Use a [gallery to display repetitive controls](#). Each control in a gallery only counts as 1 control no matter how many times it is shown.



Enable DelayOutput For Text Input Controls

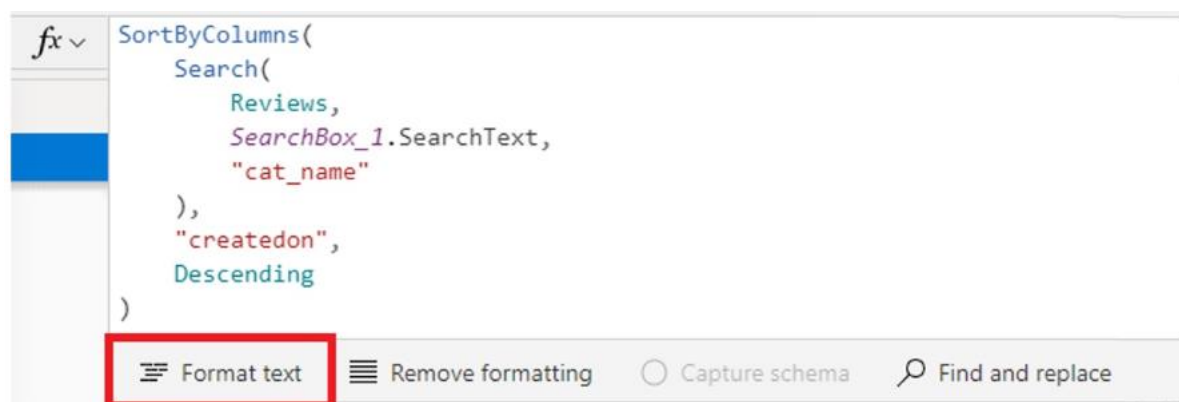
The Text property of a text input is updated after each keystroke. Set the [DelayOutput property](#) of the input to true to wait until after the user stops typing. This is useful when building a search bar connected to a gallery. With DelayOutput enabled the app will only make one request to the datasource when typing stops, as opposed to each keystroke.



Improving Code Readability

Apply Automatic Formatting

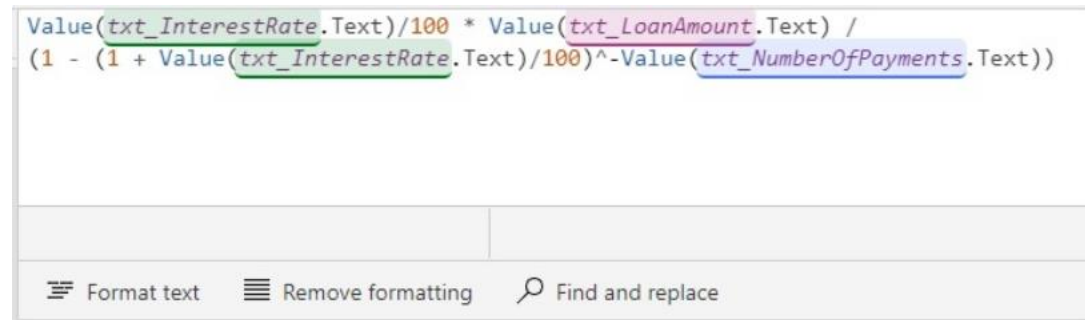
The formula bar's [format text command](#) applies indentation, spacing and line-breaks to Power Apps code. Well-formatted code has two benefits. It is easier to read and quicker to spot mistakes. Use the format text command to achieve a consistent coding style throughout a canvas app. A consistent coding style makes it easier for developers to work together on an app.



Use The WITH Function

Power Apps [With function](#) makes long formulas more readable. For example, this formula which does not use the With function calculates the monthly mortgage payment for a house:

```
Value(txt_InterestRate.Text)/100 * Value(txt_LoanAmount.Text) /  
(1 - (1 + Value(txt_InterestRate.Text)/100)^-Value(txt_NumberOfPayments.Text))
```



The mortgage calculation formula cannot be interpreted at-a-glance. It takes effort to parse. Compare it to the Power Apps code example using the With function. The formula is now human-readable because any complexity moved into one-time variables.

```
With(  
  {  
    InterestRate: Value(txt_InterestRate.Text)/100,  
    LoanAmount: Value(txt_LoanAmount.Text),  
    NumberOfPayments: Value(txt_NumberOfPayments.Text)  
  },  
  InterestRate * LoanAmount / (1 - (1 + InterestRate)^-NumberOfPayments)  
);
```



Flatten Nested IF Functions

Nested IFs are when multiple [IF functions](#) are placed inside one other. The more levels a nested IF contains the harder it becomes to understand. Use a flat structure whenever possible to improve code readability.

```

Set(gblBankAccountBalance, 5000);
Set(gblDailyWithdrawlLimit, 1000);
Set(gblWithdrawlAmount, 100);

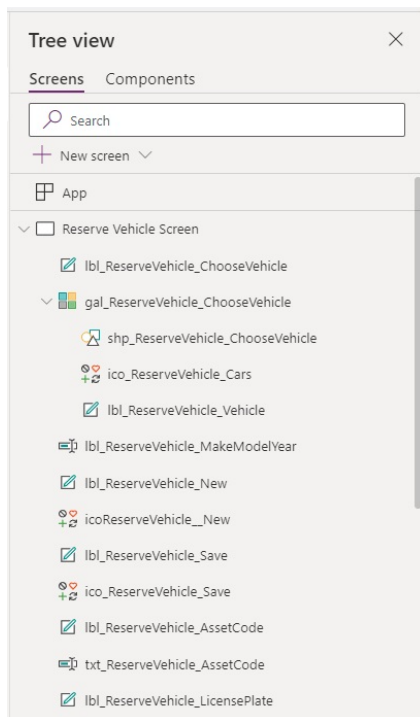
// Nested IFs
If(
    gblWithdrawlAmount > gblBankAccountBalance,
    Notify("Insufficient funds", Error),
    If(
        gblWithdrawlAmount > gblDailyWithdrawlLimit,
        Notify("Daily withdrawl limit exceeded", Error),
        Notify("You have Withdrawn $"&gblWithdrawlAmount, Success)
    )
);

// Flattened IFs
If(
    gblWithdrawlAmount > gblBankAccountBalance,
    Notify("Insufficient funds", Error),
    gblWithdrawlAmount > gblDailyWithdrawlLimit,
    Notify("Daily withdrawl limit exceeded", Error),
    Notify("You have withdrawn $"&gblWithdrawlAmount, Success)
);

```

Have A Consistent Naming Convention For Controls & Variables

Every control should follow a naming convention that includes the control type, the screen it is located on and the purpose of the control. Variables should also have a standard format that includes their scope and purpose.



Join Text Strings And Variables

Combining text can be done multiple ways in Power Apps: the & operator, the [Concatenate function](#) or String notation. Choose one way of doing it and be consistent.

```
// set variables
Set(gblUserName, User().FullName);
Set(gblStreetAddress, "123 Chestnut Street");

// join text using & operator
"Hi, my name is "&gblUserName&" and I live at "&gblStreetAddress&".";

// join text using the Concatenate function
Concatenate("Hi, my name is ",gblUserName," and I live at ",gblStreetAddress&".");

// join text using $-strings
$"Hi, my name is {gblUserName} and I live at {gblStreetAddress}."
```

Choose Consistent Logical Operators

The [logical operator And](#) can be written 3 different ways: And, And(), &&. There are often many ways to do the same thing in Power Apps code. It's OK to choose any option from the Power Apps code examples below but be consistent.

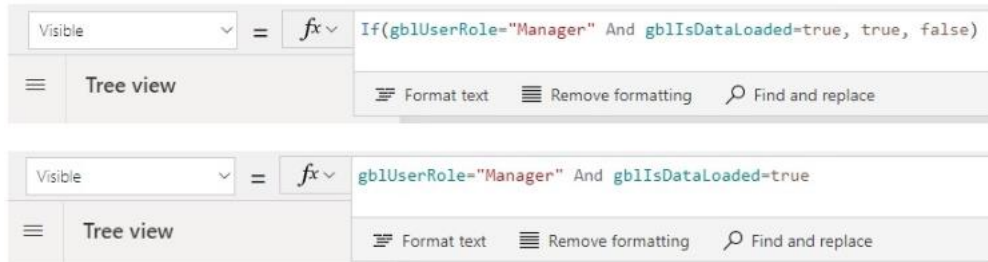
```
// And operator
Filter(
    colCustomers,
    State = "NY"
    And Status="Active"
);

// And() function
Filter(
    colCustomers,
    And(
        State = "NY",
        Status="Active"
    )
);

// && operator
Filter(
    colCustomers,
    State = "NY"
    && Status="Active"
);
```

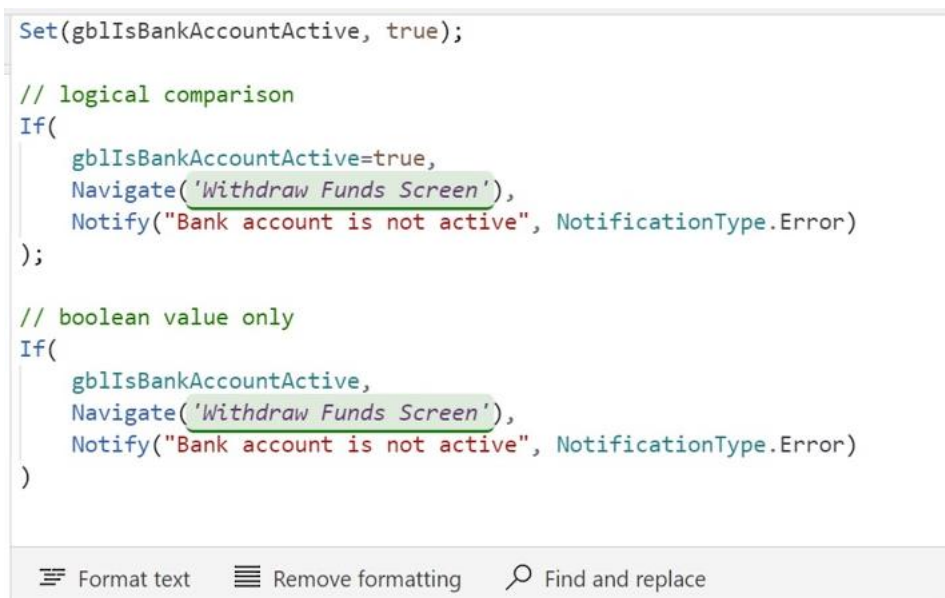
Remove IF Statements When The Result Is A True Or False Value

An IF statement that results in true or false is not necessary to write. Get rid of the IF statement and only write the logical comparison



Simplify Logical Comparisons When Evaluating A Boolean

A boolean value itself can be used as an argument to the IF function. It is not necessary to write a logical comparison.



Substitute The Self Operator For The Current Control Name

The [Self operator](#) is a concise way to access properties of the current control. Use Self instead of the full control name to make code quicker to understand.



Alphabetize Patch & UpdateContext Function Arguments

When the [Patch function](#) have a large number of fields it takes more time to find and update them. Use alphabetical order so the desired field can be quickly located. This technique can also be applied to the [UpdateContext function](#).

```
// create a new record
Patch(
    colContacts,
    Defaults(colContacts),
    {
        Active: true,
        Address: "67 Walnut Grove",
        Name: "Jane Smith",
        PostalCode: "R2G 3V3",
        Province: "Ontario"
    }
);

// update local variables
UpdateContext(
    {
        locAccountID: GUID(),
        locBlockerUserInput: true,
        locIsMenuVisible: false,
        locReadOnlyMode: false,
        locSelectedProperty: "Location"
    }
);
```

Format text Remove formatting Find and replace

Typography

The Basic

- Heading 1: 28px, Open Sans, Normal/normal
- Heading 2: 26px, Open Sans, Normal /normal
- Heading 3: 20px, Open Sans, Normal /normal
- Heading 4: 18px, Open Sans, Normal /normal
- *Heading 5: 16px, Open Sans, Normal /normal*
- Heading 6: 14px, Open Sans, Normal /normal
- Body: 16px, Open Sans, Normal

Title

- Section title: 16px, Open Sans, Normal
- Subsection title: 14px, Open Sans, Semi Bold
- Card title: 14px, bold, Open Sans, Bold
- Card subtitle: 12px, Open Sans, Semi Bold

Navigation

- Global: 12px, Open Sans, Semi Bold
- Secondary: 12px, Open Sans, Semi Bold
- Tertiary: 12px, Open Sans, Normal

Sub-content

- Hint: 10px, Open Sans, Normal
- Label: 10px, Open Sans, Semi Bold
- Unit: 10 px, Open Sans, Normal
- Description: 10 px, Open Sans, Normal
- Chip: 10 px, Open Sans, Normal
- Error text: 10px, Open Sans, Normal
- Placeholder: 14px, Open Sans, Normal

Button

- Primary: 12, Normal
- Radius: 0px, 2px

