
You

I have a calculation to work out my work hours from a line copied from my calendar, however the calculation is giving an incorrect output, this is the input:

> Start 08:30, End 16:15, Lunch 13:10-13:45

This is the calculation (in power apps):

```
Set(Day1_string, Day1_input.Value);
Set(startTime, Mid(Day1_string, Find("Start", Day1_string) + 6, Find(",", Day1_string) - Find("Start", Day1_string) - 6));
Set(endTime, Mid(Day1_string, Find("End", Day1_string) + 4, Find("Lunch", Day1_string) - Find("End", Day1_string) - 4 - 2));
Set(lunchStart, Mid(Day1_string, Find("Lunch", Day1_string) + 6, Find("-", Day1_string) - Find("Lunch", Day1_string) - 6));
Set(lunchEnd, Mid(Day1_string, Find("-", Day1_string) + 1, Len(Day1_string) - Find("-", Day1_string)));

// Convert time values to date/time format
Set(convertedST, TimeValue(startTime));
Set(convertedET, TimeValue(endTime));
Set(convertedLS, TimeValue(lunchStart));
Set(convertedLE, TimeValue(lunchEnd));

// Calculate time differences
Set(mainDiffHours, DateDiff(convertedST, convertedET, TimeUnit.Hours));
Set(lunchDiffMinutes, DateDiff(convertedLS, convertedLE, TimeUnit.Minutes));

Set(
  adjustedWorkHours,
  mainDiffHours - (lunchDiffMinutes / 60)
);
Set(workHours, Int(adjustedWorkHours));
Set(workMinutes, (adjustedWorkHours - workHours) * 60);
Set(totalHours, totalHours + workHours);
Set(totalMinutes, totalMinutes + workMinutes);
```

This is the outpyt I'm getting:

> Your total hours for all 1 days are 7 hours and 25 minutes.
>
> Your average hours per day are: 7 hours

However, I should be getting this output:

> Your total hours for all 1 days are 7 hours and 10 minutes.
>
> Your average hours per day are: 7 hours

Copilot

It looks like there might be a small error in your calculation logic. Let's break it down step by step to identify where the discrepancy might be.

1. **Extracting Time Values:**
    - startTime: 08:30
    - endTime: 16:15
    - lunchStart: 13:10
    - lunchEnd: 13:45
2. **Converting Time Values to Date/Time Format:**
    - convertedST: 08:30
    - convertedET: 16:15
    - convertedLS: 13:10
    - convertedLE: 13:45

3. **Calculating Time Differences:**
    - mainDiffHours: Difference between convertedST and convertedET in hours.
    - lunchDiffMinutes: Difference between convertedLS and convertedLE in minutes.
4. **Adjusting Work Hours:**
    - adjustedWorkHours: mainDiffHours - (lunchDiffMinutes / 60)

Let's calculate these manually to verify:

- **Total Work Time (without lunch):**
    - From 08:30 to 16:15 is 7 hours and 45 minutes.
- **Lunch Break:**
    - From 13:10 to 13:45 is 35 minutes.
- **Adjusted Work Time:**
    - 7 hours and 45 minutes - 35 minutes = 7 hours and 10 minutes.

So, your expected output of 7 hours and 10 minutes is correct. The issue seems to be in the calculation of mainDiffHours and lunchDiffMinutes.

Possible Fix:

Ensure that mainDiffHours and lunchDiffMinutes are calculated correctly in minutes and then converted to hours accurately.

```
Set(mainDiffMinutes, DateDiff(convertedST, convertedET, TimeUnit.Minutes));
Set(lunchDiffMinutes, DateDiff(convertedLS, convertedLE, TimeUnit.Minutes));

Set(adjustedWorkMinutes, mainDiffMinutes - lunchDiffMinutes);
Set(adjustedWorkHours, adjustedWorkMinutes / 60);
Set(workHours, Int(adjustedWorkHours));
Set(workMinutes, (adjustedWorkHours - workHours) * 60);
Set(totalHours, totalHours + workHours);
Set(totalMinutes, totalMinutes + workMinutes);
```

This should give you the correct output of 7 hours and 10 minutes. Let me know if this helps or if you need further assistance!

Learn more: 1. answers.microsoft.com 2. answers.microsoft.com 3. www.calculatorsoup.com 4. youtube.com

You

Can you convert this python code to work in power apps:

```
overtime = True
hours = self.totalHours - (hoursPerDay * self.dayNum)
absoluteHours = abs(hours)
justHours = absoluteHours // 1
justMinutes = round((absoluteHours - justHours) * 60, 0)
if hours < 0:
  overtime = False
```

Copilot

Sure! Here's how you can

You

How can I create a custom component using the following code withouit leaving power apps:

```
import { IInputs, IOutputs } from "./generated/ManifestTypes";

export class FilePicker implements ComponentFramework.StandardControl<IInputs, IOutputs> {
  private _container: HTMLDivElement;
  private _notifyOutputChanged: () => void;
  private _fileContent: string;

  constructor() {}

  public init(
    context: ComponentFramework.Context<IInputs>,
    notifyOutputChanged: () => void,
    state: ComponentFramework.Dictionary,
    container: HTMLDivElement
  ): void {
    this._container = container;
    this._notifyOutputChanged = notifyOutputChanged;

    const button = document.createElement("button");
    button.innerText = "Browse File";
    button.onclick = this.openFilePicker.bind(this);
    this._container.appendChild(button);
  }

  private openFilePicker(): void {
    const options: ComponentFramework.DeviceApi.PickFileOptions = {
      accept: ["text/plain"],
      allowMultipleFiles: false
    };

    context.device.pickFile(options).then((files) => {
      if (files && files.length > 0) {
        const file = files[0];
        const reader = new FileReader();
        reader.onload = (e) => {
          this._fileContent = e.target.result as string;
          this._notifyOutputChanged();
        };
        reader.readAsText(file);
      }
    });
  }

  public updateView(context: ComponentFramework.Context<IInputs>): void {}

  public getOutputs(): IOutputs {
    return {
      fileContent: this._fileContent
    };
  }

  public destroy(): void {}
}
```

Copilot

Creating a custom component in Power Apps using the provided code involves using the Power Apps Component Framework (PCF). Here's a step-by-step guide to help you create this component without leaving Power Apps:

Step 1: Set Up Your Development Environment

- **Install Node.js**: Ensure you have Node.js installed on your machine.
- **Install Power Apps CLI**: Use the following command to install the Power Apps CLI:

```
npm install -g pac
```

Step 2: Create a New PCF Project

- **Create a New Project**: Open your terminal and run the following command to create a new PCF project:

```
pac pcf init --namespace YourNamespace --name FilePicker --template field
```

Navigate to the Project Directory:

```
cd FilePicker
```

Step 3: Replace the Code

1. **Open the Project in Your Code Editor**: Open the project folder in your preferred code editor (e.g., Visual Studio Code).
2. **Replace the Code**: Replace the contents of the index.ts file with your provided code.

Step 4: Update the Manifest File

1. **Open the ControlManifest.Input.xml File**: Update the manifest file to define the properties and events for your component.
2. **Add the Required Properties**: Ensure you have the necessary properties defined in the manifest file.

Step 5: Build and Test the Component

- **Build the Component**: Run the following command to build your component:

```
npm run build
```

2. **Test the Component**: Use the pac pcf start command to test your component in a local environment.

Step 6: Import the Component into Power Apps

- **Package the Component**: Run the following command to package your component:

```
pac solution init --publisher-name YourPublisher --publisher-prefix YourPrefix
pac solution add-reference --path ./FilePicker
pac solution pack --output ./Solution
```

2. **Import the Solution**: Import the generated solution file into your Power Apps environment.

Step 7: Use the Component in Your App

1. **Add the Component**: Once imported, you can add the custom component to your Power Apps canvas app.
2. **Configure the Component**: Configure the properties and events as needed.

By following these steps, you can create and use your custom FilePicker component in Power Apps without leaving the platform. Let me know if you need any further assistance!

You

I don't want to go into node or anything like that, how can I create it in the default power apps interface?

Copilot

Got it! If you want to create a custom component directly within the Power Apps interface without diving into Node.js or the Power Apps Component Framework (PCF), you can use the built-in features of Power Apps to achieve similar functionality. Here's a simplified approach:

Step-by-Step Guide to Create a File Picker in Power Apps

☐ **Open Power Apps Studio**:

- Go to Power Apps and open your app or create a new one.

☐ **Add a Button for File Upload**:

- Insert a button on your canvas and set its text to "Browse File".

☐ **Add a Text Input for File Content**:

- Insert a text input box where the file content will be displayed.

☐ **Add a Label for File Name**:

- Insert a label to display the selected file name.

☐ **Add a File Upload Control**:

- Power Apps doesn't have a direct file upload control, but you can use the **Attachment** control from a form. Here’s how:
  - Insert a form (e.g., Edit Form).
  - Add an attachment field to the form.
  - Hide the form but keep the attachment control visible.

☐ **Configure the Button to Trigger File Upload**:

- Set the OnSelect property of the button to trigger the attachment control:

```
AttachmentControlName.AttachFile()
```

Handle File Selection:

- Use the OnAddFile property of the attachment control to read the file content:

```
Set(fileContent, AttachmentControlName.Attachments[1].Value);
Set(fileName, AttachmentControlName.Attachments[1].Name);
```

8. **Display File Content**:
   - Set the Default property of the text input box to fileContent.
   - Set the Text property of the label to fileName.

Example Code Snippets

☐ **Button OnSelect**:

```
AttachmentControlName.AttachFile()
```

Attachment Control OnAddFile:

```
Set(fileContent, AttachmentControlName.Attachments[1].Value);
Set(fileName, AttachmentControlName.Attachments[1].Name);
```

Text Input Default:

fileContent

Label Text:

fileName

This approach allows you to create a basic file picker functionality within Power Apps without needing to use external tools or frameworks. Let me know if you need any more details or further assistance!

Conversation by Fifa Utuber with Copilot on Monday, Sep 16 2024