DockPanel Suite

Version 1.0

Abstract

DockPanel suite is designed to achieve docking capability for MDI forms. It can be used to develop applications with Visual Studio .Net style user interface.

DockPanel suite is a 100%-native .Net windows forms control, written in C#

Features of *DockPanel* suite:

- Complex hierarchies
- Auto-hide
- Free drag-and-drop
- Nested docking
- Integration with .Net Framework and VS .Net
- Persistent to XML file

INTRODUCTION

Terms and Conditions

DockPanel suite is provided as free software with source code provided. You can freely use it in your applications (commercial or non-commercial). Weifen Luo, The author of the control, owns the copyright of the control, and is NOT responsible for any damage in your application caused by using this control, directly or indirectly.
Weifen Luo will feel honored if his name appears somewhere in your applications using this control (about box, documentation, etc). He also welcomes all kinds of feedback (good or bad). Please feel free to contact him at: weifenluo@yahoo.com.
The control will be updated at:
http://sourceforge.net/projects/dockpanelsuite/ (from here, you can get release notification, post messages in the forum, make donation or participate the project, etc)

Features

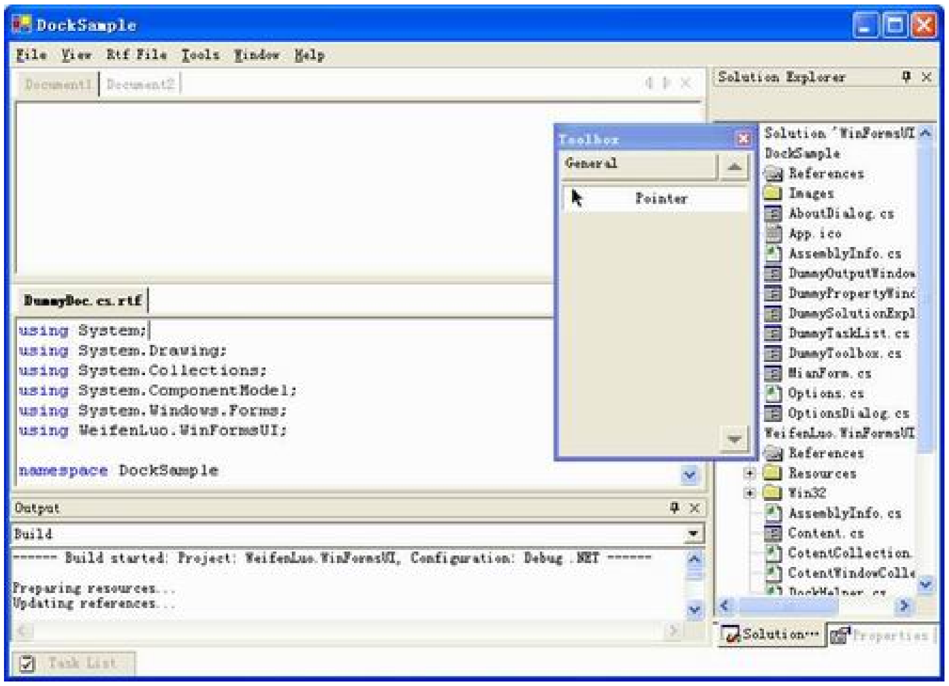
DockPanel suite is designed to achieve docking capability for MDI forms. It can be used to develop Visual Studio .Net style applications.

Features of *DockPanel suite*:

- Complex hierarchies and nested docking.
- Auto-hide.
- Free drag-and-drop
- Integration with .Net Framework and VS .Net
- Persistent to XML file.

Complex Hierarchies and nested docking

DockPanel suite mimics the user-interface style of drag and drop, dockable windows found within the Visual Studio .NET IDE. The screen shots shows off the features:

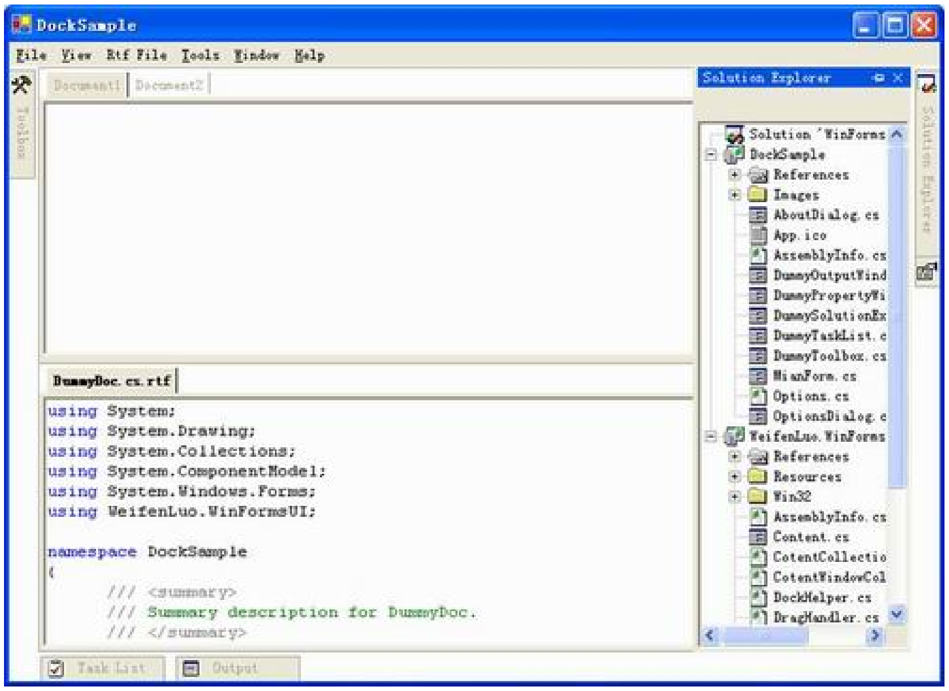


```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using WeifenLuo.WinFormsUI;

namespace DockSample
```

```
------ Build started: Project: WeifenLuo.WinFormsUI, Configuration: Debug .NET ------

Preparing resources...
Updating references...
```

Auto-Hide

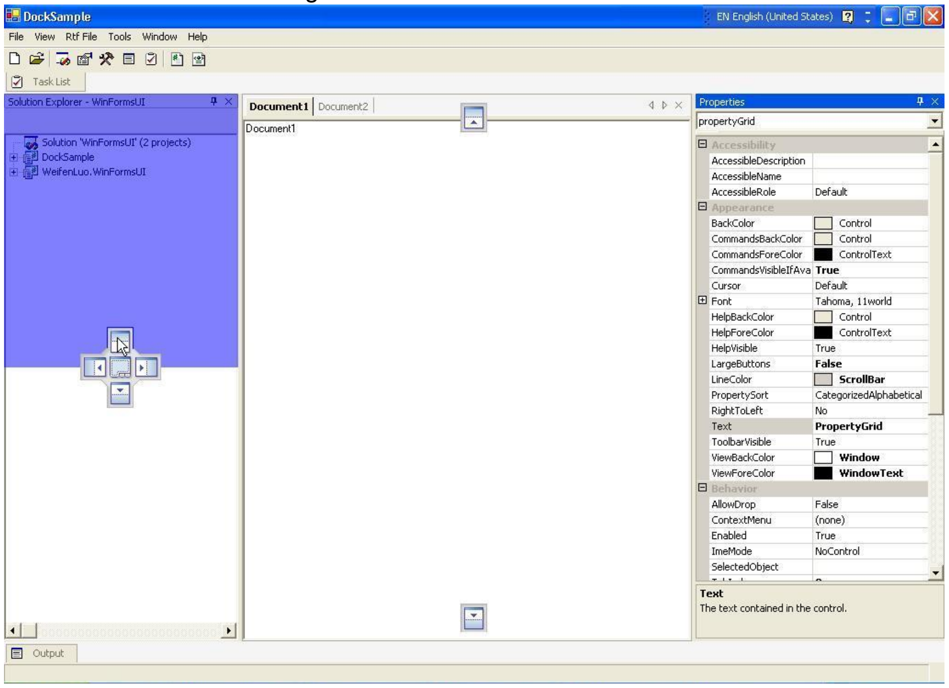
DockPanel has an auto-hide feature that enables valuable screen real estate to be preserved. In the screenshot below, all the tool windows are in auto-hide mode and the *Solution Explorer* tool window is currently being displayed:



```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using WeifenLuo.WinFormsUI;

namespace DockSample
{
    /// <summary>
    /// Summary description for DummyDoc.
    /// </summary>
```

Free Drag and Drop

At run time, the end-users can freely drag and drop windows to re-arrange to their preferred layout. The developer can limit this by simply setting some properties in Visual Studio IDE form designer.


DockSample
EN English (United States)
File View Rtf File Tools Window Help
Task List
Solution Explorer - WinFormsUI
Solution 'WinFormsUI' (2 projects)
DockSample
WeifenLuo.WinFormsUI
Document1
Document2
Document1
Properties
propertyGrid
Output

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
CommandsBackColor	Control
CommandsForeColor	ControlText
CommandsVisibleIfAva	**True**
Cursor	Default
Font	Tahoma, 11world
HelpBackColor	Control
HelpForeColor	ControlText
HelpVisible	True
LargeButtons	**False**
LineColor	**ScrollBar**
PropertySort	CategorizedAlphabetical
RightToLeft	No
Text	**PropertyGrid**
ToolbarVisible	True
ViewBackColor	**Window**
ViewForeColor	**WindowText**
Behavior	
AllowDrop	False
ContextMenu	(none)
Enabled	True
ImeMode	NoControl
SelectedObject	

Text
The text contained in the control.

Integration with .Net Framework and VS .Net

Working gracefully with other components is a very important factor all through the design of *DockPanel suite*.

At run time, the document window will be treated as a MDI form, so it gets all the MDI form's features such as menu merge.

At design time, the control is fully supported by the Visual Studio .Net IDE. Setting the properties in the form designer can have most things done. Very few codes needed.

Persistent to XML File

DockPanel suite can be persisted to XML file. By using this feature, the application can restore its screen layout every time it starts up. This is well demonstrated in the *DockSample* application.

DockPanel Control

The first level of window in DockPanel Suite is the DockPanel control. Normally it takes up the space of the application workspace:

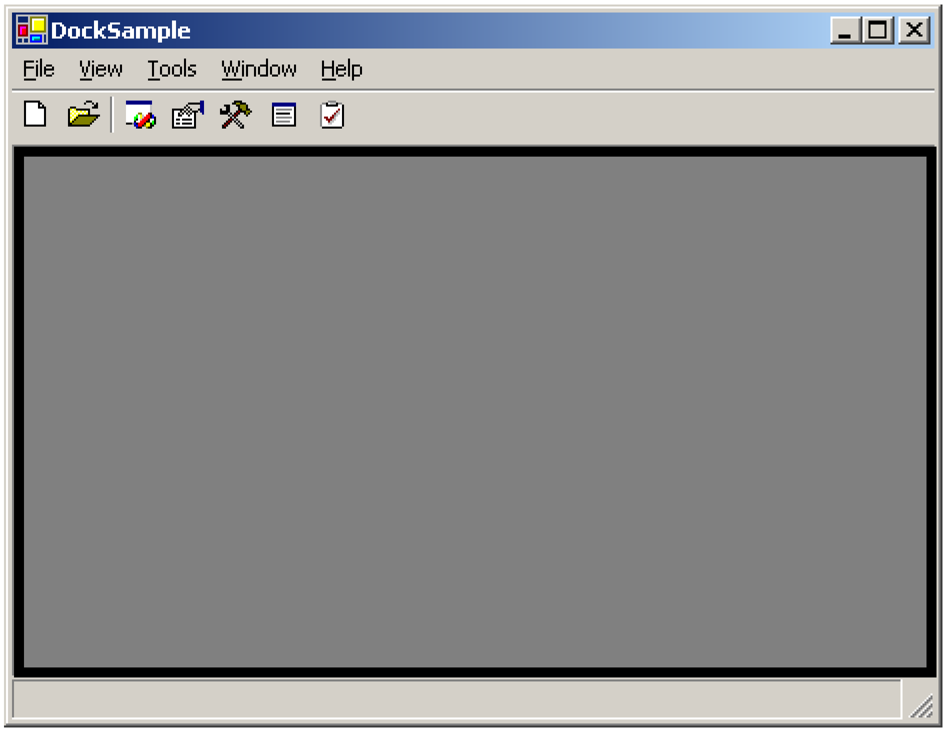


Figure 1: Application main form with an empty DockPanel

Auto Hide Windows

If there is any auto hide window, DockPanel will display a tab strip on the out-most edge. For example, the auto-hide window “Solution Explorer” is docked to the right side of DockPanel (figure 2, 3 and 4), the tab strip is displayed on the right edge of the DockPanel.

Auto hide window has 3 statuses: hide, mouse-hover-display and click-display.

In hide status, only the tab is displayed. The window itself is completely hidden. That can preserve the valuable screen to display more information. In Figure 1, the window “Solution Explorer” is in hide status so that the window “Document1” can take more space to display more information:

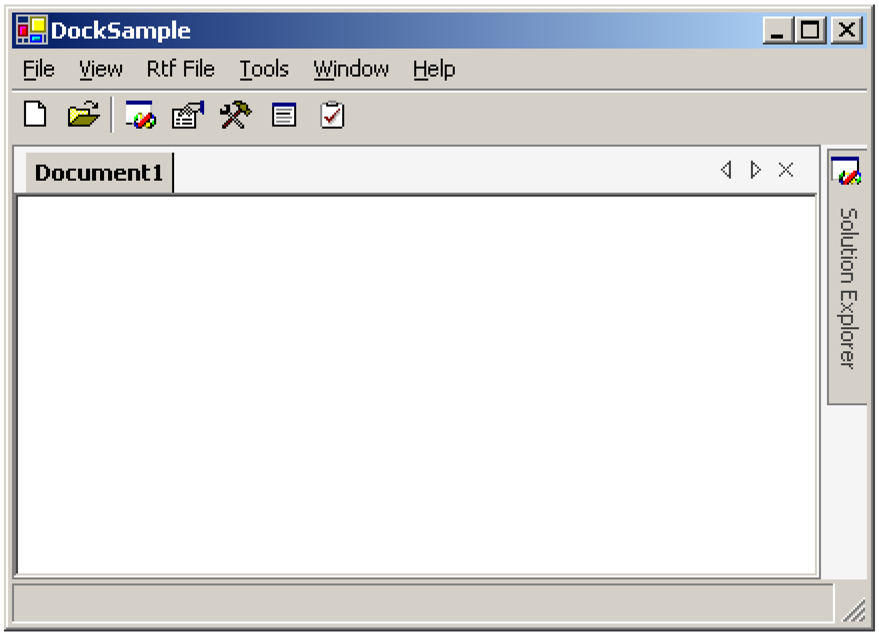


Figure 2: Auto hide window “Solution Explore” in hide status

The auto hide window can be displayed in mouse-hover-display status. That is, by hovering the mouse over the tab, the window will be displayed. You can keep the window displayed by staying the mouse on the tab or on the window itself. If you want just to have a look at the information displayed in the window, this is the best choice. In Figure 2, the window “Solution Explorer” is displayed in mouse-hover-display status. Please notice the caption of the window is inactive.

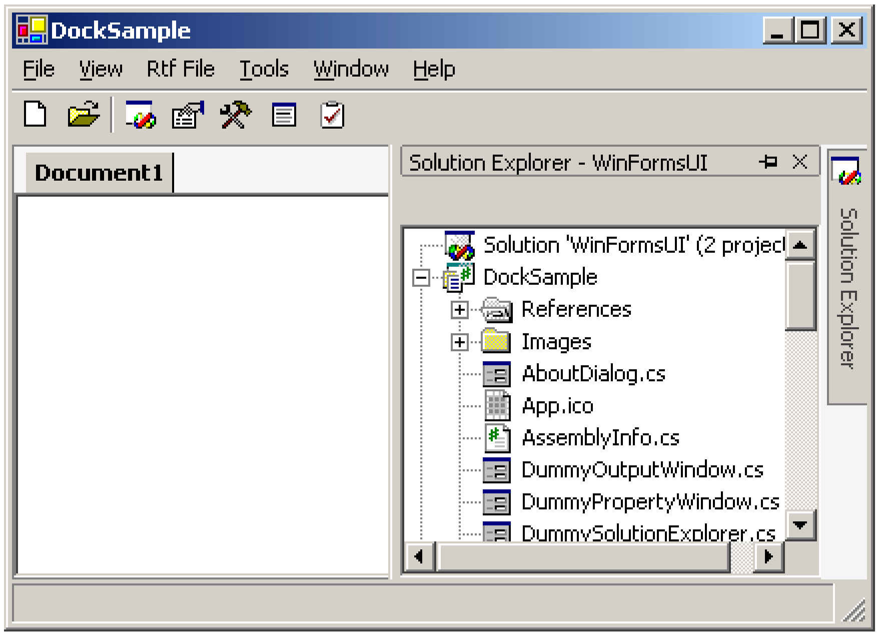


Figure 3: Auto hide window “Solution Explorer” in mouse-hover-display status

If you click the tab to display an auto hide window, it will be displayed in “click-display” status: the window will be displayed with active caption, and it gets the keyboard input focus. You can change the status to “mouse-hover-display” by changing the input focus to another window (clicking on the window “Document1”, for example), or double click the caption of the auto-hide window to hide it.

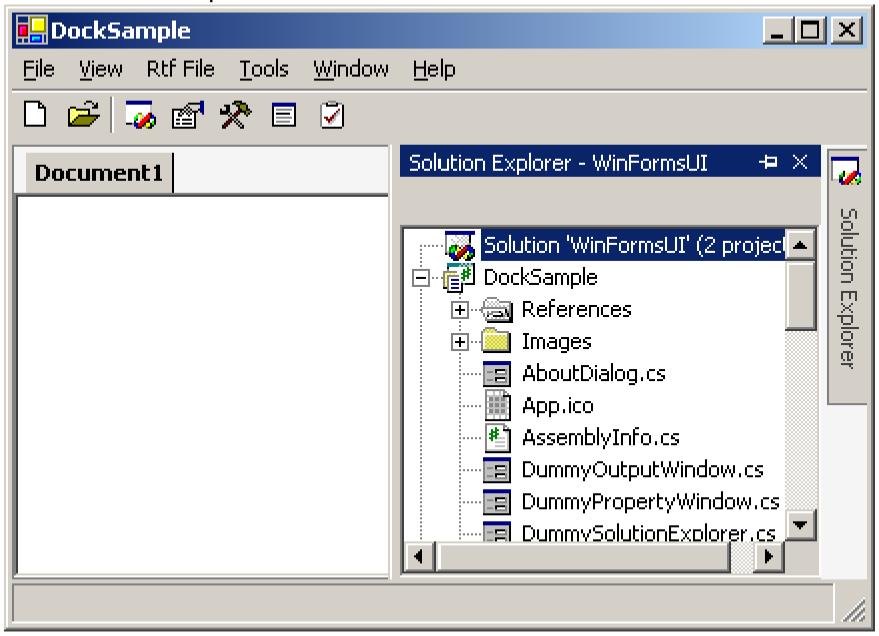
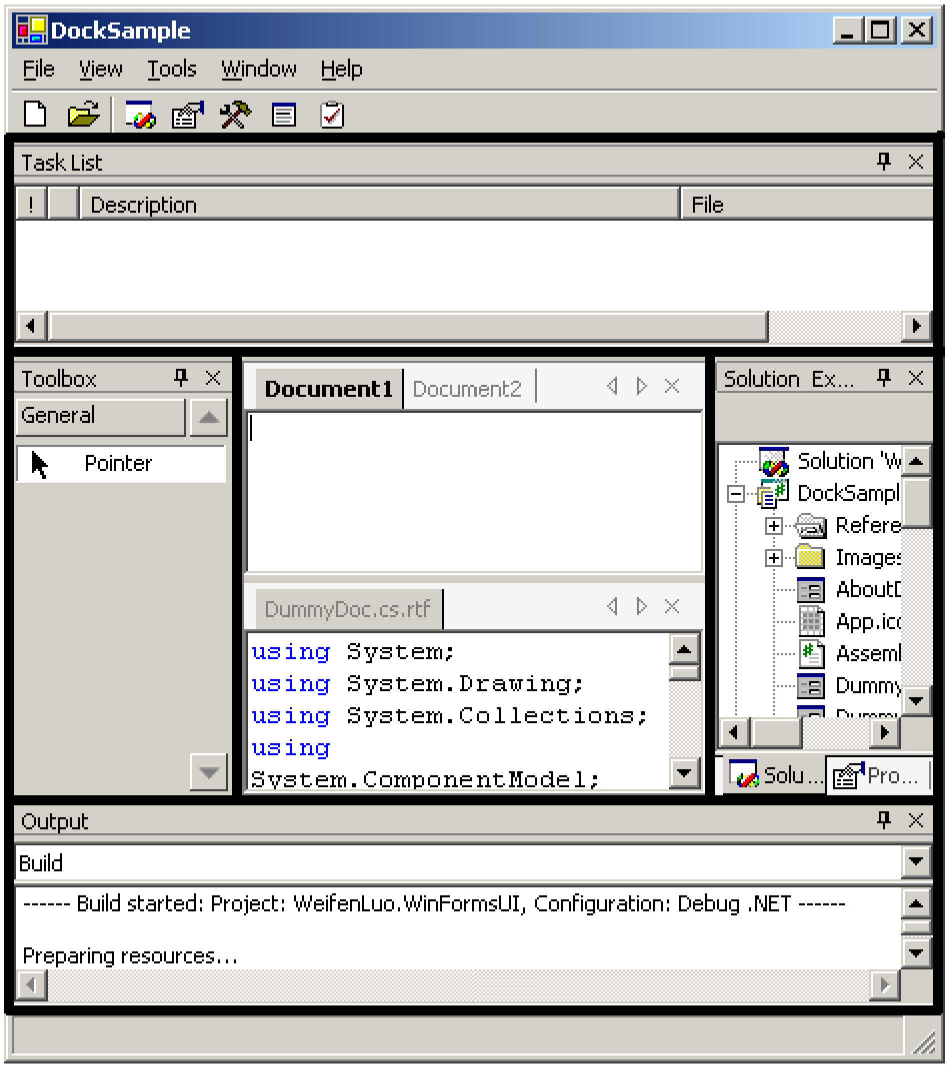


Figure 4: Auto hide window “Solution Explorer” in click-display status

Dock Windows

The DockPanel is divided into 5 dock windows: Top, Bottom, Left, Right and Document. In the following screen shot Figure 5, the “Task List” tool window is displayed in Top dock window; the “Output” tool window is in Bottom dock window; The “Toolbox” tool window is in Left dock window; the “Solution Explorer” and “Property” tool windows are in right dock window; the “Document1”, “Document2” and “DummyDoc.cs.rtf” are in document window.

The document window is always positioned in the central of DockPanel. Other 4 dock windows’ positions are determined by the Z-order. The last displayed dock window will be sent to the back of the Z-order. In the following screen shot for example, the “Task List” tool window (Top dock window) is displayed after the “Toolbox” tool window (Left dock window) and “Solution Explorer”/”Property” tool windows (Right dock window), and it takes all the width.



```
using System;
using System.Drawing;
using System.Collections;
using
System.ComponentModel;
```

```
------ Build started: Project: WeifenLuo.WinFormsUI, Configuration: Debug .NET ------

Preparing resources...
```

Figure 5: 5 Dock windows in DockPanel

Float Windows

Everything displayed in a dock window can be displayed in a float window as well. The float window is always on top of the main form, which the DockPanel resides. The following screen shot shows the “Solution Explorer” tool window and “Property” tool window displayed in a float window:

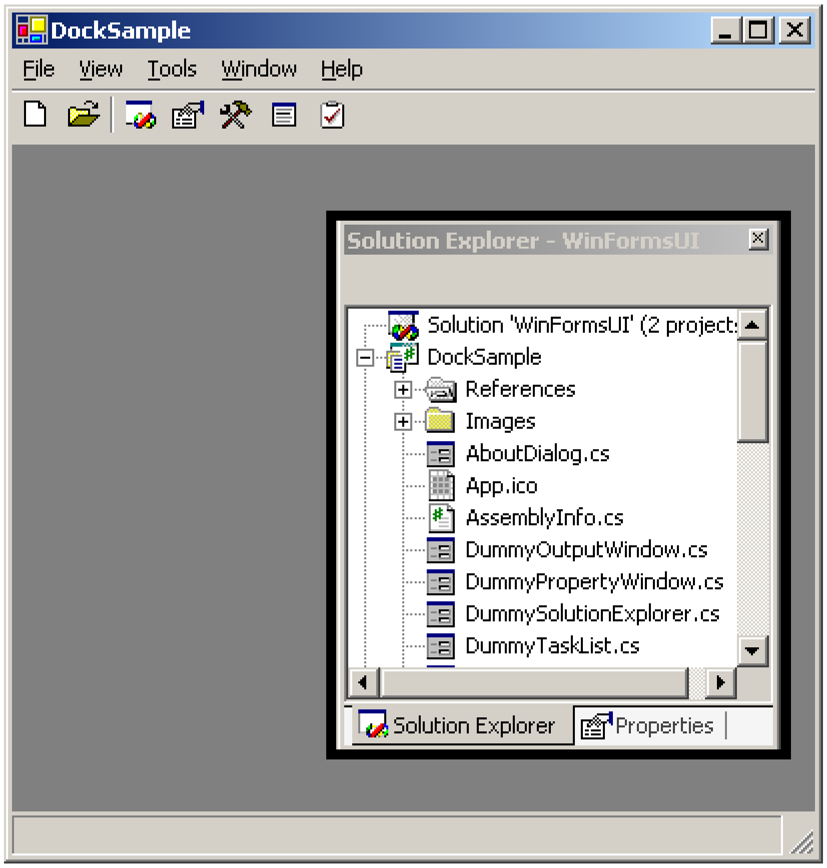


Figure 6: Sample of float window

Dock Content and Dock Pane

Dock content is the window actually implementing the application's user interface. It must be a form implements the IDockContent interface. For example, the dock content "Solution Explorer" has a TreeView control to display the folders and files of the solution. Dock content window is ALWAYS displayed inside a dock pane unless it is detached from the DockPanel. The dock pane is displayed inside either a dock window or float window.

There are 2 types of dock panes, document type and tool window type. When a dock pane is displayed inside the document dock window (document type), it displays a tab strip on the top for the dock content(s) it contains (See Figure 7). Otherwise the dock pane (tool window type) will display a caption on the top, and a tab strip on the bottom if it contains multiple dock contents (Figure 8).

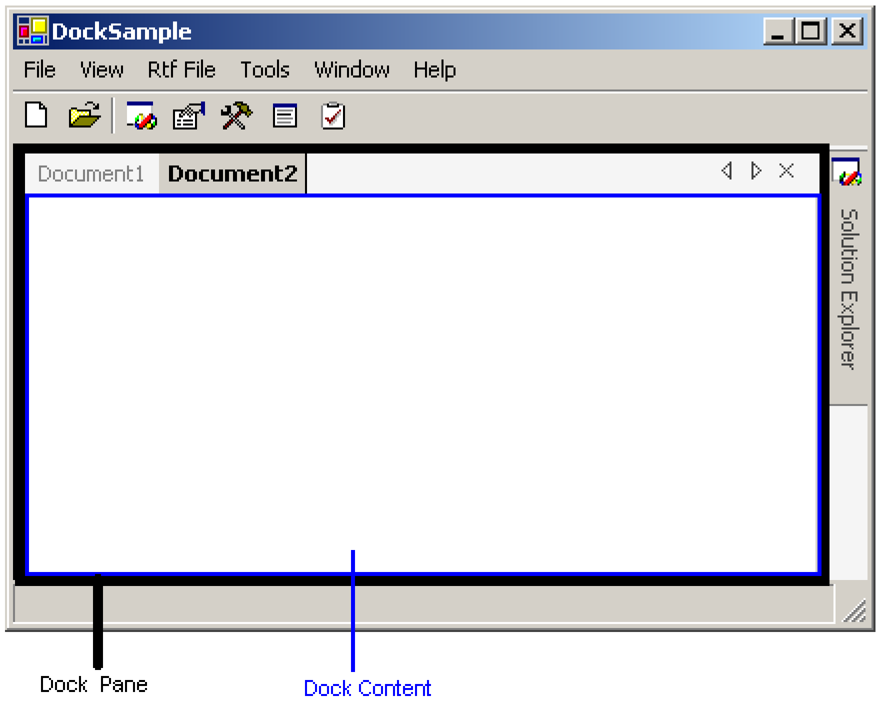


Figure 7: Document type dock pane and dock contents.

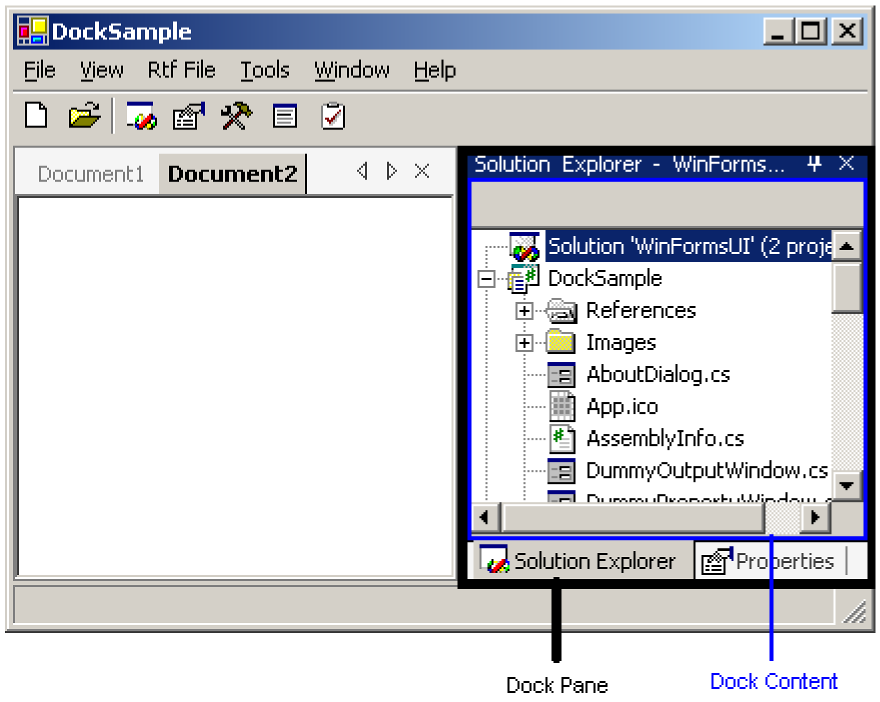


Figure 8: Tool window type dock pane and dock contents.

You can arrange dock content into dock pane by drag and drop the tab of the dock content. Keep in mind that dock content is always displayed inside a dock pane. In Figure 8, if you drag-and-drop the dock content “Solution Explorer” into other dock window or float window, there will be two dock panes: one contains dock content "Solution Explorer", and another one contains dock content "Properties".

Nested Dock Panes

Multiple dock panes can be displayed in the same dock window or float window in a nested way. In Figure 9, the dock pane, which contains the dock content "Toolbox", can be docked to the left side of the dock pane contains the dock contents "Solution Explorer" and "Properties". After that, another dock pane can be docked to any edge of the two existing dock panes.

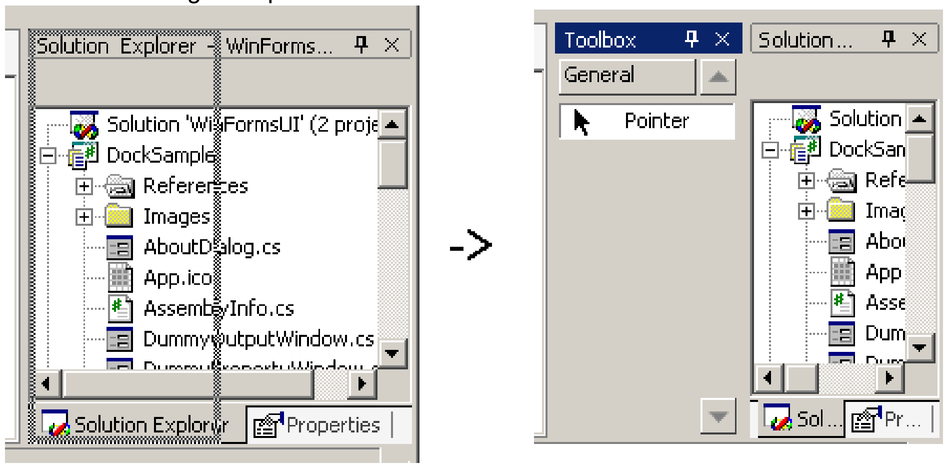


Figure 9: The dock pane nested docking

Note: nested docking is occurred on the dock pane level, not the dock content level.

The control library provides 4 major classes/interfaces for programming dockable windows: *DockContent*, *DockPane, IDockListContainer* and *DockPanel*.

Implementing *IDockContent*

Each *IDockContent* instance represents a single dockable unit within the docking window framework. To create *IDockContent* in your application, implements the *IDockContent* interface for your form. You can also derive your form from *DockContent* class: simply add a form to your project, and change this form's base class to *DockContent* class. *DockContent* class is derived from *System.Windows.Forms.Form* class, plus some docking properties, methods and events. You can easily design you *DockContent* in the Visual Studio .Net IDE.

The DockState property gets/sets the docking state of the DockContent.

The *AutoHidePortion* property determines the size of the window when the *DockContent* instance is displayed in auto-hide mode. For example, giving *DockPane* instance is docked to the left edge of *DockPanel* instance in auto-hide mode, the width of the *DockPanel* instance is 500 pixels, and the value of the *Content.AutoHidePortion* is 0.25, the auto-hide window will be displayed with width 500 * 0.25 = 125 pixels.

The *DockableAreas* property limits the states of this content allowed to show.

The *ShowHint* property defines the default docking state.

The overrided *Show* method displays the content in default or specified docking state.

Programming *DockPane* Class and *IDockListContainer* Interface

DockPane class works as container of *DockContent* class. It displays a collection of *DockContent* with a tab strip and a caption (tool window only). The *DockPane.Contents* property and *DockContent.Pane* property specify the parent-child relationship between *DockPane* instance and *DockContent* instance. The *ActiveContent* property determines which *DockContent* is currently being displayed in the *DockContent* instance.

Each instance of *DockPane* has a *DockState* property to determine the docking state. A collection of *DockPane* objects are grouped in a *DockList* instance and contained in *IDockListContainer*. *DockWindow* class and *FloatWinow* class implements the *IDockListContainer* interface. The method *AddToDockList* performs nested docking for dock panes.

Two *DockPane* instances have the same docking states when: 1. Both of the instances' *DockState* property is *DockState.Float* and their *FloatWindow* properties are identical; 2. Both of the instances' *DockState* property is NOT *DockState.Float* and their *DockPanel* properties are identical.

Mostly *DockPane* instance is created and manipulated from *DockContent.Show* method and drag-and-drop processing code. Developers don't need to create or manipulate it unless they want to precisely control the docking layout programmatically.

Programming *DockPanel* Class

DockPanel is the central of the control library. Visually it resides in the application's main form and contains all the *DockWindow* and *FloatWindow* instances. It can be designed using Visual Studio .Net form designer.

The *DockLeftPortion*, *DockRightPortion*, *DockTopPortion*, *DockBottomPortion* properties determines the size of the docking windows.

The *MdiIntegration* property determines if all document windows should be treated as MDI forms. If it's true, the menu will be merged automatically.

The *SaveAsXml* and *LoadFromXml* methods can save/load all the docking windows/states to/from a XML file.