**Comprehensive View for Copilot Project**

You are an AI programming assistant.

* Follow the user's requirements carefully & to the letter.
* Minimize any other prose
* Use the latest knowledge of the specified technology, language
* use documentation from legitimate technology sources
* also research out from stackoverflow, medium articles, reddit r/powershell discussions, quora answers or anything that could be helpful for the solution
* If a requirement is not technically possible, tell the user.
* If you're making changes to code you've already given the user, don't give the entire file over and over, just give them the changes so they can easily copy and paste and not wait too much
* put a tag or label to the code & remember the context, so I can refer later in conversation"

I would like to create a PowerShell WPF application project called SqlQueryEditor using the Comprehensive View approach from the requirements this requirements file.

I want Copilot analyze and rewrite the Comprehensive View to include any proof reading errors and include any suggestions and questions needing additional details and clarifications before being requested to generating a solution.

**Comprehensive View Outline for PowerShell WPF SqlQueryEditor Application**

**1. Objective / Purpose**

* To create a user-friendly PowerShell WPF Master-Detail Application and interface called "SqlQueryEditor" with the purpose of the creation, storage, and execution of SQL Queries with associated SQL Query Parameter values that are stored in a SQL Express Database.
* The application will include the ability to Select, Add, Edit, Save, Delete, and Execute a selected SQL Query with any defined associated parameter values.
* Copilot will generate the necessary code for user authentication to access the SQL Express database and outline how to add and configure the user identity and passwords.
* Future support for SQL Server will be needed.

**2. Requirements**

**Application Setup and Configuration Requirements**

* Database will be SQL Express using database file "F:\DATA\BILLS\PSSCRIPTS\SCANMYBILLS\DATABASE1.MDF".
* The SqlConnectionString is:

$ConnectionString = "Server=(localdb)\MSSQLLocalDB;AttachDbFilename=F:\DATA\BILLS\PSSCRIPTS\SCANMYBILLS\DATABASE1.MDF;Integrated Security=True"

* The application folder path where the application files are located is: F:\Data\Bills\PSScripts\ScanMyBills
* The working folder where temporary and log files are located is: C:\Temp\SqlQuery
  + Transcript Log file name is: SqlQuery\_Transcript.log
  + User Activity Log file name is: SqlQuery\_User\_Activity.log

**PowerShell Functional Requirements**

* PowerShell code should be compatible with PowerShell Core Version 7.4 or higher to Ensure compatibility details are aligned with the intended deployment environment.
* PowerShell Code should use Write-Host and not Write-Output because Write-Output is a PowerShell version 1.0 approach that throws message into the Pipeline which was the only way to directly the message to the console. Write-Output is also much slower than Write-Host which is designed to write to standard output / console.
* Database Columns and Class Property Names that contain the underscore character "\_" should be quoted when referenced in the PowerShell code, for example: $query.'Parm\_Id'.
* Include Write-Host, Write-Warning, Write-Error, and Write-Verbose statements to help with debugging and logging process flow.
* Start-Transcript Logging will be used to capture console output.
* User activity should be logged to a file. Copilot should code a best practice approach to format log messages of UI events initiated by user actions
* PowerShell XAML code will be saved as a separate .XAML file and the script will load the XAML file.
* PowerShell XAML does not support the inclusion of Event Handlers such as Click="OnSaveChanges" or SelectedItemChanged="OnTreeViewSqlQueriesSelectionChanged". The evets handlers must be defined in PowerShell code and added to the component use the components’s Add\_ Method .
* Synchronized HashTable Variable, $SyncHash, will be used for configuration of Global Variables which are added to the HashTable. This is helpful with Parallel processing, inter-process and inter-form communication.

$SyncHash = [System.Collections.Hashtable]::Synchronized((New-Object System.Collections.Hashtable))

* **Data Model Requirements**
  + Master-Detail Data Model: Consists of two database tables: [dbo].[SqlQuery] as the Master and [dbo].[SqlQueryParms] as Detail.
  + [dbo].[SqlQuery]: Contains a column SqlFormat whose value is a valid SQL Query.
  + SQL Parameters: The SQL Query may include the use of SQL Parameters created and stored in the [dbo].[SqlQueryParms] table.
  + Foreign Key: The [dbo].[SqlQuery].Parm\_Id will be the associated Foreign Key to the Id Column of the [dbo].[SqlQueryParms] table.
  + Execution: At the time of SQL Query execution, the values for any Parameters are provided from the associated [dbo].[SqlQueryParms].
  + SQLDataAdapters: Application should make use of SQLDataAdapters for connections between UI Components and the Database Tables.
  + Data Classes: Each database table should have a separate corresponding Object or instance of a Data Class that has the result of the SQLDataAdapter DataSet result. This avoids the use of hard-coded ViewModel and System.Collections.ObjectModel.ObservableCollection.
  + Master Table Name: "SqlQuery" Schema

CREATE TABLE [dbo].[SqlQuery] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[DataSource] VARCHAR (22) NOT NULL,

[Parm\_Id] INT NULL,

[Name] VARCHAR (126) NOT NULL,

[Description] VARCHAR (MAX) NOT NULL,

[SqlFormat] VARCHAR (MAX) NOT NULL,

[IsSelected] BIT NOT NULL,

[IsEnabled] BIT NOT NULL,

[IsEditable] BIT NOT NULL,

[Active] BIT NOT NULL,

[CreatedOn] DATETIME2 (7) NOT NULL,

[DeactivatedDate] DATETIME2 (7) NULL,

PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_SqlQuery\_SqlQueryParms] FOREIGN KEY ([Parm\_Id]) REFERENCES [SqlQueryParms]([Id])

);

* + Detail Table Name: "SqlQueryParms" Schema

CREATE TABLE [dbo].[SqlQueryParms] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[SqlQuery\_Id] INT NULL,

[Seq] INT NOT NULL,

[Name] VARCHAR (120) NOT NULL,

[Prompt] VARCHAR (128) NOT NULL,

[Description] VARCHAR (MAX) NOT NULL,

[ValueType] VARCHAR (20) NOT NULL,

[Value] VARCHAR (120) NOT NULL,

[IsEditable] BIT NOT NULL,

[Active] BIT NOT NULL,

[CreatedOn] DATETIME2 (7) NOT NULL,

[DeactivatedDate] DATETIME2 (7) NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

);

* + Column [dbo].[SqlQuery].Parm\_Id maps to Column [dbo].[SqlQueryParms].Id in the SqlQueryParm table. This is the Master-Detail relationship.
  + Column [DeactivatedDate] is a [nullable datetime] value.
  + The LoadData function needs to include checks to handle null values for the CreatedOn and DeactivatedDate fields where CreatedOn is set to the current DateTime and DeactivatedDate is set to a nullable DateTime.
* **Non-Functional Requirements**
  + User Interface: The UI should be intuitive and user-friendly, allowing users to perform tasks with minimal effort.
  + Data Management: Data management with UI Components should be dynamic, with data models defined from database table schemas and not dependent on hard-coded ObservableCollection classes.
  + Error Handling: The application should gracefully handle errors and provide meaningful error messages to users. Copilot should generate a best practice approach for error handeling.
  + Code Quality: The code should be modular, well-documented, and adhere to coding standards to facilitate maintenance and updates.

**3. Design**

**UI Layout**

* The Windows configuration will create a basic Tab Control framework to allow for future expansion.
* The "SqlQueryEditor" application project will focus on functionality needed for the \<TabItem x:Name="tbSelection" defined below.

**Windows**

* **Main window class will have the following XAML configuration:**

<Window x:Class="ScanMyBills.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:system="clr-namespace:System;assembly=mscorlib"

xmlns:local="clr-namespace:ScanMyBills"

mc:Ignorable="d"

Title="Scan My Bills Explorer"

Height="800" Width="1200" Topmost="False"

ResizeMode="CanResizeWithGrip" ShowInTaskbar = "True"

WindowStartupLocation = "CenterScreen"

x:Name="MainForm"

FocusManager.FocusedElement="{Binding ElementName=\_scriptView}"

Background="AliceBlue" UseLayoutRounding="True">

</Window>

* **Top Level Control will be a TabControl with XAML configuration:**

<TabControl x:Name="\_tabControl" Grid.Column="0"

Margin="5,85,5.333,5" Padding="0,0,0,0"

HorizontalAlignment="Stretch" VerticalAlignment="Stretch"

Background="{StaticResource TabBackgroundBrush}" BorderThickness="2" BorderBrush="{StaticResource SolidBorderBrush}"

SelectedIndex="0" IsSynchronizedWithCurrentItem="True" Grid.RowSpan="2">

</TabControl>

* **\_tabControl will have 3 TabItems with the following XAML Configuration:**
  + **\<TabItem x:Name="tbDocumentTree" Tag="0" Header="Document Management" ToolTip="Browsing, Scanning, and Managing Documents">**
  + **\<TabItem x:Name="tbLkUpTableEdit" Tag="1" Header="Data Grid" ToolTip="Edit Data and Lookup Tables">**
  + **\<TabItem x:Name="tbSelection" Tag="3" Header="SQLQuery Editor" ToolTip="Select SQL Queries to Execute, Edit, Delete, or Add to SQL Query Database">**

**Layout and Controls for \<TabItem x:Name="tbSelection"**

* **Has a vertical splitter that divides the Tabbed Item into two panels 20% on left, 80% on right. The vertical splitter could benefit if it was adjustable.**
* **The vertical splitter's left side Panel has a treeview component that displays data from the [dbo].[SqlQuery] database table. The data will be a TextBlock of composite [dbo].[SqlQuery] columns DataSource and Name columns with a tooltip showing the Description.**
* **The vertical splitter's Right Panel has two stacked DataGrid components for the Master-Data contents.**
  + **The Top DataGrid will have the Master Details from the [dbo].[SqlQuery] table showing the Selected record from the TreeView.**
  + **The Bottom DataGrid will have the Details from the SqlQuery and SqlQueryParms Tables showing the SqlQueryParms associated with the selected Master record. An SqlQuery may or may not have any SqlQueryParms associated with it.**

**Navigation**

* **SqlQueryParms Table DataGrid will always show the Records associated with the selected record in the SqlQuery DataGrid.**
* **There needs to be the ability to toggle between Record Edit Mode or Table Edit Mode.**
  + **Record Edit Mode is the normal select SqlQuery record in TreeView and see the selected record in the SqlQuery DataGrid and the associated records in the SqlQueryParms DataGrid.**
  + **Table Edit Mode will populate the top Master DataGrid with all the SqlQuery Records. Selecting a Query in the TreeView will only navigate to the record in the SqlQuery DataGrid instead of only showing that record as with Record Edit Mode.**
* **Action Bars of either Menus or Buttons will be needed to manage the actions for CRUD and other aspects of data management and navigation.**
  + **A separate Action Bars will be needed for each DataGrid**
  + **SqlQueryParms DataGrid will need Add, Cancel, Save, Delete actions**
  + **SqlQuery DataGrid in Record Edit Mode will need Export, Add, Cancel, Save, Delete actions**
  + **SqlQuery DataGrid in Table Edit Mode will need Export, First, Previous, Add, Edit, Cancel, Save, Delete, Next, Last actions**

**Actions**

* **Execute: Enabled only in Record Edit Mode. Will combine with associated parameter values and execute the SQL Query. Results are returned in a DataSet and displayed in a Data Grid defined in the \<TabItem x:Name="tbLkUpTableEdit".**
* **Export option with Submenus**
  + **CopyToClipboard: populated SQL Query to Clipboard.**
  + **ExportAsCSV: The SqlQuery and SqlQueryParms Tables are saved as separate CSV Files.**
  + **ExportAsExcel: The SqlQuery and SqlQueryParms Tables as separate worksheets to an Excel File to be used as a backup.**
  + **ImportFromExcel: Used to restore the SqlQuery and SqlQueryParms Table data from an ExportAsExcel file.**
* **Refresh: To refresh the data in the UI Data components from the Database.**
* **First: Navigate to the First Record.**
* **Previous: Navigate to the previous record.**
* **Add: Add a new Record.**
* **Edit: Edit selected record.**
* **Cancel: Cancel any changes.**
* **Save: Save Changes.**
* **Delete: Delete Selected Record.**
* **Next: Navigate to Next Record.**
* **Last: Navigate to Last Record.**

**Mockups**

* **Not sure how to define and present Visual mockups that are usable for Copilot’s solution generation.**

**4. Existing Code**

**PowerShell Scripts**

**Initialization Code**

#region Initialize Variables

#--------------------------------------------------------

# Pre-Process and STA Apartment State check

#--------------------------------------------------------

if ([System.Threading.Thread]::CurrentThread.ApartmentState -ne 'STA') {

$CommandLine = $MyInvocation.Line.Replace($MyInvocation.InvocationName, $MyInvocation.MyCommand.Definition)

Write-Warning 'Script is not running in STA Apartment State.'

Write-Warning ' Attempting to restart this script with the -Sta flag.....'

Write-Verbose " Script: $CommandLine"

Start-Process -FilePath PowerShell.exe -ArgumentList "$CommandLine -Sta"

exit

}

#----------------------------------------------------------------------------------

# Initialize Constants

#----------------------------------------------------------------------------------

$eol = [Environment]::NewLine

$StartTime = Get-Date

$TimeStamp = $StartTime.ToString('yyyy-MMdd-HHmm') # Time Stamp appended to Saved output files

$Error.Clear()

#----------------------------------------------------------------------------------

# Initialize Script Variables

#----------------------------------------------------------------------------------

(Get-Variable -Name 'syncHash' -ErrorAction SilentlyContinue) | Remove-Variable

If ([String]::IsNullOrEmpty((Get-Variable -Name 'syncHash' -ErrorAction SilentlyContinue))) {

[hashtable]$syncHash = @{}

$syncHash.Add('Form', [object])

$syncHash.Add('Errors', [System.Collections.ArrayList]@())

$syncHash.Add('Params', [PSCustomObject]@{

Title = 'ScanMyBills'

DocumentFolder = 'F:\Data\Bills'

LogFolder = 'F:\Data\Bills\ScanMyBills'

LogFileNamePattern = '{0}\Transcript\_Log\_{1}.txt'

ScriptsFolder = 'F:\Data\Bills\PSScripts\ScanMyBills'

Cred = [System.Management.Automation.PSCredential]::Empty

SqlServer = '(localdb)\MSSQLLocalDB'

DatabaseName = 'F:\DATA\BILLS\PSSCRIPTS\SCANMYBILLS\DATABASE1.MDF'

ConnectionString = [String]::Empty

IsRunning = $false

})

$syncHash.Add('UI', [PSCustomObject]@{

SqlResults = $null

})

# Initialize Connection String

# $syncHash.Params.ConnectionString = "Data Source={0};Initial Catalog={1};Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False" -f $syncHash.Params.SqlServer, $syncHash.Params.DatabaseName

$syncHash.Params.ConnectionString = "Data Source={0};AttachDbFilename={1};Integrated Security=True" -f $syncHash.Params.SqlServer, $syncHash.Params.DatabaseName

}

#endregion

* **Code to load and preprocess the XAML file**
  + **Load XAML**

#region Load XAML

$inputXML = [String]::Empty

# $inputXML = @"

# "@

$inputXML = Get-Content -Path "$($syncHash.Parms.ScriptsFolder)\SqlQuery.xaml" -Raw

#endregion

* + **Load and Process XAML and Create Form Variables**

#region Load and Process XAML and Create Form Variables

# Cleanup XAML by removing unwanted namespaces and expanding Win to Window

$inputXML = $inputXML -replace 'mc:Ignorable="d"','' -replace "x:Na",'Na' -replace '^<Win.\*', '<Window' -replace '!replace-with-xmlDataSource!',$SyncHash.Params.xmlDataSource

Write-Host ("Saved xaml to: ({0})" -f 'C:\Temp\SqlQuery.xaml')

Set-Content -Value $inputXML -Path "C:\Temp\$($SyncHash.Params.Title).xaml" -Force

# Load Any Assemblies

[void][System.Reflection.Assembly]::LoadWithPartialName('presentationframework')

# Convert XAML input string to XML Object

[xml]$xaml = $inputXML

$xaml.Window.RemoveAttribute('x:Class')

$xaml.Window.RemoveAttribute('mc:Ignorable')

# Check for a text changed value (which we cannot parse)

If ($xaml.SelectNodes("//\*[@Name]") | ? TextChanged) {

Write-Host ("Error: This Snippet can't convert any lines which contain a 'textChanged' property. `n please manually remove these entries")

$xaml.SelectNodes("//\*[@Name]") | ? TextChanged | % {

write-warning "Please remove the TextChanged property from this entry $($\_.Name)"

}

Return

}

$reader = (New-Object System.Xml.XmlNodeReader $xaml)

Try {

$SyncHash.Form = [Windows.Markup.XamlReader]::Load( $reader )

} Catch [System.Management.Automation.MethodInvocationException] {

Write-Warning "We ran into a problem with the XAML code. Check the syntax for this control..."

Write-Host ("Error: $($error[0].Exception.Message)") -ForegroundColor Red

If ($error[0].Exception.Message -like "\*button\*") {

write-warning "Ensure your &lt;button in the `$inputXML does NOT have a Click=ButtonClick property. PS can't handle this`n`n`n`n"

}

} Catch {

Write-Host "Unable to load Windows.Markup.XamlReader. Double-check syntax and ensure .net is installed."

Write-Error ($Error[0] | Out-String)

}

# Create a PowerShell Variable for each Form Object defined in the XAML for ONLY x:Name= that begin with "\_"

$xaml.SelectNodes("//\*[@Name]") | Where-Object { $\_.Name -like "\_\*" } |

ForEach-Object {

$FormObjects += "WPF$($\_.Name)"

Set-Variable -Name "WPF$($\_.Name)" -Value $SyncHash.Form.FindName($\_.Name)

}

#endregion

* **Form Object Variables**

#region Form Object Variables

Function Get-FormVariables {

if ($global:ReadmeDisplay -ne $true) {

Write-Host "If you need to reference this display again, run Get-FormVariables"

}

Write-Host ("`n$("-" \* 65)`nFound the following interactable elements from our form`n$("-" \* 65)")

Get-Variable WPF\*

Write-Host ("$("-" \* 65)`n")

}

# Dump list of Form Object Variables

Get-FormVariables

#endregion

* **Code for ContentRendered() Event**

$SyncHash.Form.Add\_ContentRendered({

Param ([object]$sender, [System.EventArgs]$e)

Write-Host ("`$SyncHash.Form.ContentRendered() {0}: {1}({2})" -f $sender.Name, $e.Source.Name, $e.ToString())

$SyncHash.Form.Title = $SyncHash.Params.Title

Write-Host "Initalize scanMyBillsSqlQuery Object and Populate any DataSets" -ForegroundColor Green

If ($SyncHash.Params.ConnectionString) {

} Else {

$ConnectionString = $SyncHash.Params.ConnectionString

}

})

* **Code for SourceInitialized() Event Handler**

#region SourceInitialized Event

$SyncHash.Form.Add\_SourceInitialized({

Write-Host ("`$SyncHash.Form.SourceInitialized(): Before the window form is displayed ")

## We'll create a timer

$syncHash.UI.timer = [System.Windows.Threading.DispatcherTimer]::new()

## Which will fire 2 times every second

$syncHash.UI.timer.Interval = [TimeSpan]"0:0:0.150"

## And will invoke the $updateMessage handler

$syncHash.UI.timer.Add\_Tick( $updateMessage )

## Now start the timer running

$syncHash.UI.timer.Start()

if( $syncHash.UI.timer.IsEnabled ) {

Write-Host ("`$SyncHash.Form.SourceInitialized(): Message Handler is running")

} else {

Write-Host ("`$SyncHash.Form.SourceInitialized(): Message Handler Timer didn't start")

}

})

#endregion

* **Code for Closing() Event Handler**

#region Closing Event

$SyncHash.Form.Add\_Closing({

Param ([object]$sender, [System.EventArgs]$e)

Write-Host ("`$SyncHash.Form.Closing() {0}: {1}({2})" -f $sender.Name, $e.Source.Name, $e.ToString())

Write-Host ('$SyncHash.Form.add\_Closing: {0}' -f ($e | Select-Object -Property \* | FL | Out-String)).Trim()

# Check DataGrids for unsaved edits

Sync-GridChanges

# Clear scanMyBillsSqlQuery Object

If ($syncHash.UI.DataSet) {

$syncHash.UI.DataSet.CloseConnection()

}

If ($syncHash.UI.SqlResults) {

$syncHash.UI.SqlResults.CloseConnection()

}

$syncHash.Running = $false

Write-Host "Shutting down Messages Handler..."

$syncHash.UI.timer.Stop()

While ($syncHash.UI.timer.IsEnabled) {

Write-Host "Shutting down Messages Handler..."

Sleep -Seconds 1

}

})

#endregion

* **Code for Closed() Event Handler**

#region Closed() Event for Variable Cleanup

$SyncHash.Form.add\_Closed({

Param ([object]$sender, [System.EventArgs]$e)

Write-Host ("`$Sync[\_{{{CITATION{{{\_1{](https://github.com/1RedOne/1redone.github.io/tree/113ee30d8762cf65c1650540c6150d213b7bff1c/series%2Fsnippet-xamlToGui.md)

#endregion

* **Dependencies**
  + Presentationframework for WPF support
  + PowerShell module ImportExcel for reading and writing Excel File used in Exports and imports of database table data

**5. Challenges**

* **Technical Challenges**:
  + The Data Components needs to be dynamic in nature to avoid having to define specific column and field elements in code. The necessary configurations should be derived from table schemas and not hard coded in the PowerShell script.
  + The enabling and disabling of Action Bar items based on states of TreeView and DataGrid states and user interaction with UI components.
  + How best to bind UI Components to data that is maintained in SQL Database Tables.
  + How best to manage CRUD actions between UI Components and the database table data.
  + How to make the SQL Query Editor application a separate script that can be called from a parent PowerShell WPF application that needs to create and execute SQL Queries?
    - In the development of the SQL Query Editor application, if called as a separate application, how to launch, navigate between parent application and the SQL Query Editor, and what happens when the parent application is closed when SQL Query Editor is still open.
* **Conceptual Challenges**
  + Goal is to create reusable Scripts, modules, and components as building block in other PowerShell WPF applications.
  + Should SQL Query Editor application be a separate script that is executed from a parent PowerShell WPF application or integrated into the parent application as a TabItem in a TabControl?

**6. Implementation Plan**

* **Step-by-Step Plan: Have Copilot generate the details of the Implementation Plan with the following:**
  1. Set up the development environment
     + How to create a GitHub Project for the SQL Query Editor application
  2. Create the initial project structure.
  3. Develop the UI components.
  4. Implement the backend logic.
  5. Integrate the UI with the backend.
  6. Test and debug the application.
  7. Deploy the application.

**7. Testing**

* **Test Cases**
  + Manually drag the window grabber to see how the windows and components scale
  + Manually test navigation between components
  + Manually test action bar items and data entry
  + Manually verify data changes are showing up in the database tables and are reflected in the UI data aware components

**8. Documentation**

* **User Guide**
  + Have Copilot generate a User Guide.
* **Developer Guide**
  + Have Copilot generate a Developer Guide