# PowerApps canvas app coding standards and guidelines

This article is aimed at Microsoft PowerApps makers in the enterprise. It contains standards for naming objects, collections, and variables, and guidelines for developing consistent, performant, and easily maintainable apps.

## Introduction

Microsoft PowerApps is a high-productivity application development platform from Microsoft. Microsoft uses this platform to build first-party applications in Dynamics 365 for Sales, Dynamics 365 for Service, Dynamics 365 for Field Service, Dynamics 365 for Marketing, and Dynamics 365 for Talent. Enterprise customers can also use the same platform to build their own custom line-of-business applications. Individual users and teams within your organization can also build personal or team productivity applications without having to write very much code (or any).

## Purpose of this article

This white paper is targeted at the enterprise application maker (developer) who is responsible for designing, building, testing, deploying, and maintaining PowerApps apps in a corporate or government environment. This white paper was developed as a collaboration between the Microsoft PowerApps team, Microsoft IT, and industry professionals. Of course, enterprise customers are free to develop their own standards and practices. However, we feel that adherence to these guidelines will help developers in these areas:

* Simplicity
* Readability
* Supportability
* Ease of deployment and administration
* Performance
* Accessibility

This article is intended to be a living document. As Microsoft Power platform capabilities and industry standards change, so will this white paper.

## General naming conventions

This section describes “camel case” and “Pascal case” naming conventions. If you’re already familiar with those terms, you can skip ahead.

### Camel case

You should use camel case for controls and variables. Camel case begins with a lowercase prefix, removes all spaces from object or variable names, and capitalizes the first letter of each word after the first. For example, a text input control might be named txtUserEmailAddress.

[Camel case - Wikipedia](https://en.wikipedia.org/wiki/Camel_case)

### Pascal case

You should use Pascal case for data sources. Pascal case is sometimes referred to as “upper camel case.” Like camel case, it removes all spaces and capitalizes the first letter of words. However, unlike camel case, Pascal case also capitalizes the first word. For example, a common data source in PowerApps is the Microsoft Office 365 Users connector, which is named Office365Users in your code.

## Object naming conventions

As you create objects in your PowerApps apps, it’s important to use consistent naming conventions for screens, controls, and data sources. This approach will make your apps easier to maintain, can help improve accessibility, and will make your code easier to read as those objects are referenced.

This is OK! The object naming conventions in this white paper are intended as guidelines, and organizations are free to develop their own standards. The main point is to be consistent.

### Screen names

Screen names should reflect the purpose of the screen, so that it’s easier to navigate through complex apps in PowerApps Studio.

What’s less obvious is that screen names are read aloud by screen readers, which are needed for users who have vision accessibility needs. Therefore, it’s imperative that you use plain language to name your screens, and that the names include spaces and no abbreviations. Also, we recommend that you end the name with the word “Screen,” so that the context is understood when the name is announced.

Here are some good examples:

* Home Screen
* Thrive Help Screen

Here are some bad examples:

* Home
* LoaderScreen
* EmpProfDetails
* Thrive Help

### Control names

All control names on the canvas should use camel case. They should begin with a three-character type descriptor, followed by the purpose of the control. This approach helps identify the type of control and makes it easier to build formulas and search.

Here’s a good example: lblUserName

The following table shows the abbreviations for common controls

|  |  |
| --- | --- |
| Control name | Abbreviation |
| Address Input | add |
| Audio | aud |
| Barcode Scanner | bar |
| Button | btn |
| Camera control | cam |
| Canvas | can |
| Card | crd |
| Charts | chr |
| Check Box | chk |
| Collection | col |
| Container | con |
| Combo box | cmb |
| Date Picker | dte |
| Drop down | drp |
| Form | frm |
| Gallery | gal |
| Group | grp |
| Header page shape | hdr |
| Html text | htm |
| Icon | Ico |
| Image | img |
| Label | lbl |
| List Box | lst |
| Map | map |
| Microphone | mic |
| Microsoft Stream | str |
| Page section shape | sec |
| PDF Viewer | pdf |
| Pen Input | pen |
| Picture | pic |
| Radio button | rad |
| Rating | rtg |
| Rich Text Editor | rte |
| Shapes (rectangle, circle, and so on) | shp |
| Slider | sld |
| Table data | tbl |
| Text input | txt |
| Timer | tmr |
| Toggle | tgl |
| Video | vid |

Control names must be unique across an application. If a control is reused on multiple screens, the short screen name should be suffixed at the end – for example, galBottomNavMenuHS, where “HS” stands for “Home Screen.” This approach makes it easier to reference the control in formulas across screens.

### Data source names

When you add a data source to your application, the name can’t be changed in the PowerApps app. The name is inherited from the source connector or data entities that are derived from the connection.

Here are some examples:

* Name inherited from the source connector: The Office 365 Users connector is named Office365Users in your code.
* Data entities derived from the connection: A Microsoft SharePoint list that’s named Employees is returned from the SharePoint connector. Therefore, the name of the data source in your code is Employees. The same PowerApps app can also use the same SharePoint connector to access a SharePoint list that’s named Contractors. In this case, the name of the data source in the code is Contractors.

### Standard action connectors

In Standard action connectors that expose functions, such as LinkedIn, you’ll see that the data source name and its operations use Pascal casing (that is, UpperUpperUpper). For example, the LinkedIn data source is named LinkedIn and has an operation named ListCompanies.

### Custom connectors

Custom connectors can be created by any maker in your environment. They’re used to connect to custom application programming interfaces (APIs) such as third-party services or line-of-business APIs that your IT department has created. Pascal casing is also recommended for the data source name and its operations. Just be aware that the custom connector name and the way that it appears in PowerApps can differ.

For example, here’s a custom connector named MS Auction Item Bid API.

## Code naming conventions

As you add code to your PowerApps apps, it becomes increasingly important to use consistent naming conventions for variables and collections. If variables are named correctly, you should be able to quickly discern the type, purpose, and scope of each.

We found lots of variance in the code naming and object naming conventions of different organizations. For example, one team uses data type prefixes for its variables (such as strUserName to indicate a string), whereas another team prefixes all its variables with an underscore (\_) to group them in IntelliSense. There are also differences in the way that teams denote global variables versus context variables.

The same guidance applies here: Have a pattern for your team, and be consistent in its usage.

### Variable names

* Be descriptive of the variable’s function. Think about what the variable is bound to and how it’s used, and then name it accordingly.
* Prefix your global and context variables differently.

Be smart! PowerApps lets context variables and global variables share the same names. This can cause confusion, because, by default, your formulas use context variables unless the disambiguation operator is used. Avoid this situation by following these conventions:

* Prefix context variables with **loc**.
* Prefix global variables with **gbl**.
* The name after the prefix should indicate the intent/purpose of the variable. Multiple words can be used and don’t have to be separated by any special characters (for example, spaces or underscores), provided that the first letter of each word is capitalized.
* Use Camel casing. Begin your variable names with a prefix in lowercase letters, and then capitalize the first letter of each word in the name (that is, lowerUppperUpper).

Here are some good examples:

* Global variable: gblFocusedBorderColor
* Context variable: locSuccessMessage

### Collection names

* Be descriptive of the collection’s contents. Think about what the collection contains and/or how it’s used, and then name it accordingly.
* Collections should be prefixed with col.
* The name after the prefix should indicate the intent or purpose of the collection. Multiple words can be used and don’t have to be separated by spaces or underscores, provided that the first letter of each word is capitalized.
* Use Camel casing. Begin your collection names with a lowercase col prefix, and then capitalize the first letter of each word in the name (that is, colUpperUpper).

Here are some good examples:

* colMenuItems
* colThriveApps

Here are some bad examples:

* orderscoll
* tempCollection

## Organizing your objects and code

### Using groups for organization

All controls on a screen should belong to a group, so that you can easily recognize their purpose, move them around a screen or between screens, or collapse them to simplify your view. Gallery, Form, and Canvas controls are already groups, but they can also, optionally, be part of another group to help improve organization.

### Format text feature

As the complexity of a formula increases, readability and maintainability are affected. It can be very difficult to read a large block of code that contains multiline functions. The Format text feature adds line breaks and indentation to make your formula easier to read. As for code comments, the extra white space is removed from the app package that’s downloaded to the client. Therefore, there’s no need to use the **Remove formatting** feature before you publish your app.