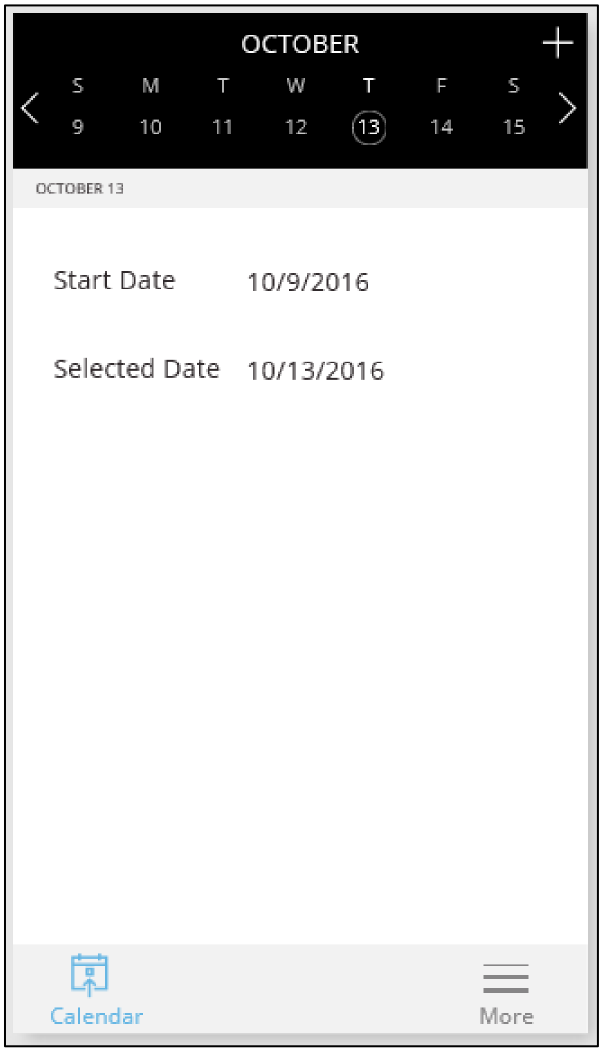
Creating a Calendar App using PowerApps

What we will be building

For this simple example, we are going to be building an application that allows you to display a 7-day-calendar-like navigation for users to select a day. As you select the dates, it highlights the date by using a different color for the font and draws a circle around the selected date.



Start Date 10/9/2016

Selected Date 10/13/2016

Figure 1. Calendar application

This is not meant to be a complete application but this navigation can easily be extended to build a richer application that displays appointments, tasks or other items, by adding a Vertical Gallery and setting its Items to include a Filter by the selected date and leverage connections to services like Common Data Model, Office 365 or Google Calendar very easily.

Storing the date

To build this application we will use two context variables (using UpdateContext) in the main screen:

1) **SelectedDate**: which will be the one indicating the date that the user selected.
2) **StartDate**: which we will use to determine the first day of the week to be displayed on the Date selector.

Initializing variables

The best place to initialize both of these variables is on the Screen::OnVisible event of the first screen which will be triggered when the user launches the application and navigates to that screen. One thing to keep in mind is that navigating back from other screens will trigger this event as well, so it is a best practice to make sure you only initialize the state when it has not been set to avoid resetting the data that the user might have set. We'll look on how to do this later.

Start a new Blank - Phone Layout - Application.

Launch PowerApps.

Select the option New→Blank app→Phone layout.

Select the first screen and switch to the Content tab in the ribbon. Click the Screen1 label name and rename it to MainScreen.

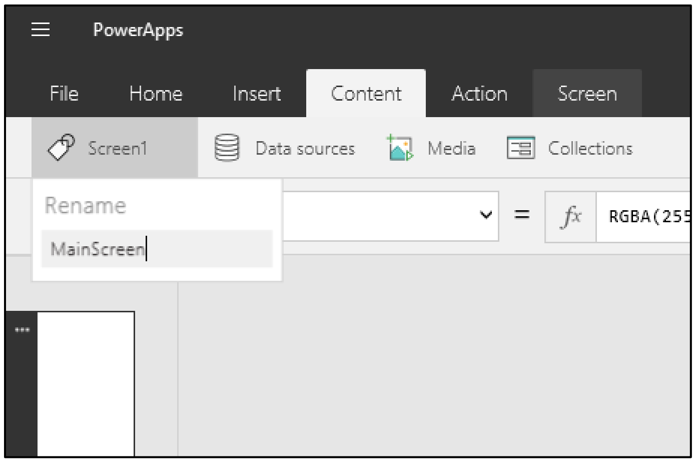
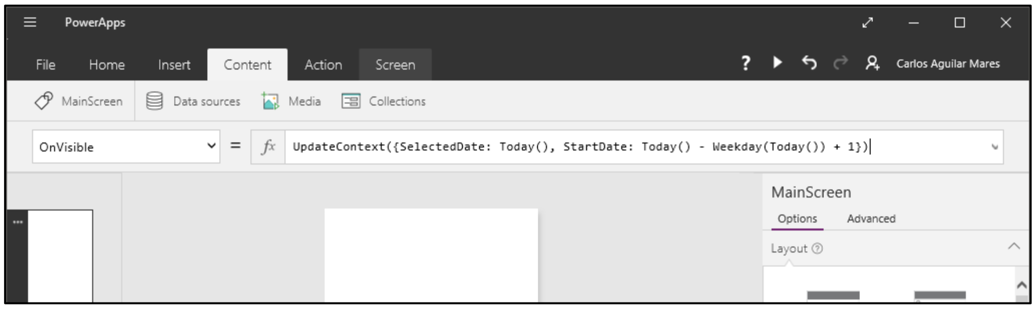


Figure 2. Renaming a screen or control

Now, we are going to add the code to initialize the variables, for this select the OnVisible on the properties dropdown and set the following rule:

```
UpdateContext({SelectedDate: Today(), StartDate: Today() -
Weekday(Today()) + 1})
```



OnVisible = UpdateContext({SelectedDate: Today(), StartDate: Today() - Weekday(Today()) + 1})

Figure 3. Screen.OnVisible event

This rule is initializing the SelectedDate context variable to Today(), and updating the StartDate context variable to be the Sunday for the current week by subtracting the day of the week.

Tip: You can add two Text boxes and set their Text property to SelectedDate and StartDate respectively to allow you to easily inspect their content. This is a common practice while you are building applications to quickly inspect easily values as they mutate.

Important Note: You might need to add a new screen, select it in the screen selector and come back to the main screen to ensure that the OnVisible event of the screen is triggered and the variable is initialized properly. This is a current limitation of PowerApps that will be addressed in the future.

Building the User Interface

Now that we have the two variables initialized and available to us, it is time to build the user interface that will allow us to read the values as well as change the date as needed.

1) Adding the Background for the date selector

We will start adding the rectangle that gives the black background for the date selector. So, add a Rectangle by clicking the Insert→Icon→Rectangle which will be the background for the top section of our calendar app where we will display the date selector. Set the following properties:

```
Name: BackgroundRectangle
Fill: Black
X: 0, Y:0, Width: 640, Height: 174
```

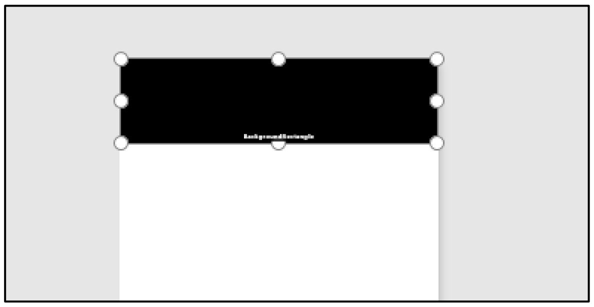


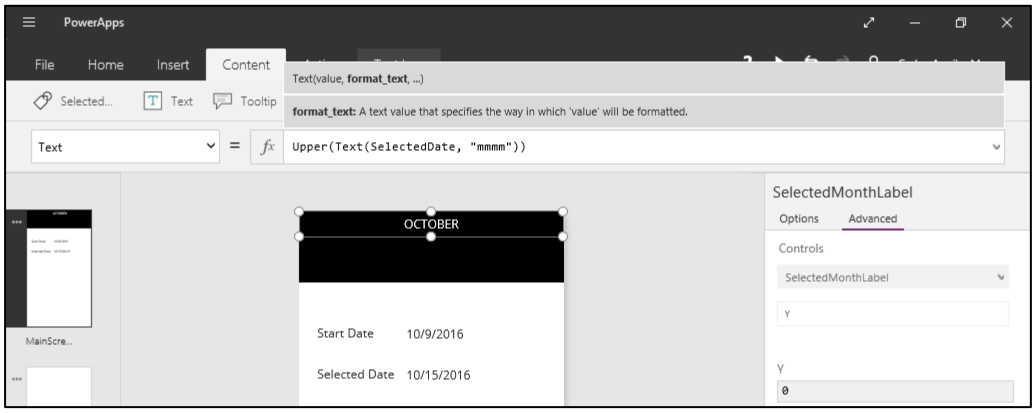
Figure 4. Background for the date selector

2) Adding a Label to Display the Month

This will be the label that displays the name of the month of the currently visible dates so users can get the right context.

```
Name: SelectedMonthLabel
Align: Center
Color: White
Size: 21
X: 0, Y:0, Width: 640
Text: Upper(Text(StartDate, "mmmm"))
```

The Text rule above uses the formatting formula “Text” to display the full name of the month (i.e. October), and then uses the “Upper” formula to convert it to upper case. (i.e. OCTOBER)



Text(value, format_text, ...)

format_text: A text value that specifies the way in which 'value' will be formatted.

Text = Upper(Text(SelectedDate, "mmmm"))

Start Date 10/9/2016

Selected Date 10/15/2016

SelectedMonthLabel

Y 0

Figure 5 - Formatting the month

Note: If you don’t see the Label displaying the Month, make sure to add a new Screen, navigate to it, and navigate back. This will ensure that the OnVisible event is triggered. This will not be a problem for your users, and is only a current limitation of the authoring environment.

3) Adding the date selector.

To allow users to select different days, we want to display a full week (7 days) starting Sunday so they can easily click one of them. For this we will use a Horizontal Gallery, and we will bind it to an array of 7 items (0-6) that we will use to add one day for each item to the StartDate so that we can easily calculate the date that each item will represent.

a. Add a new Horizontal Gallery.

Click Insert→Gallery→Horizontal (text gallery)

By default it will include three text box controls, remove all of them from the default template that is included. You can use the Advanced view to facilitate the selection of those controls.

b. Display the day of the number

For this we will use a textbox to display the day (number). So, add a **textbox** to the Gallery by clicking Insert→Text. Make sure that the gallery is selected so that the textbox is added inside the template. Set the following properties:

```
Name: DateNumberLabel
X: 0, Y: 44, Width: 76, Height: 40
Align: Center
Size: 16
Color: If(ThisItem.IsSelected,  RGBA(255,255,255,1), RGBA(200,
200, 200, 1))
Text: Day(StartDate + Value)
```

The Color property above uses the “If” statement to check if the current date is the selected one in the gallery, and if so it will use a different color to differentiate the users selection. In a gallery you can use the **ThisItem** to get a reference to the current item and you can use the **IsSelected** property to determine if it is the currently selected one.

The Text property is being set by using the “Day” formula that returns the day of the month for the specified date.

c. Add the highlight for the currently selected date

For this, we will use a Circle to surround the day of the month that is currently selected. Add a Circle by using the Insert→Icon→Circle, and set the following properties:

```
Name: DateCircle
X: 20, Y: 46, Width: 38, Height: 38
BorderColor: White
BorderStyle: Solid
Fill: RGBA(0,0,0,0)
Visible: If(SelectedDate = DateAdd(StartDate, ThisItem.Value),
true, false)
```

The Visible property is being set to the true if the selected date is the same as the date that they current item represents.

d. Display the First letter of the month

We want to also help the user by displaying the first letter of the month so they can easily know if it is Wednesday or Thursday without the need to count days, so to do that add another textbox where we will display the day of the Week. Note that this will also be used as the "hit point" for our selection, so it will take over the entire template of the gallery to capture all clicks.

```
Name: DateFirstLetter
X: 0, Y: 0, Width: 76, Height:106
Size: 16
Align: Center
VerticalAlign: Top
Text: Left(Text(StartDate + Value, "ddd"), 1)
Color: If(ThisItem.IsSelected,  RGBA(255,255,255,1), RGBA(200,
200, 200, 1))
```

```
OnSelect: UpdateContext({SelectedDate: DateAdd(StartDate,
ThisItem.Value, Days)})
```

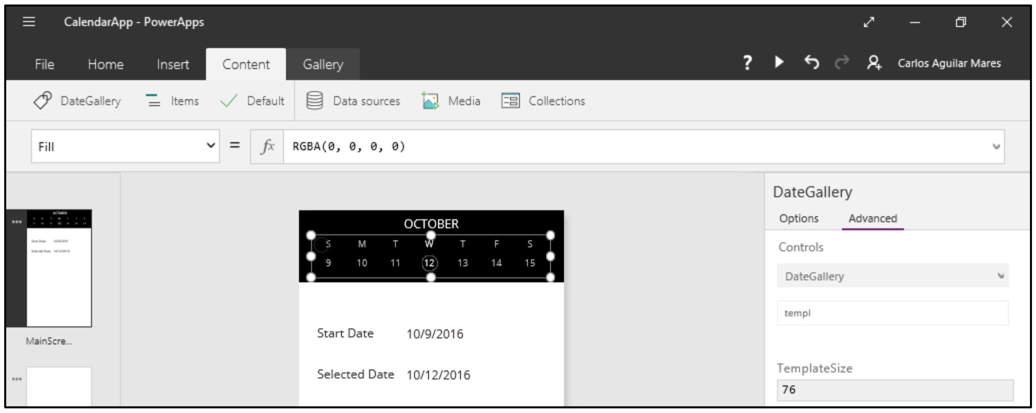
Here the Text property is using the "Left" formula to get only the first character of the three letter day (Mon, Tue, Wed, etc).
We are also establishing the "OnSelect" event to Update the SelectedDate with the value that this item represents.

e. Finally, lets format the gallery/date selector itself

For that select the gallery in the Canvas and set the following properties to it.

```
Name: DateGallery
X: 29, Y: 58, Width: 580, Height: 106
TemplateSize: 76
Items: [0, 1, 2, 3, 4, 5, 6]
```

By now you should be able to select any dates displayed and the application will react to the selected date and reflect the value on any control that depends on it.



Fill = RGBA(0, 0, 0, 0)

Start Date 10/9/2016

Selected Date 10/12/2016

TemplateSize 76

Figure 6. Days gallery complete

4) Adding the "Previous Week" navigation.

For this we will use an Icon, so click Insert→Icon→ Chevron Left, and set the following properties:

```
Name: PrevWeekArrow
X: 0, Y: 70, Width: 36, Height: 72
Color: White
OnSelect: UpdateContext({StartDate: DateAdd(StartDate, -7,
Days)})
```

The OnSelect event is specifying to update the StartDate variable by decreasing one week from the current StartDate.

5) Adding the “Next Week” navigation.

For this add a Chevron Right with the following properties:

```
Name: NextWeekArrow
X: 600, Y: 70, Width: 36, Height: 72
Color: White
OnSelect: UpdateContext({StartDate: DateAdd(StartDate,  7)})
```

Similarly, here the OnSelect event is specifying to update the StartDate variable by adding one week from the current StartDate.

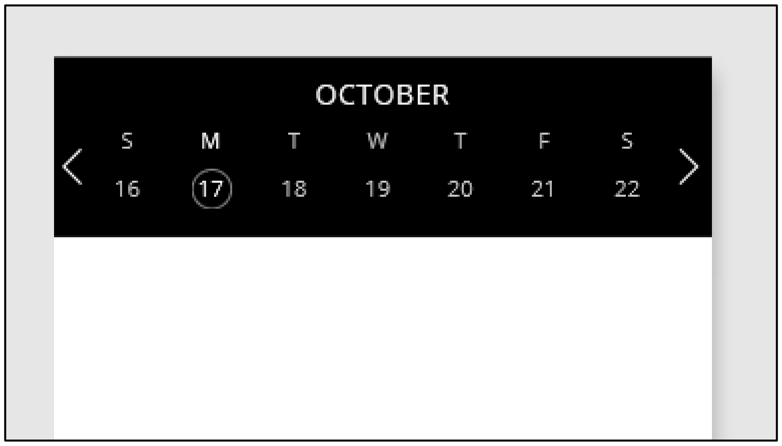


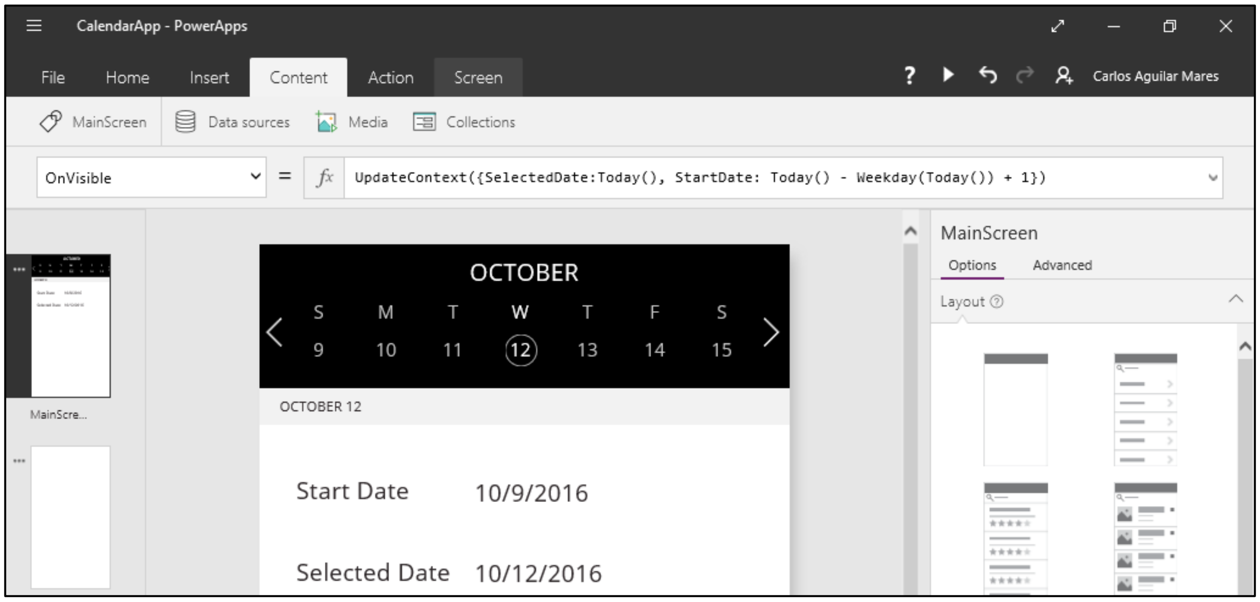
Figure 7. Previous and Next week navigation included

6) Display the selected date

We want to make sure that it is obvious to the user which date is selected so for that add a text box with the following properties:

```
Name: SelectedDateTextBox
Size: 12
X: 0, Y: 174, Width: 640, Height: 43
Fill: RGBA(242, 242, 242, 1)
PaddingLeft: 25
Text: Upper(Text(SelectedDate, "mmmm dd"))
```

At this point our date selector is completed and you can now navigate using the arrow icons, and select any date.



```
UpdateContext({SelectedDate:Today(), StartDate: Today() - Weekday(Today()) + 1})
```

Start Date 10/9/2016

Selected Date 10/12/2016

Figure 8. Date selector completed

7) Adding a create option

To complete the app, we can use a “Create” icon by clicking Insert→Icon→Add with the following properties:

```
Name: AddIcon
Color: White
X: 586, Y: 12, Width: 42, Height: 42
```

```
OnSelect: Navigate(CreateTask, ScreenTransition.Fade,
{SelectedDate: SelectedDate})
```

Here the OnSelect event is navigating to the CreateTask screen and it is passing the local (to MainScreen) variable SelectedDate to it, so that we can reference the screen easily from there.

Add a new Screen and call it "CreateTask" to allow the "+" navigation to work.

Final thoughts

You can easily add a navigation bar, and use formulas like Navigate and Back, to complete the application.
To make it part of a real application you can use now an expression such as "Filter('Sales Order', OrderDate = SelectedDate)" and bind that to a Gallery where you display all the sales orders coming from your data.

I mentioned briefly that it in our application it is important to only update the context on the OnVisible event when navigating to it for the first time so that when the user navigates to another screen and goes back we do not reset the selection that they previously had. In this case we can use an If statement to check for IsBlank to only trigger the UpdateContext in that case, for example:

```
If (IsBlank(SelectedDate),  UpdateContext({SelectedDate:Today(),
StartDate: Today() - Weekday(Today()) + 1}))
```