MessagePanel is a UserControl that can be used in place of the standard MessageBox. It provides much of the same functionality but in a more flat UI and includes “shading” over the main window area to not make the message not only modal functionally but also visually.

The MessagePanel works by the consuming code setting a local value indicating the local operation for which the message is being displayed, showing the message, and then acting upon the user’s response.

# Consuming MessagePanel

In your view model add two things:

* An enumeration that identifies each action that can be taken; e.g., MessageAction.
* A property to contain the current MessageAction in play; e.g., currentMessageAction.
* A property to which the MessagePanel.MessageResponse property will be bound; e.g, MainMessageResponse;

In your window XAML, add a reference to the Common.UserControls namespace and add an entry for the user control as a peer to your main control:

<Window x:Class="YourApp.Views.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:uc="clr-namespace:Common.UserControls;assembly=Common"

...

>

<Grid>

...

</Grid>

<uc:MessagePanel Name="MessagePanel" MessageResponse="{Binding MainMessageResponse}" />

</Window>

MessagePanel provides several variations of its Show() method similar to the standard MessageBox. The various options support whether or not to have a title, which buttons to show, etc. NOTE: MessagePanel does not currently provide for any icons to be displayed.

MessagePanel provides three message types with three pre-defined button sets:

MessagePanel.MessageType.Ok

MessagePanel.MessageType.OkCancel

MessagePanel.MessageType.YesNo

# Using MessagePanel

MessagePanel is invoked via one of its Show() methods. The view model sets the currentMessageAction indicating why the message is being displayed and then MessagePanel.Show() with the desired options.

The view model must contain a switch(currentMessageAction) – or some other means – of determining the action to take based upon: a) the user selection, and b) the previously set currentMessageAction.

Upon the user response, the window is closed, and the code in the setter of the view model’s bound property (e.g., MainMessageResponse) determines how to proceed by examining the user’s response and the previously set currentMessageAction.

For example:

currentMessageAction = MessageAction.DeleteTransaction;

AssociatedWindow.MessagePanel.Show("Verifying", "The transaction is marked Cleared or Resolved. Are you sure you want to delete this?");

This displays a [Yes] and [No] button by default. If the user clicks [No], the bound property is set to “Halt”; if [Yes], it is set to “Proceed”; and the message is hidden.

An example view model implementation is below:

#region MessagePanel

public enum MessageAction

{

Acknowledge,

DeleteAccount,

DeleteTransaction,

DeleteSubtransaction,

MergeTransactions,

ResolveTransfers

}

private MessageAction currentMessageAction;

private string mainMessageResponse;

public string MainMessageResponse

{

get => mainMessageResponse;

set

{

mainMessageResponse = value;

NotifyPropertyChanged();

if (mainMessageResponse == "Proceed")

{

switch (currentMessageAction)

{

case MessageAction.Acknowledge:

break;

case MessageAction.DeleteAccount:

// Delete the account

break;

case MessageAction.DeleteTransaction:

// Delete the transaction

break;

case ...

break;

default:

break;

}

}

}

}

#endregion