How to set up custom window size:

Method 1 → use WindowEx
Method 2 → use AppWindow

Method one : WindowEx

Note that the WindowEx package is only compatible with net5.0 and lower, with net6.0 and higher it wont work

– Create a new blank Application project in winui3

in the solution explorer you should see something like this:

Solution 'name of solution' (2/2)
v 'solution name'
    > Depencies
    > Properties
    app.manifest
    v App.xaml
        - App.xaml.cs
    v MainWindow.xaml
        - MainWindow.xaml.cs

to install the WindowEx:

- Right Click on Depencies
- Manage Nugget Packages
- In the search box on the top type: "WindowEx"
- Once you've found the package install it.

once you've installed the package, go inside the App.xaml.cs class,
the code inside it should look like this:

```
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Controls.Primitives;
using Microsoft.UI.Xaml.Data;
using Microsoft.UI.Xaml.Input;
using Microsoft.UI.Xaml.Media;
using Microsoft.UI.Xaml.Navigation;
using Microsoft.UI.Xaml.Shapes;
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;

// To learn more about WinUI, the WinUI project structure,
// and more about our project templates, see: http://aka.ms/winui-project-info.

namespace 'solution name'
{
        /// <summary>
        /// Provides application-specific behavior to supplement the default Application class.
        /// </summary>
        public partial class App : Application
        {
                /// <summary>
                /// Initializes the singleton application object.  This is the first line of authored code
                /// executed, and as such is the logical equivalent of main() or WinMain().
                /// </summary>
                public App()
                {
                        this.InitializeComponent();
                }

                /// <summary>
                /// Invoked when the application is launched.
                /// </summary>
                /// <param name="args">Details about the launch request and process.</param>
                protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
                {
                        m_window = new MainWindow();
                        m_window.Width = 600;
                        m_window.Height = 500;
                        m_window.Activate();
                }

                private Window m_window; → Change this into private WindowEx m_window
        }
}
```

Method two : AppWindow

- Create a new Blank app project

Solution 'name of solution' (2/2)
v 'solution name'
    > Depencies
    > Properties
    app.manifest
    v App.xaml
        - App.xaml.cs
    v MainWindow.xaml
        - MainWindow.xaml.cs

inside the App.xaml.cs the code should look like this:

```
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Controls.Primitives;
using Microsoft.UI.Xaml.Data;
using Microsoft.UI.Xaml.Input;
using Microsoft.UI.Xaml.Media;
using Microsoft.UI.Xaml.Navigation;
using Microsoft.UI.Xaml.Shapes;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;

// To learn more about WinUI, the WinUI project structure,
// and more about our project templates, see: http://aka.ms/winui-project-info.

namespace 'solution name'
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    public partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
```

```
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
        }

        /// <summary>
        /// Invoked when the application is launched.
        /// </summary>
        /// <param name="args">Details about the launch request and process.</param>
        protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
        {
            m_window = new MainWindow();
            m_window.Activate();
        }

        private Window m_window;
    }
}
```

We are going to use the AppWindow class to set our custom size for the application.

- Declare the _AppWindow variable (page 5)
- Declare the GetAppWin() method (page 6)
- Set the _AppWindow variable (page 8)

Declaring the _AppWindow variable:

```
namespace 'solution name'
{
	/// <summary>
	/// Provides application-specific behavior to supplement the default Application class.
	/// </summary>
	public partial class App : Application
	{
		/// <summary>
		/// Initializes the singleton application object.  This is the first line of authored code
		/// executed, and as such is the logical equivalent of main() or WinMain().
		/// </summary>
		public App()
		{
			this.InitializeComponent();
		}

		/// <summary>
		/// Invoked when the application is launched.
		/// </summary>
		/// <param name="args">Details about the launch request and process.</param>
		protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
		{
			m_window = new MainWindow();
			m_window.Activate();
		}

		private Window m_window;
		private AppWindow _AppWindow; →Declare the _AppWindow variable
		// AppWindow is part of the using: Microsoft.UI.WIndowing
	}
}
```

Declaring the GetAppWin() method:

```
namespace 'solution name'
{
	/// <summary>
	/// Provides application-specific behavior to supplement the default Application class.
	/// </summary>
	public partial class App : Application
	{
		/// <summary>
		/// Initializes the singleton application object.  This is the first line of authored code
		/// executed, and as such is the logical equivalent of main() or WinMain().
		/// </summary>
		public App()
		{
			this.InitializeComponent();
		}

		/// <summary>
		/// Invoked when the application is launched.
		/// </summary>
		/// <param name="args">Details about the launch request and process.</param>
		protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
		{
			m_window = new MainWindow();
			m_window.Activate();
		}

		private AppWindow GetWinApp()
		{
			IntPtr HWND = WindowNative.GetWindowHandle(m_window);
			if (HWND != IntPtr.Zero)
			{
				WindowId WINID = Win32Interop.GetWindowIdFromWindow(HWND);
				return AppWindow.GetFromWindowId(WINID);
			}
			return null;
		}

		private Window m_window;
		private AppWindow _AppWindow;
	}
}
```

if you want to use the GetWinApp() method in other classes like in the MainWindow.xaml.cs you can modify the method like this:

```
public AppWindow GetWinApp(object _target)
{
        IntPtr HWND = WindowNative.GetWindowHandle(_target);
        if (HWND != IntPtr.Zero)
        {
                WindowId WINID = Win32Interop.GetWindowIdFromWindow(HWND);
                return AppWindow.GetFromWindowId(WINID);
        }
        return null;
}
```

Set the _AppWindow variable:

```
namespace 'solution name'
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    public partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
        }

        /// <summary>
        /// Invoked when the application is launched.
        /// </summary>
        /// <param name="args">Details about the launch request and process.</param>
        protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
        {
            m_window = new MainWindow();
            _AppWindow = GetAppWin(); → add this line to set the _AppWindow variable
            m_window.Activate();
        }

        private AppWindow GetWinApp()
        {
            IntPtr HWND = WindowNative.GetWindowHandle(m_window);
            if (HWND != IntPtr.Zero)
            {
                WindowId WINID = Win32Interop.GetWindowIdFromWindow(HWND);
                return AppWindow.GetFromWindowId(WINID);
            }
            return null;
        }

        private Window m_window;
        private AppWindow _AppWindow;
    }
}
```

Now that we have declared our AppWindow variable we can begin setting some properties.

- Change the size (page 10)
- Extend content into the title bar (page 11)

Change the Size:

```
namespace 'solution name'
{
	/// <summary>
	/// Provides application-specific behavior to supplement the default Application class.
	/// </summary>
	public partial class App : Application
	{
		/// <summary>
		/// Initializes the singleton application object.  This is the first line of authored code
		/// executed, and as such is the logical equivalent of main() or WinMain().
		/// </summary>
		public App()
		{
			this.InitializeComponent();
		}

		/// <summary>
		/// Invoked when the application is launched.
		/// </summary>
		/// <param name="args">Details about the launch request and process.</param>
		protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
		{
			m_window = new MainWindow();
			_AppWindow = GetAppWin();
			int width_size = 800;
			int height_size = 600;
			_AppWindow.Resize(new Windows.Graphics.SizeInt32(width_size, height_size));
			m_window.Activate();
		}

		private AppWindow GetWinApp()
		{
			IntPtr HWND = WindowNative.GetWindowHandle(m_window);
			if (HWND != IntPtr.Zero)
			{
				WindowId WINID = Win32Interop.GetWindowIdFromWindow(HWND);
				return AppWindow.GetFromWindowId(WINID);
			}
			return null;
		}

		private Window m_window;
		private AppWindow _AppWindow;
	}
}
```

Extend content into the title bar:

```
namespace 'solution name'
{
	/// <summary>
	/// Provides application-specific behavior to supplement the default Application class.
	/// </summary>
	public partial class App : Application
	{
		/// <summary>
		/// Initializes the singleton application object.  This is the first line of authored code
		/// executed, and as such is the logical equivalent of main() or WinMain().
		/// </summary>
		public App()
		{
			this.InitializeComponent();
		}

		/// <summary>
		/// Invoked when the application is launched.
		/// </summary>
		/// <param name="args">Details about the launch request and process.</param>
		protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
		{
			m_window = new MainWindow();
			_AppWindow = GetAppWin();
			int width_size = 800;
			int height_size = 600;
			_AppWindow.Resize(new Windows.Graphics.SizeInt32(width_size, height_size));
			_AppWindow.TitleBar.ExtendsContentIntoTitleBar = true;
			m_window.Activate();
		}

		private AppWindow GetWinApp()
		{
			IntPtr HWND = WindowNative.GetWindowHandle(m_window);
			if (HWND != IntPtr.Zero)
			{
				WindowId WINID = Win32Interop.GetWindowIdFromWindow(HWND);
				return AppWindow.GetFromWindowId(WINID);
			}
			return null;
		}

		private Window m_window;
		private AppWindow _AppWindow;
	}
}
```

Here is some WinUi 3 link that might help you get started:

https://learn.microsoft.com/it-it/windows/apps/winui/winui3/
https://learn.microsoft.com/it-it/windows/apps/winui/winui3/create-your-first-winui3-app

Hope i helped all of you, happy coding!!