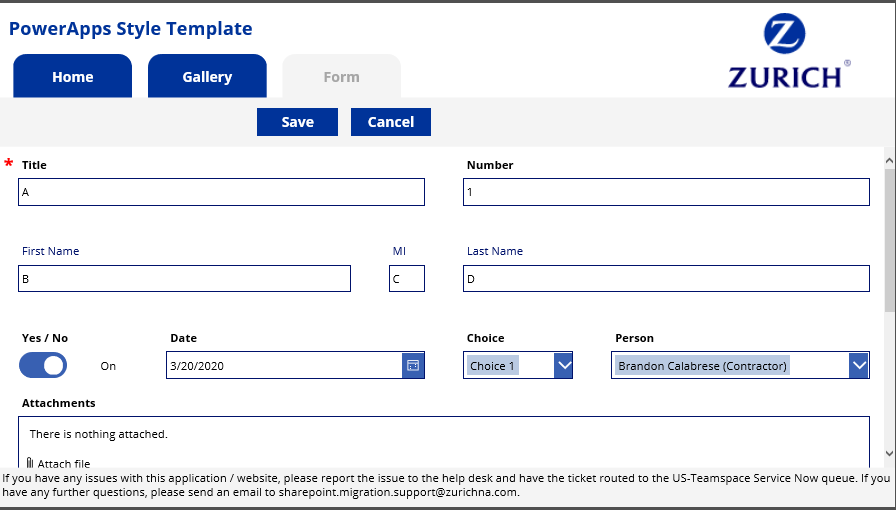
**ZNA Power APP Template Guidelines**

# Zurich PowerApps Style Guidelines

The SPaaS team will begin using a new Power App template which provides all pre-defined styles and colors, variable naming, etc. to be used when developing Power Apps for ZNA. The template will ensure the team provides a consistent look and feel for the applications which we develop and support. Tabs will be used at the top of the forms to categorize data into different sections to allow ease of use to navigate within the application.

Below is the template look and feel.

<https://apps.powerapps.com/play/4237b668-08e6-4302-b757-adb79b69c279?tenantId=473672ba-cd07-4371-a2ae-788b4c61840e>



## Variable Naming

Variables should be named using camel case, with a prefix to distinguish between global and context variables. ‘loc’ should be prefixed to context variables, while ‘glo’ should be prefixed to global variables.

Some examples of variable names that meet these standards are:

locSelectedItem

gloTab

gloStyleSheet

Some variable names that do not meet these standards are:

selectedItem -> Missing the prefix

glotab -> Subsequent words are not capitalized in a camel case fashion

GloStyleSheet -> Camel case for variables should begin with a lower case letter

## Control Naming

Controls should be names using camel case, with a prefix to distinguish the type of control. For example, a label for the title of the app, lblTitle, derives its name by combining the prefix for a label control, lbl, with a name describing what the control represents, Title.

A table showing some common prefixes for controls. It is usually preferred that prefixes are kept at three characters in length for consistency across the app.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| btn | Button | chk | CheckBox | cmb | ComboBox |
| dtp | DateTimePicker | lbl | Label | lst | ListBox |
| icn | Icon | rdo | RadioButton | txt | TextBox |
| rtx | RichTextBox | icn | Icon | scr | Screen |
| frm | Form | gal | Gallery | crd | DataCard |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Style Sheet Values

The style sheet used variable used in the app start for the template app is as follows:

Set (

gloStyleSheet,

{

textLight: ColorValue("#FFFFFF"),

textDark: ColorValue("#000000"),

paleWhite: ColorValue("#F4F4F4"),

textGrey: ColorValue("#B0B0B0"),

titleFont: Font.'Open Sans',

titleSize: 21,

titleColor: ColorValue("#003399"),

titleEmph: FontWeight.Bold,

fieldFont: Font.'Open Sans',

fieldSize: 13,

fieldColor: ColorValue("#000000"),

fieldEmph: FontWeight.Bold,

fieldPaddingTop: 5,

fieldPaddingRight: 5,

fieldPaddingBottom: 5,

fieldPaddingLeft: 0,

fieldX: 32,

fieldHeight: 43.5,

inputFont: Font.'Open Sans',

inputSize: 13,

inputHeight: 40,

inputColor: ColorValue("#000000"),

inputEmph: FontWeight.Normal,

inputPaddingTop: 5,

inputPaddingRight: 5,

inputPaddingBottom: 5,

inputPaddingLeft: 5,

inputX: 32,

errorPaddingTop: 0,

errorPaddingRight: 0,

errorPaddingBottom: 0,

errorPaddingLeft: 0,

errorHeight: 15,

requiredStarX: 2,

requiredStarWidth: 30,

backgroundColor: ColorValue("#FFFFFF"),

tabFont: Font.'Open Sans',

tabSize: 16,

tabColor: ColorValue("#FFFFFF"),

tabBackgroundColor: ColorValue("#003399"),

tabDisabledBackgroundColor: ColorValue("#F4F4F4"),

tabEmph: FontWeight.Bold,

requiredAstColor: ColorValue("#FF0000"),

requiredAstSize: 15

}

);

Set (

gloSpacing,

{

fieldLabelAndValue: 5,

betweenFields: 10

}

);

The title values refer to properties in the app title label

The field values refer to properties in form field labels

The input values refer to properties in form inputs

The tab values refer to properties in the tab buttons

Required ast values refer to the required mark in form fields

## When to use Tabs

Tabs should be used when separating portions of a form into distinct groupings. For example, if a form has several fields about a claimant’s information, and also several fields about the claim being made, it might make sense to separate these into two tabbed sections. One for the personal info, and one for the claim information, in order to make the form easier to use for the end user.

The tabs then operate by setting a global or context variable to the currently selected tab, and all form fields refer to that variable to decide if they should be visible or not.

## When to use Screens

Screens are more appropriate to use over tabs when there are several stages to an application. For example, on the first screen a user could select an item from a gallery or click a button to create a new item. Both of these actions would transition to a second screen where the form resides. Due to the way screen transitioning works, data can be passed as input to the new screen. This is a good method for passing selected item information and whether the form is for display / new / or editing.

## Development Process

When creating a new app, the app can be developed directly in production up until the point where the client end-users begin to use the app in their business processes, and add real data to any of the apps data sources.

After this point, any future additions / fixes / changes made to the app should be done through the following process:

1. If development data sources or a dev copy of the app do not yet exist follow these steps:
2. If the app is utilizing SharePoint data connections, create a new test site and recreate any data sources as a copy of the lists on the production site here.
3. “Save as” the production app to make a copy of it, or export and import it as a new app.
4. Before making any changes, update the data connections in the development app to point towards the development data sources. Be sure no production data sources are going to be altered.
5. If the development data / app already exist or have just been created
6. Make any new changes in the development data / app environment. All testing and UAT should be done here before moving changes to prod. The client should be informed that the app is pointing to test data so that they are not confused when testing the app.
7. Once everything is tested and has UAT sign off, save the app as the production app. **This will not affect the currently published version of that app. It will add a new development version.** From here, reconnect the production data sources and verify that everything is working correctly. Once ready to roll the changes out, publish the app and write a meaningful version update note.

## Responsive Design Considerations

By disabling the locked aspect ratio, a powerapp can automatically resize its dimensions to fit the size of the screen / device. However, there are no automatic mechanisms to resize any of the app components to fit the dynamic size, so every control will need to be designed with this in mind.

The primary considerations need to be put into variable screen width and height.

App.Width and App.Height will provide the current value of the width and height of app respectively. Every control in the app will need to be made to consider these. Below is a list of examples.

1. A bar that needs to span the whole width of the screen would have its Width set to App.Width, and height set to some static value. X and Y would be zero if it rested upon the top left of the screen.

2. A form that would sit under this bar and stretch to the bottom of the screen would have its width set to App.Width, and its height set to App.Height - bar.Height. Then, it’s Y value would be set to the bar.Y + bar.Height.

All controls can be organized in this way using a bit of math, and will either be organized via other controlls or the dimensions of the screen.

Finally if statements will need to be used to reorganize things based on screen width. If the screen width is too small, we would not want to have multiple controls on one line for example. So, an if statement in each form card for the width might look like:

If(App.Width < 1000, App.Width, App.Width / 3).

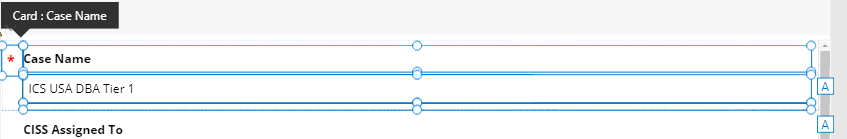
This If makes the control take up full width if the app is less than 1000 pixels wide, or only a third of the width if it is larger.

## Datacards, Labels, Input Fields, and Error Messages

Each object has various properties for controlling the look and feel of the form. Please make use of the guidance below regarding data cards, labels, input fields, and error messages as they relate to the ZNA stylesheet.

There are four controls in a typical card -

* The "required star" label - **lbl*CardName*RequiredStar**
* The "error message" label - **lbl*CardName*ErrorMessage**
* The label for the field - **lbl*CardName***
* The input control for the field - ***abcCardName*** (where abc is the three letter prefix, such as txt, cmb, dtp, rtx, etc.)



The positioning/sizing/formatting of the controls I have set up as follows -

* The card itself (crd*CardName*)
  + WidthFit = true (so it fills the width of the parent form!)
  + Height = lbl*CardName*ErrorMessage.Y + lbl*CardName*ErrorMessage.Height
* lbl*CardName*RequiredStar
  + Color = gloStyleSheet.requiredAstColor
  + Font = gloStyleSheet.fieldFont
  + Size = gloStyleSheet.requiredAstSize
  + FontWeight = gloStyleSheet.fieldEmph
  + PaddingTop = gloStyleSheet.fieldPaddingTop
  + PaddingRight = gloStyleSheet.fieldPaddingRight
  + PaddingBottom = gloStyleSheet.fieldPaddingBottom
  + PaddingLeft = gloStyleSheet.fieldPaddingLeft
  + X = gloStyleSheet.requiredStarX
  + Y = gloSpacing.betweenFields (10)
  + Width = gloStyleSheet.requiredStarWidth
  + Height = gloStyleSheet.fieldHeight
* lbl*CardName*
  + AutoHeight = true
  + Color = gloStyleSheet.fieldColor
  + Font = gloStyleSheet.fieldFont
  + Size = gloStyleSheet.fieldSize
  + FontWeight = gloStyleSheet.fieldEmph
  + PaddingTop = gloStyleSheet.fieldPaddingTop
  + PaddingRight = gloStyleSheet.fieldPaddingRight
  + PaddingBottom = gloStyleSheet.fieldPaddingBottom
  + PaddingLeft = gloStyleSheet.fieldPaddingLeft
  + X = gloStyleSheet.fieldX
  + Y = gloSpacing.betweenFields (10)
  + Width = Parent.Width - 60
  + Height = gloStyleSheet.fieldHeight
* *abcCardName*
  + Color = gloStyleSheet.inputColor
  + Font = gloStyleSheet.inputFont
  + Size = gloStyleSheet.inputSize
  + FontWeight = gloStyleSheet.inputEmph
  + PaddingTop = gloStyleSheet.inputPaddingTop
  + PaddingRight = gloStyleSheet.inputPaddingRight
  + PaddingBottom = gloStyleSheet.inputPaddingBottom
  + PaddingLeft = gloStyleSheet.inputPaddingLeft
  + X = gloStyleSheet.inputX
  + Y = lbl*CardName*.Y + lbl*CardName*.Height + gloSpacing.fieldLabelAndValue (5)
  + Width = Parent.Width - 60
  + Height = gloStyleSheet.inputHeight
* lbl*CardName*ErrorMessage
  + AutoHeight = true
  + Color = gloStyleSheet.requiredAstColor
  + Font = gloStyleSheet.fieldFont
  + Size = gloStyleSheet.fieldSize
  + FontWeight = gloStyleSheet.fieldEmph
  + PaddingTop = gloStyleSheet.errorPaddingTop
  + PaddingRight = gloStyleSheet.errorPaddingRight
  + PaddingBottom = gloStyleSheet.errorPaddingBottom
  + PaddingLeft = gloStyleSheet.errorPaddingLeft
  + X = gloStyleSheet.fieldX
  + Y = *abcCardName*.Y + *abcCardName*.Height
  + Width = Parent.Width - 60
  + Height = gloStyleSheet.errorHeight

## Making Use of Tabs

It is often necessary to break up a large form into smaller pieces, and the tabs are one approach to doing this. Please make use of the guidance in this section when developing Power Apps with forms.

Note – if using a list-based form Power App, you do **not** need to include Save and Cancel buttons! They are already present on the chrome provided by SharePoint around the Power App.



* The tabs should be colored based on the stylesheet. The **Color** property should be set to **gloStyleSheet.tabColor**, the **PressedColor** property should be set to **gloStyleSheet.tabColor**, the **Fill** property should be set to **gloStyleSheet.tabBackgroundColor**, the **DisabledFill** property should be set to **gloStyleSheet.tabDisabledBackgroundColor**, the **PressedFill** property should be set to **ColorFade(gloStyleSheet.tabBackgroundColor, 30%)**, and the **HoverFill** property should be set to **ColorFade(gloStyleSheet.tabBackgroundColor, 30%)**.
* Each tab is a button. Underneath the multiple buttons is a rectangle which spans the entire width of the app. The rectangle should have the **Fill** property set to **gloStyleSheet.tabDisabledBackgroundColor** and the **DisabledFill**, **PressedFill**, and **HoverFill** properties all set to **rctTabBase.Fill**. (Assuming you name the rectangle **rctTabBase**.)
* The tabs should have their font face/size/weight set per gloStyleSheet. --- **Font** = gloStyleSheet.tabFont, **Size** = gloStyleSheet.tabSize, & **FontWeight** = gloStyleSheet.tabEmph
* To help with positioning, I set up the following global variable which I'd suggest doing on the App object's **OnStart** action:  
    
  Set (  
      gloTabData,  
      {  
          aspectRatio: 180 / 70,  
          count: 9,   **<-- TODO: Change this to match the number of tabs in your form!**  
          spacing: 5, **<-- Can change this value also if you want more space between your tabs.**  
          standardCornerRadius: 20,  
          standardHeight: 70,  
          standardTabBaseHeight: 75,  
          standardWidth: 180  
      }  
  );
* To set the rounded corners for each tab,
  + RadiusTopLeft ... **gloTabData.standardCornerRadius \* Self.Width / gloTabData.standardWidth**
  + RadiusTopRight ... **gloTabData.standardCornerRadius \* Self.Width / gloTabData.standardWidth**
  + RadiusBottomLeft ... **0**
  + RadiusBottomRight ... **0**
* To position each tab,
  + X ... For the first tab, set it to **gloTabData.spacing**. For the second and all subsequent tabs, set it to **btnPreviousTab.X + btnPreviousTab.Width + gloTabData.spacing**
  + Y ... For all tabs, set it to **gloTabData.spacing**
  + Width ... For all tabs, set it to **Min ( gloTabData.standardWidth, (App.Width - (gloTabData.spacing \* (gloTabData.count + 1))) / gloTabData.count )**
    - The formula ensures that tabs are not longer than the standard width, but they can be shorter if they need to be so that they fit in the space alloted.
    - Adjust the font size downward as needed for shorter tabs with longer text ... make it a percentage of tabSize ... for example, Size = gloStyleSheet.tabSize \* 0.75 for 75% of the desired font size.
  + Height ... For all tabs, set it to **Self.Width / gloTabData.aspectRatio**
    - The formula also shrinks the height relative to the width (if the width were to shrink lower than the standard 180) to ensure the standard aspect ratio is kept!
* To position and size the "tab base" rectangle,
  + X ... **0**
  + Y ... **btnFirstTab.Y + btnFirstTab.Height**
  + Width ... **App.Width**
  + Height ... **btnFirstTab.Height / gloTabData.standardHeight \* gloTabData.standardTabBaseHeight**
* To position and size the form on the page,
  + X ... **0**
  + Y ... **rctTabBase.Y + rctTabBase.Height + gloTabData.spacing**
  + Width ... **App.Width**
  + Height... **App.Height - frmSharePoint.Y**
* To switch between tabs, use a local variable updated via the **UpdateContext** method.
  + In the screen's **OnVisible** action, use the formula **UpdateContext ( { locSelectedTab: btnFirstTab.Text } );** (best practice is *not* to hard code any string literals/constants!)
  + For all tab buttons' **OnSelect** actions, use the formula **UpdateContext ( { locSelectedTab: Self.Text } );**
  + For all tab buttons' **DisplayMode** properties, use the formula **If ( locSelectedTab = Self.Text, DisplayMode.Disabled, DisplayMode.Edit )** so that the tab is disabled when selected, and not disabled when not selected. It seems a bit counterintuitive, but it makes it so that the currently selected tab cannot be clicked while all of the rest of them can be!
* Each of the form's cards can then have the following formula in their **Visible** property - for example, if a card was to be shown on the first tab, **locSelectedTab = btnFirstTab.Text**
* To handle error messages, the ideal scenario is that after clicking save and one or more error messages being present, the user would automatically be switched to the first tab that has an error message present. We will handle this scenario entirely in the **OnFailure** action formula of the form itself. This formula will have two steps. The first step is to run **UpdateContext** to set local variables, one per tab, which check all fields' error messages within a given tab and distill that down to a true/false value on if an error message exists on that tab. The second step is to run a second **UpdateContext** to set the selected tab to the first tab with an error message present anywhere on it.  
    
  For this example, let's say we have an app with three tabs and three fields on each tab. The error message label follows a naming standard such as **lblTab1Field1ErrorMessage**.  
    
  UpdateContext (  
      {  
          locFirstTabHasFieldWithError: (Not IsBlank(lblTab1Field1ErrorMessage.Text)) Or (Not IsBlank(lblTab1Field2ErrorMessage.Text)) Or (Not IsBlank(lblTab1Field3ErrorMessage.Text)),  
          locSecondTabHasFieldWithError: (Not IsBlank(lblTab2Field1ErrorMessage.Text)) Or (Not IsBlank(lblTab2Field2ErrorMessage.Text)) Or (Not IsBlank(lblTab2Field3ErrorMessage.Text)),  
          locThirdTabHasFieldWithError: (Not IsBlank(lblTab3Field1ErrorMessage.Text)) Or (Not IsBlank(lblTab3Field2ErrorMessage.Text)) Or (Not IsBlank(lblTab3Field3ErrorMessage.Text))  
      }  
  );  
    
  UpdateContext (  
      {  
          locSelectedTab: If (  
              locFirstTabHasFieldWithError,  
              btnFirstTab.Text,  
              locSecondTabHasFieldWithError,  
              btnSecondTab.Text,  
              locThirdTabHasFieldWithError,  
              btnThirdTab.Text,  
              locSelectedTab    **<-- This is so that if the failure wasn't due to an error message, we'll just stay on the currently selected tab!**  
          )  
      }  
  );

## Currency Formatting

* **Format** property - **TextFormat.Number** (it should already be set to this by default, but just making sure!)
* **OnChange** formula -  
    
  UpdateContext (  
      {  
          loc*CardName*Value: If (  
              IsBlank ( Self.Text ),  
              "",  
              Text ( Value ( Self.Text ), "[$-en-US]$#,##0.00" )  
          )  
      }  
  );  
    
  Reset ( txt*CardName* );
* **Default** formula -  
    
  If (  
      loc*CardName*Value = Blank ( ),  
      If (  
   IsBlank ( Parent.Default ),  
   "",  
   Text ( Value ( Parent.Default ), "[$-en-US]$#,##0.00" )  
   ),  
      loc*CardName*Value  
  )

Screen **OnVisible** action formula & Form **OnReset** action formula -

UpdateContext (

    {

        loc*CardName*Value: Blank ( )

    }

);

## Other Standards

Multiple patterns and best practices are fully documented on this GitHub repository: [SharePointMigrations/modernize-sharepoint-applications: Generally applicable solutions and patterns to modernize SharePoint on-premises customizations for hosting in Microsoft 365 and Azure (dxc.com)](https://github.dxc.com/SharePointMigrations/modernize-sharepoint-applications)

These patterns and best practices should be followed whenever practical. Additionally, the repository is regularly updated with new or updated patterns and best practices, so the list below is only a sampling of what is available.

* Choice Fields: Enabling users to add values manually
* People Picker Best Practices
* Using the current list item when an in-list form opens
* Using SharePoint Security: Obtaining security information from SharePoint sites to enable changing the logic of a Power App based on the current user’s role or security group membership