How Do I...? Examples

1. Adding a ScatterViewItem Control Programmatically

   A ScatterView control is a container for other controls or objects of any type. It provides the framework for moving and manipulating objects on the Microsoft Surface screen. You can add items to the ScatterView control by using XAML markup or programmatically such as in response to a particular event. This article shows how you can programmatically add items to a ScatterView control.

2. Adding Video to a ScatterViewItem Control

   A ScatterViewItem control can accept content of any type. This article describes how you can create a ScatterViewItem control with video content.

3. Capturing and Rendering a Raw Image

   This example shows how to capture and render a raw image using XNA.
   Note: You must have a Microsoft Surface unit to run the code that is used in this article. You cannot run this code by using a separate workstation and Surface Simulator.

4. Capturing Raw Images by Using WPF

   This article describes how to capture and persist a raw image from the Microsoft Surface screen by using the Presentation layer of the Microsoft Surface SDK and elements of the Core layer and the System.Drawing assembly. The Microsoft Windows Presentation Foundation (WPF) enables you to create and use custom graphics, and the System.Drawing assembly includes methods for handling bitmap images.

   The sample application in this topic creates a SurfaceWindow control that contains a single SurfaceButton control. When you place an item on the Microsoft Surface screen and press the button, an image of the item is captured and saved. The Microsoft Surface screen does not indicate that the image has been processed, but the button glows when you press it to indicate that the application responded.

   The captured image is saved to a local Temp.bmp file, and the sample application is immediately ready to capture another image. The file name of the captured image is always the same (Temp.bmp), so previously saved images are overwritten every time that you press the button.

   Note: You must have a Microsoft Surface unit to run the code that is used in this article. You cannot run this code by using a separate workstation and Surface Simulator.

5. Combining a User Control with a ScatterViewItem Control

   When you host a user control in a ScatterViewItem control, your control gains all the manipulation, inertia, and scaling advantages of the ScatterViewItem. When a user performs some task with your control (such as touching a button), you might want to close the control and (optionally) the hosting ScatterViewItem control. This topic describes how to perform these tasks.

6. Creating a Continuously Panning List
When you use a \textit{SurfaceScrollViewer} control to view content that is larger than its viewport area, users can scroll the content by direct contact with the content itself, instead of using a scroll bar. This ability is called panning.

Typically, as users pan content and it reaches its beginning or end, panning stops. This topic describes how you can create a SurfaceScrollViewer control that is based on a custom implementation of the \textit{ISurfaceScrollInfo} interface. The custom interface enables users to pan content in a continuous loop. The concepts and code in this article are based on portions of the \textit{Shopping Cart} sample application.

7. \textbf{Creating a Custom Shape for a ScatterViewItem Control}

By default, a \textit{ScatterViewItem} control is rectangular in shape. This topic describes how you can create a ScatterViewItem control that uses a custom shape.

8. \textbf{Creating a Tag Visualization}

A \textit{TagVisualization} object is the user interface that appears on the Microsoft Surface screen when a user places a \textit{tagged object} on the screen. Tags might specify different values so that your application can recognize different objects and display the appropriate user interface. For example, you might have five digital camera models, each with a tag that is coded to a different value. When you place a camera on the Microsoft Surface screen, the user interface that appears describes the features of that model and enables users to activate other logic such as a side-by-side comparison of features. The user interface that appears directly, and related user interface that appears in response to user selection, can perform whatever logic that your application needs. For example, if the camera has a wireless communication channel, you could download and display the images it contains. For more information about tag visualizations, see \textit{Using Tag Visualizations}.

There are three main classes that your application uses when it creates a visualization:

- \textit{TagVisualization}. A TagVisualization object represents the content that appears to users when a tagged object is placed on the Microsoft Surface screen. This content (the user interface) takes whatever form is appropriate.
- \textit{TagVisualizationDefinition}. A TagVisualizationDefinition object defines the details that are used to create a TagVisualization control for a particular tag value. Among other things, this class enables you to define the position and orientation of the visualization in relation to the tagged object. By default, the visualization appears centered on the tag that is affixed to the object and oriented the same direction as the tag.
- \textit{TagVisualizer}. The TagVisualizer control reacts to one or more tagged objects that are placed on the Microsoft Surface screen by creating and displaying TagVisualization objects. A TagVisualizer control automatically tracks the motion of a tagged object and moves the TagVisualization object with the physical object.

This topic shows how you can create a tag visualization.

9. \textbf{Creating a Touch-Enabled User Control}

This example shows how to create a touch-enabled user control. This example uses manipulations to handle rotation for a dial. You could add value labels to the control, along with a get method that would make the current setting of the dial available to other parts of your application.
10. **Customizing the Disabled Effect of a LibraryBarItem**

By default, when you drag an item from a **LibraryBar** control, a copy of the item remains inside the LibraryBar control with its IsItemDataEnabled property set to false. By default, when an item is disabled, it appears dimmed. This topic describes how you can create a custom visual effect for disabled items in a LibraryBar control.

11. **Data Binding a ScatterView Control**

As with other Microsoft Windows Presentation Foundation (WPF) controls, a **ScatterView** control supports data binding. This topic describes how you can create a ScatterView control that is bound to a directory of images. The images then appear as individual ScatterViewItem controls.

12. **Grouping Data of a LibraryBar Control**

When you use a **LibraryBar** control to display items, you can arrange the items in groups. This topic describes how you can create LibraryBar control, populate it with sample data, and arrange the data in groups.

13. **Implementing a Custom Removal Behavior for Tag Visualizations**

A **TagVisualization** object represents the user interface that appears on the Microsoft Surface screen when a user places a tagged object on it. When you remove the tagged object from the screen, the **TagRemovedBehavior** property determines what happens to the visualization. By default, the visualization fades from the screen. This topic describes how you can implement a custom removal behavior for a TagVisualization object. The custom behavior consists of an animated bitmap effect that is applied to the TagVisualization object.

14. **Manipulating a Large Image by Using XNA**

A **TagVisualization** object represents the user interface that appears on the Microsoft Surface screen when a user places a tagged object on it. When you remove the tagged object from the screen, the **TagRemovedBehavior** property determines what happens to the visualization. By default, the visualization fades from the screen. This topic describes how you can implement a custom removal behavior for a TagVisualization object. The custom behavior consists of an animated bitmap effect that is applied to the TagVisualization object.

15. **Manipulating Shapes**

This example shows how to create user interface elements (such as shapes, pictures, and controls) that users can manipulate by touching the Microsoft Surface screen. The **ScatterView** control, which is available in the Presentation layer of the Microsoft Surface SDK, can help you implement common manipulations, such as rotating a shape around a pivot, moving a shape across the screen by using one or more fingers, and enlarging or reducing a shape by using two or more fingers.
16. Moving UI Elements with Touch

This example shows how to create a user interface (UI) element and drag it by using a single touch.

Note: The ScatterView control provides a framework for moving and manipulating objects on the Microsoft Surface screen. This example demonstrates a lower-level look at how to move a UI element.

17. Overriding Control Styles

When you use a Microsoft Surface control in your application, the control has a certain default appearance. However, you can use Microsoft Windows Presentation Foundation (WPF) styles and templates to alter the appearance of Microsoft Surface controls. This topic describes how to use built-in, custom styling for a SurfaceButton control and how to create your own custom styling that you can apply to a SurfaceButton control or other Microsoft Surface controls.

18. Reducing CPU Usage by Using ApplicationLauncher Events

When a Microsoft Surface application is deactivated (that is, it loses focus), its process thread continues running in the background. To maximize the operating efficiency of a Microsoft Surface unit, an application should stop capturing raw images and avoid audio playback, video processing, or other CPU-intensive processing that might use resources that should be available to the current foreground application. The Microsoft.Surface.ApplicationLauncher class provides event handlers that can notify an application when it is being deactivated. With this event information, you can make sure that an application knows when to avoid using CPU intensive code segments.

The Microsoft Surface application templates for Microsoft Visual C# 2008 (and Microsoft Visual Studio 2008) include ApplicationLauncher event handler registrations (ApplicationActivated, ApplicationPreviewed, and ApplicationDeactivated) and methods or logic to economize CPU usage.

The Microsoft Surface XNA template includes all ApplicationLauncher event handler registrations and the relative event handler logic to prevent a deactivated process from gaining access to the Update method. This article refers to (but does not further describe) the Microsoft Surface XNA template.

19. Testing Applications by Using the Surface Simulator Automation API

The Surface Simulator Automation API (Microsoft.Surface.Simulator.Automation) enables you to programmatically simulate contact input for Microsoft Surface. The Surface Simulator Automation API enables you to directly manipulate Microsoft Surface applications that you are testing by using Microsoft.Surface.Simulator.Automation.SimulatedContact objects. This example describes how to set up an automated test of an application that is running in Surface Simulator by using a Windows Forms project.