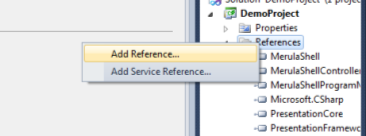
# Creating a taskbar

In this tutorial i will explain how to create a windows taskbar in c#, with the Merula Shell Library.

## Requirements

* Visualstudio 2010
* Merula Shell library

First create a new project in visualstudio. In the new project window choose for a WPF application. Add the three dlls from the Merula shell library to the reference of the project.



## Creating a usercontrol

For every window I want to create a fancy button with an image and the tile, in my taskbar. So let’s create a usercontrol, I called it TaskbarButton.

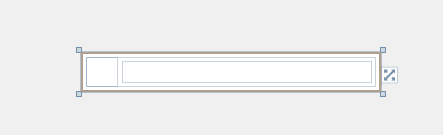
|  |
| --- |
| <UserControl x:Class="DemoProject.TaskbarButton"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  mc:Ignorable="d"  d:DesignHeight="300" d:DesignWidth="300">  <Grid Margin="5">  <Grid.ColumnDefinitions>  <ColumnDefinition Width="32" />  <ColumnDefinition Width="\*" />  </Grid.ColumnDefinitions>  <Image Name="imgIcon"/>  <TextBlock Name="lblTitle" Grid.Column="1" Margin="4" />  </Grid>  </UserControl> |

In the usercontrol add an image control and a textblock control.

Add the following code in the codebehind from the usercontrol:

|  |
| --- |
| using System;  using System.Windows.Controls;  using Window = MerulaShell.windows.Window;  namespace DemoProject  {  /// <summary>  /// Interaction logic for TaskbarButton.xaml  /// </summary>  public partial class TaskbarButton : UserControl  {  private readonly Window window;  //Contructor with a MerulaShell.windows.window as input  public TaskbarButton(Window window)  {    InitializeComponent();  this.window = window;  window.TitleChanged += WindowTitleChanged; //when the title of the window changes  SetProperties(); //set the window properties  }  private delegate void DelegateVoid();  void WindowTitleChanged(object sender, EventArgs e)  {  Dispatcher.Invoke(new DelegateVoid(SetTitle)); //invoke beacause merula shell runs in another thread  }  private void SetTitle()  {  lblTitle.Text = window.Title; // sets the title in the textblock  }  private void SetProperties()  {  imgIcon.Source = window.ProgramIcon; // sets the icon of the window  lblTitle.Text = window.Title; // sets the title in the textblock  }  }  } |

Your usercontrol will now look something like this:



## Loading windows

The next step is loading the windows in our taskbar. Go to the xaml of the main window and replace the grid with the stackpanel.

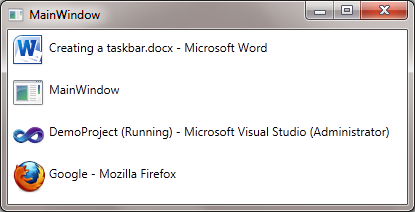
Mine main window xaml:

|  |
| --- |
| <Window x:Class="DemoProject.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="MainWindow" Height="350" Width="525">  <StackPanel Name="pnlTasks">    </StackPanel>  </Window> |

Add the following code to the codebehind of the mainwindow:

|  |
| --- |
| using System;  using System.Windows;  using MerulaShellController.ManageWindows;  namespace DemoProject  {  /// <summary>  /// Interaction logic for MainWindow.xaml  /// </summary>  public partial class MainWindow : Window  {  private ManageWindows windowManager;  public MainWindow()  {  InitializeComponent();  windowManager = new ManageWindows(); //create a new windowmanager / only one needed  Closed += MainWindow\_Closed; // on close event  windowManager.WindowListChanged += WindowManagerWindowListChanged; //when the list of windows is changed  LoadWindows(); //load the windows  }  private delegate void DelegateVoid();  void WindowManagerWindowListChanged(object sender, EventArgs e)  {  //invoke beacause merula shell runs in another thread  Dispatcher.Invoke(new DelegateVoid(LoadWindows));  }  void MainWindow\_Closed(object sender, EventArgs e)  {  Environment.Exit(0); //stops the merula shell  }  private void LoadWindows()  {  ClearTasks();//delete old tasks  var windows = windowManager.GetWindows();//windowManager.GetWindows() returns all the active windows  foreach (var window in windows)  { //foreach window add a taskbar button  pnlTasks.Children.Add(new TaskbarButton(window));  }  }  private void ClearTasks() //delete old tasks  {  pnlTasks.Children.Clear();  }  }  } |

As a result we have now a window with the current active windows. The list of windows will be updated when the number of windows change and if a tile of a window changes.



## Making it interactive

You still can’t interact with the windows, like minimize or maximize them. Open up the TaskButton usercontrol. Add a MouseUp handler to the usercontrol tag.

|  |
| --- |
| MouseUp="UserControlMouseUp" |

Add the following code to the codebehind of the usercontrol.

|  |
| --- |
| private bool active;  private void UserControlMouseUp(object sender, System.Windows.Input.MouseButtonEventArgs e)  {  Background = Brushes.LightBlue; //set a nice active color  if(active) //when active minimize and maximize  window.MaximizeMinimize(); //minimize or maximize  else  {  active = true; // set active  window.SetToForeground(); //set window to foreground  }  InvokeActivated(new EventArgs());  }  public void SetNonActive()  {  active = false; //set active to false  Background = Brushes.White; //reset color to white  }  public event EventHandler Activated; //event to notify the mainwindow  public void InvokeActivated(EventArgs e)  {  EventHandler handler = Activated;  if (handler != null) handler(this, e);  } |

You’ll also have to edit the code behind of the mainwindow. Change the method LoadWindows() to:

|  |
| --- |
| private void LoadWindows()  {  ClearTasks();//delete old tasks  var windows = windowManager.GetWindows();//windowManager.GetWindows() returns all the active windows  foreach (var window in windows)  { //foreach window add a taskbar button  var button = new TaskbarButton(window);  button.Activated += ButtonActivated; //add a event to the taskbarbutton  pnlTasks.Children.Add(button);  }  } |

Last but not least you’ll have to create an event handler. Add it to the codebehind of the mainwindow:

|  |
| --- |
| void ButtonActivated(object sender, EventArgs e)  {  var senderButton = (TaskbarButton) sender; //the sender button  var otherButtons = pnlTasks.Children.OfType<TaskbarButton>().Where(b => b != senderButton); //select the other buttons  foreach (var otherButton in otherButtons)  {  otherButton.SetNonActive(); //sets the other buttons to nonactive  }  } |

As result you have a working taskbar.

