Microsoft Surface
Software Development Kit

Microsoft Surface Overview

Microsoft Surface is the first commercially available surface computing platform from Microsoft Corporation. It turns an ordinary tabletop into a vibrant, interactive surface. The product provides effortless interaction with digital content through natural gestures, touch and physical objects. In essence, it's a surface that comes to life for exploring, learning, sharing, creating, buying and much more. In restaurants, hotels, retail and public entertainment venues, this experience will transform the way people shop, dine, entertain and live.

Microsoft Surface is a 30-inch display in a table-like form factor that's easy for individuals or small groups to interact with in a way that feels familiar, just like in the real world. Microsoft Surface can simultaneously recognize dozens and dozens of movements such as touch and gestures and can recognize unique objects that have identification tags (similar to bar codes).

Microsoft also provides a portfolio of basic Microsoft Surface applications, including Concierge, Photos, and Music. You can customize these applications to provide customers with unique experiences.

Surface computing breaks down traditional barriers between people and technology, changing the way people interact with all kinds of everyday content, from photos to maps to menus. The intuitive user interface works without a traditional mouse or keyboard, allowing people to interact with content and information by using their hands and natural movements. Users are able to access information either on their own or collaboratively with their friends and families, unlike any experience available today. Surface computing features four key attributes:

- **Direct interaction**: Users can actually "grab" digital information with their hands and interact with content through touch and gesture, without the use of a mouse or keyboard.
- **Multitouch contact**: Surface computing recognizes many points of contact simultaneously, not just from one finger as with a typical touch screen, but up to dozens of items at once.
- **Multiuser experience**: The horizontal form factor makes it easy for several people to gather around surface computers together, providing a collaborative, face-to-face computing experience.
- **Object recognition**: Users can place physical objects on the surface to trigger different types of digital responses.

All it takes is a simple touch.

What's more, Microsoft Surface responds to multiple touches at one time (*multitouch*).

All of this functionality enables a new range of social, multiuser applications.

With Microsoft Surface, you can develop applications that:

- Respond to touch and gestures.
- Respond to multiple touches so that several people can work and play at the same time.
- Respond to object blobs. For example, users can use a paintbrush in a drawing application.
- Recognize objects and initiate an action based on tags (tagged objects). For example, users can use a domino-like tag (with small white circles on a black background) on an object start an action, draw a shape, and so on.
- Have 360-degree user interfaces. For example, four people can sit around the Microsoft Surface unit and have the content appear correctly for everybody.

To learn more about the Microsoft Surface system and what users encounter when they interact with a Microsoft Surface unit, see the following sections:

- Microsoft Surface System
- Microsoft Surface User Experience

Microsoft Surface
Software Development Kit

Microsoft Surface Platform

The Microsoft Surface platform takes advantage of advances in the Windows Vista operating system. The Microsoft Surface platform adds touch-enabled controls to the Microsoft Windows Presentation Foundation (WPF) technology. These controls extend WPF controls and enable a whole new world of user interface (UI). The Extensible Application Markup Language (XAML) provides a declarative markup syntax for creating WPF elements. Instead of WPF, developers can also use the Microsoft XNA development platform to develop Microsoft Surface applications. To learn more about how to use WPF or XNA to create Microsoft Surface applications, see Presentation and Core Layers.

Developers can create Microsoft Surface applications by using Microsoft Visual C# 2008 Express Edition or Microsoft Visual Studio 2008. Visual C# 2008 and Visual Studio 2008 provide the same tools for Microsoft Surface that developers can use to create applications with the Microsoft .NET Framework (such as powerful editors, project management, debugging, and so on).

> **Note:** Visual C# 2008 is included on developer Microsoft Surface units, but you can also install Visual Studio 2008 on a Microsoft Surface unit. Or, you can develop Microsoft Surface applications on a separate computer that includes Visual C# 2008 or Visual Studio 2008.

Designers can easily leverage the end-to-end capabilities of Microsoft Expression Studio 2 to design, build, and deliver Microsoft Surface applications. Expression Studio 2 includes all the capabilities that designers need to create graphics and media assets and design interactivity in Microsoft Surface applications. Microsoft Expression Blend 2 can generate XAML output, so developers who use Visual C# 2008 (or Visual Studio 2008) and designers who use Expression Blend 2 can share the same files.

For more information about the Microsoft Surface platform, see Microsoft Surface Architecture.

Hardware Recommendations

Hardware recommendations are given in reference to the Windows Experience Index.

- CPU: 4.0 or above
- RAM: 4.0 or above
- Graphics: both 5.0 or above
    - At least 256 MB with Microsoft DirectX 9.0c and Shader Model 2.0 support.
    - A monitor that is capable of 1280 × 960 screen resolution or a widescreen monitor that is capable of 1440 × 900 screen resolution.

      Important: These screen resolutions enable you to successfully run the Surface Simulator application. Otherwise, it will not run properly.
    - A graphics card that supports the Windows Aero user interface in Windows Vista.

      Important: If your graphics card does not support Windows Aero, Surface Simulator and Surface Shell will not run properly.
- Disk: 4.0 or above
- Input: Microsoft-compatible mouse required

Installation

Install the following software in the order listed:

- Windows Vista OS, Service Pack 1 (SP1) (Microsoft Surface SDK, Workstation Edition is supported by Vista Business, Enterprise, and Ultimate ONLY)
- Windows Media Player (if not present in current installation of Vista)
- Microsoft Visual Studio 2008 (.NET Framework 3.5 is included)
- Microsoft XNA Framework Redistributable 2.0
- Microsoft Expression Blend 2
- Microsoft Surface SDK 1.0 SP1, Workstation Edition
- (Optional) Microsoft XNA Game Studio 3.0 and XNA Framework Redistributable 3.0

When installed after Visual Studio, the Surface SDK registers project templates with Visual Studio. If the Surface SDK is installed before Visual Studio, you will have to register VS templates manually by doing a repair of your Surface SDK installation, or running devenv.exe from the VS command prompt.

Setting Up a Development Environment

On a separate workstation, you can install the Microsoft Surface SDK 1.0 SP1, Workstation Edition, and use the Surface Simulator application to simulate a Microsoft Surface test environment. You can then use one or more mice to simulate contacts on a Microsoft Surface unit. Surface Simulator runs the Surface Shell and the applications that you develop. However, there are differences between Surface Simulator and an actual unit. For example, cameras are not available in Surface Simulator. If you write an application that uses raw camera input, you cannot test it by using Surface Simulator.

Microsoft Surface
Software Development Kit

Surface Simulator

Microsoft Surface Simulator is a Microsoft Windows application. You can use Surface Simulator together with Microsoft Visual C# 2008 Express Edition (or Microsoft Visual Studio 2008) to test Microsoft Surface applications on a separate workstation. Surface Simulator runs the applications and enables you to use a mouse or mice to simulate finger, blob, and tagged-object contacts, like the contacts that a user uses on a Microsoft Surface unit.

> **Important:** Surface Simulator does not support the use of raw image input. If your application requires the use of raw image APIs like **TryGetRawImage** and **UpdateRawImage**, you must test it on an actual Microsoft Surface unit.

Surface Simulator replicates the user interface and behavior of a Microsoft Surface unit that is in user mode. Surface Simulator has access points, Launcher, and the loading screen. When you start an application in Surface Simulator, the application displays like it is on a Microsoft Surface unit.

You can use Surface Simulator to evaluate how an application and its user interface respond to basic input. For example, if you simulate a painting application and if you touch multiple colors, one at a time, and then add the colors to a mixing bucket, you can test the logic of the application and how well it mixes the colors by using the touch-based interface.

Surface Simulator runs with the appearance and functionality of a Microsoft Surface unit in user mode (the way that it appears to users). You can switch applications by using Launcher and the access points that display on the Launcher screen and the applications.

Surface Shell is always turned on when you use Surface Simulator, so you see the attract application and Launcher (if you have touched an access point on the attract application to start Launcher).

This section includes the following topics:

- Starting Surface Simulator
- Surface Simulator User Interface
- How to Use Surface Simulator
- Recording and Playing Interactions

Microsoft Surface
Software Development Kit

Starting Surface Simulator

Microsoft Surface Simulator uses a 1024 × 768 simulation region to display Microsoft Surface applications. In this simulation region, Surface Simulator interprets mouse inputs as types of contacts (fingers, blobs, gestures, or tagged objects).

Note: You cannot use Surface Simulator in a Remote Desktop or virtual environment.

You do not have to have Microsoft Visual C# 2008 Express Edition (or Microsoft Visual Studio 2008) running when you start Surface Simulator. However, if Visual C# 2008 (or Visual Studio 2008) is running when you start Surface Simulator, make sure that you start Surface Simulator before you start a Microsoft Surface application from Visual C# 2008 (or Visual Studio 2008).

You must run Surface Simulator at the same privilege level as Visual C# 2008 (or Visual Studio 2008). If you start Visual C# 2008 (or Visual Studio 2008) as Administrator, you must also start Surface Simulator as Administrator. If you start Visual C# 2008 (or Visual Studio 2008) as a User, you must also start Surface Simulator as a User.

To start Surface Simulator, click **Start**, click **All Programs**, click **Microsoft Surface SDK 1.0 SP1**, click **Tools**, and then click **Surface Simulator**.

You can start a Microsoft Surface application from one of the following locations:

- From Launcher in Surface Simulator (as a user accesses it on a Microsoft Surface unit). To start the application from Launcher, the application must be registered with the unit.
- From outside Surface Simulator:
  - From Visual C# 2008 (or Visual Studio 2008), by pressing F5. This method is useful if you are debugging the application.
  - From the application executable file in Windows Explorer.
  - From the application executable file at a command prompt.

Important:

- If you start the application outside of Surface Simulator while Launcher is open, Launcher displays on top of the application. You must click an access point to open the application.
- If you start more than one application from outside Surface Simulator, you must use the Windows taskbar to switch between the applications.

Microsoft Surface
Software Development Kit

Surface Simulator User Interface

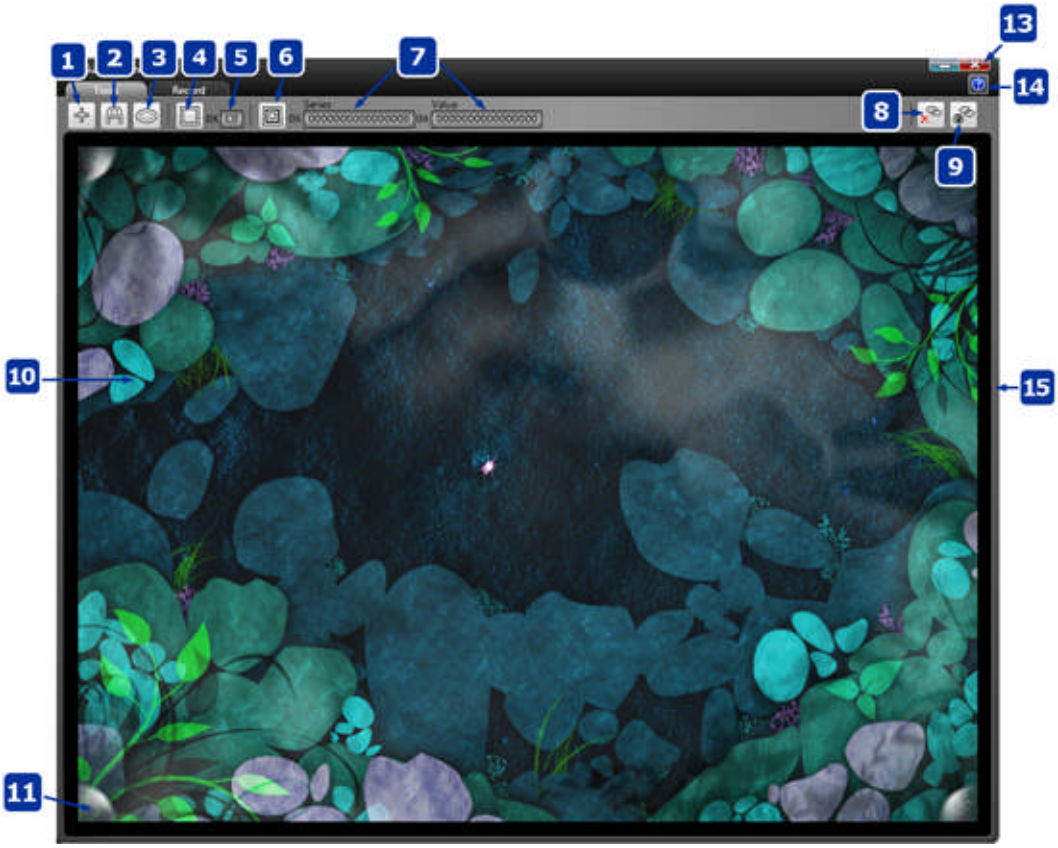
This topic describes the types of Microsoft Surface contacts you can use in Surface Simulator and describes Surface Simulator's interface.

Surface Simulator enables you to use a mouse to simulate three types of Microsoft Surface input:

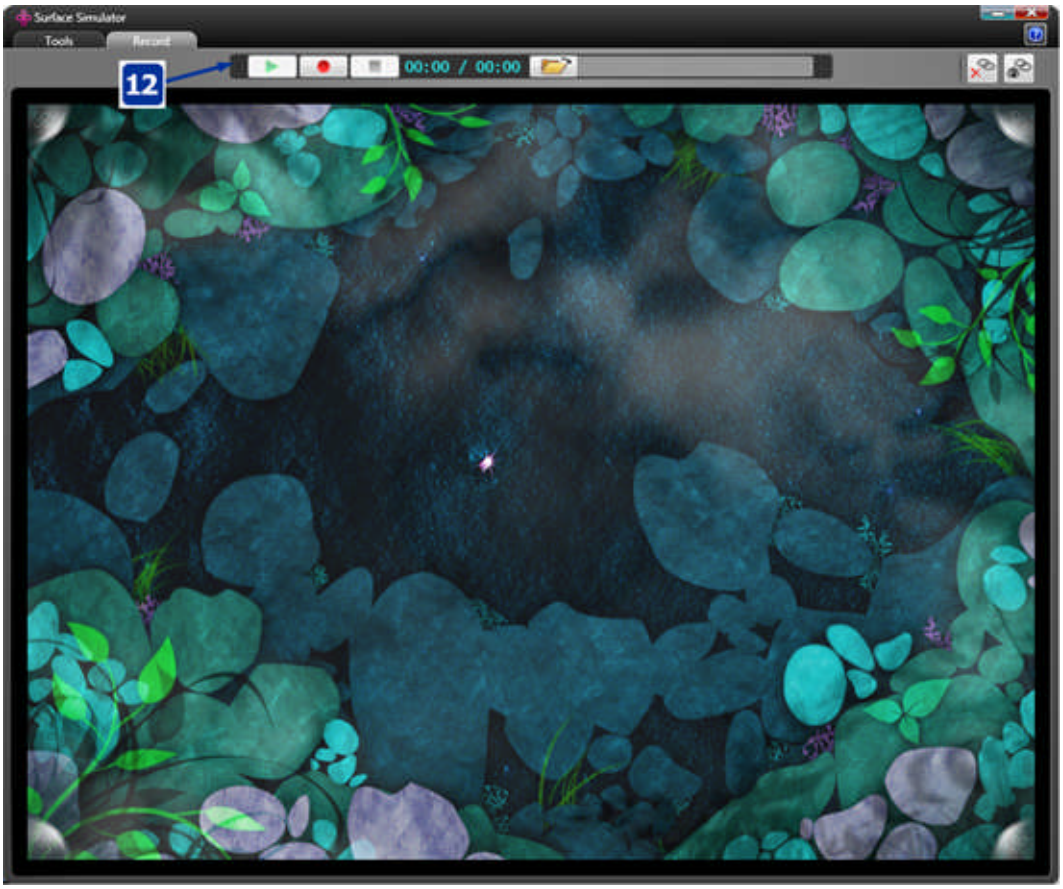
- Finger. The contact of a finger.
- Blob. The contact of an object.
- Tagged object. The contact of an object that is marked with a byte tag or identity tag.

Parts of Surface Simulator

The following illustrations show the Surface Simulator interface with an attract application.



Surface Simulator user interface with Tools tab selected



Surface Simulator user interface with Record tab selected

Surface Simulator interface includes the following pieces. (The numbers in the previous illustrations correspond to the numbers in the following list.)

1. The **Contact Selector** button enables you to select and manipulate one or more contacts. Manipulations include move, resize, stretch, and rotate.

2. The **Finger** button simulates a finger that contacts the application and triggers finger contact events.

3. The **Blob** button simulates an arbitrary ellipse shape that contacts the application and triggers blob contact events. You can resize the **Blob** button to simulate shapes of different sizes.

4. The **Tag** button simulates a tagged object that contacts the application and that uses user-specified hexadecimal values. This button triggers tagged-object contact events according to the specified hexadecimal values.

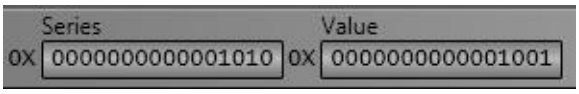
5. The **Tag Value** data entry control enables you to enter a hexadecimal value (00 through FF) to assign to tagged-object contacts that you create by using the **Tag** button.

6. The **Identity Tag** button enables you to simulate an identity tag contact that is placed on a

Microsoft Surface screen. Enter identity tag series and value properties in the adjacent **Series** and **Value** data entry controls.

7. The **Identity Tag Series** and **Identity Tag Value** controls enable you to assign hexadecimal values (0000000000000000 through 7FFFFFFFFFFFFFFF) to the series and value of an identity tag contact that you create by using the **Identity Tag** button.

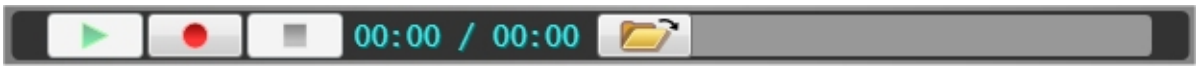


8. The **Remove All Contacts** button removes all contacts from the simulation region.

9. The **Hide/Show Contacts** button determines whether the current contacts on the simulation region are visible. You can click the button to switch from showing the contacts to hiding the contacts. You can still put new contacts on the active area of the simulated screen when existing contacts are invisible.

10. The *simulation region* is the area of the Surface Simulator user interface where the application appears. In this area, you can test the application by using finger, blob, and tagged object contacts. When you first start Surface Simulator, the active area displays the default attract application.
11. The *Surface Simulator access points* that appear in each corner of the active area are Microsoft Surface controls that enable you to move between applications and Launcher. These access points and Launcher work the same way that they do on Microsoft Surface units.

12. The *recording panel* enables you to record interactions and then play them for testing purposes. For more information about how to record and play interactions, see Recording and Playing Interactions.



13. The **Minimize** and **Exit** buttons minimizes or closes Surface Simulator and all Surface applications, respectively.

14. The **Help** button opens the Surface Simulator section of the Surface SDK documentation.

15. The Surface Simulator window is a movable window that encloses the Surface Simulator user interface and the application that you are testing. The window size is fixed to 1224 × 868, but you can minimize and restore the window.

For more information about how to use Surface Simulator and the contact types, see How to Use Surface Simulator.

Microsoft Surface
Software Development Kit

How to Use Surface Simulator

This topic describes how to work with Microsoft Surface contacts in Surface Simulator.

Adding Contacts to Surface Simulator

In the upper-left corner of the Surface Simulator window, you can use the buttons to simulate the different types of Microsoft Surface input (finger, blob, and tagged object). A *contact* is an instance of an input type that is interacting with the simulated region.

To simulate contact input, do the following:

1. Select a contact tool by clicking one of the buttons on the toolbar in the upper-left area of the Surface Simulator window (for example, the **Finger** button).
2. Move the cursor over the simulation region. The cursor displays the pointer that corresponds to the tool that you selected.
3. Press the left mouse button to add the contact and then release the mouse button to remove the contact. Or, to put a contact on the screen and leave it there, press-and-hold the left mouse button and then right-click.

You can also click the **Contact Selector** button and then select a contact on the simulated Microsoft Surface screen and drag it across the screen.

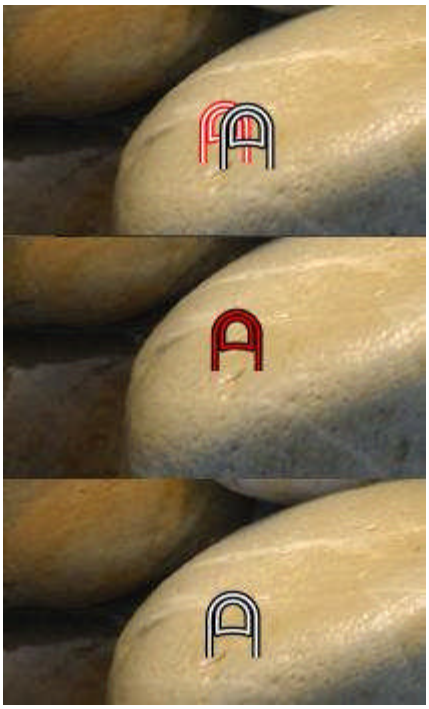
To simulate more than one contact, you can connect additional mice to your computer's USB ports.

Note: For additional USB ports, you can connect a USB hub to the computer.

Contact States

A Surface Simulator contact can appear in one of the following states:

- White with a red shadow indicates that a contact is selected for manipulation (but is still on the simulated Microsoft Surface screen).
- Red indicates that the contact is placed on the screen.
- White indicates that the contact is hovering and is not yet placed on the screen.



Using Surface Simulator Controls and Contacts

This section describes how to use Surface Simulator controls and contacts.

Contact Selector

Click the **Contact Selector** button (or press CTRL+S) to choose the **Contact Selector** tool. You can then select contacts that are already placed in the simulation region by doing one of the following:

- Place the cursor directly over an already placed contact and then click it.
- Drag a selection region around one or more contacts that have already been placed.

You can remove selected contacts by using the DELETE key.

For examples of multitouch gestures that you can simulate by using **Contact Selector**, see Common Microsoft Surface Actions in Surface Simulator.

Finger Contact

Click the **Finger** button (or press CTRL+F) and then click in the active area to simulate a finger contact in the application. You can also click the **Finger** button and then drag it in the active area to simulate a dragging gesture across the application.

Press-and-hold the left mouse button and then right-click to place a finger contact, which leaves a finger pressed in that location of the application. You might use this procedure to press multiple controls at the same time (for example, buttons or pieces of content), to press-and-hold a control, or to anchor part of an interface. For example, you might use this procedure to hold a corner of a photo and then use the mouse to rotate the photo by interacting with the other corner.

Blob Contact

Click the **Blob** button (or press CTRL+B) to simulate an arbitrary ellipse shape. This button works the same way as the **Finger** button. You might want to use blob input when users are pressing different parts of their hands on the surface. For example, if a user leaves a handprint on the screen, the application might need to receive blob input.

Tagged-Object Contact (Byte Tag)

To simulate a byte tag

1. Click the **Tag** button (or press CTRL+T).
2. Click **Tag Value**, and enter the hexadecimal value you want.
3. Click the active area to simulate an object that is touching the screen and lifting up.
4. Press-and-hold in the active area, and then right-click to place the tagged object.

You can leave unlimited tagged-object contacts on the application. The Shopping Cart sample application demonstrates the code to use tagged-object contacts.

Tag Value

Click the **Tag Value** box and enter a valid hexadecimal value ("00" through "FF") for a tagged object.

Note: You can select the tagged-object contact before or after you enter your hexadecimal value.

Important: The updated hex value is applied to:

- The cursor that clicked the **Tag Value** box before changing it.
- All cursors that subsequently select the **Tag** button.

Tagged-Object Contact (Identity Tag)

To simulate an identity tag

1. Click the **Identity Tag** button (or press CTRL+I).
2. Click the **Series** box, and enter the hexadecimal series value that you want ("0000000000000000" through "7FFFFFFFFFFFFFF").
3. Click the **Value** box, and enter the hexadecimal value that you want ("0000000000000000" through "7FFFFFFFFFFFFFF").
4. Click the active area to simulate an object that is touching the screen and lifting up.
5. Press-and-hold in the active area, and then right-click to place the tagged object.

Identity Tag Series and Identity Tag Value

Click **Identity Tag Series** or **Identity Tag Value** boxes, and enter a valid hexadecimal value ("0000000000000000" through "7FFFFFFFFFFFFFFF") for a tagged object.

Note: You can select the tagged-object contact before or after you enter your hexadecimal series and value.

Important: The updated series and value are applied to:

- The cursor that clicked the **Identity Tag Series** or **Identity Tag Value** before changing it.
- All cursors that subsequently select the **Identity Tag** button.

Hide/Show and Remove Contacts

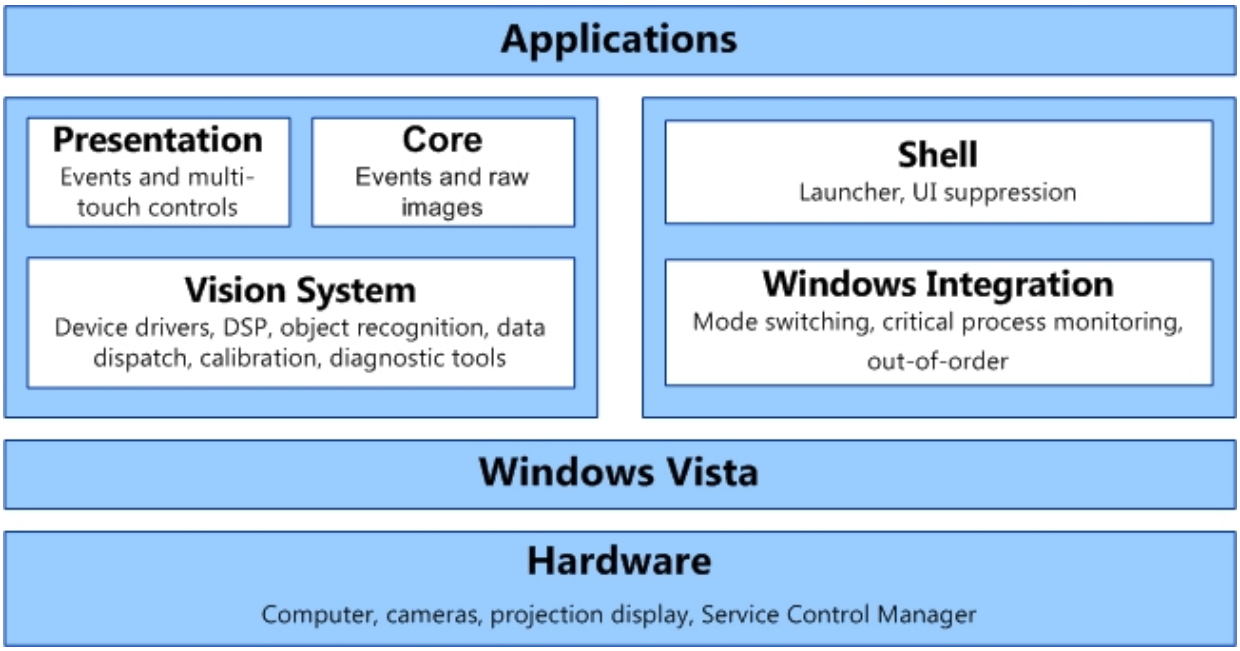
Click the **Remove All Contacts** button to remove all contacts that you have placed in the active area of the Surface Simulator window. You can also press the **Delete** key to remove all currently selected contacts. Click the **Hide/Show Contacts** button to hide all your placed contacts or to show all the contacts that you have hidden. These features enable you to clear any extra contacts from the user interface.

Microsoft Surface
Software Development Kit

Microsoft Surface Architecture

Microsoft Surface is a hardware and software platform for developing multi-input, touch-enabled applications. The platform enables designers and developers to create rich and visually appealing applications that offer a new user experience in which users use only their hands and various objects to manipulate and interact with the applications. A Microsoft Surface application should be natural and intuitive and should show little or no resemblance to a traditional Microsoft Windows or Web application. In fact, a user should not know that there is a computer in a Microsoft Surface unit.

The Microsoft Surface development platform integrates several features and complex hardware and software technologies. To create effective Microsoft Surface applications, you should understand the architecture of the development platform.



Architecture of the Microsoft Surface development platform

The following table describes the components of the Microsoft Surface platform.

Component	Description
Windows Vista	Microsoft Surface runs on the Windows Vista operating system. Windows Vista provides all the administrative, security, and directory functionality of the Surface unit. Developers and administrators who are working on a Microsoft Surface unit have full access to Windows functionality (in *administrator mode*). However, when users interact with Microsoft Surface applications, the Windows user interface is completely suppressed (in *user mode*).
Hardware	The hardware of a Microsoft Surface unit includes the cameras, projection display, and computer that is running Windows Vista. The hardware captures video of contacts on or close to the screen at a specific frame rate.
Vision System	The Vision System software processes the video data that the hardware captures and converts the raw video into data that you can access through Surface SDK APIs. Occasionally, you might have to use a calibration tool to configure the

	cameras for optimum performance. There are two types of calibration: basic and full. You run *full calibration* whenever you move a Microsoft Surface unit to a new location, and you run *basic calibration* whenever lighting conditions change drastically. For more information about how to use the calibration tool, see the Calibrating a Surface Unit article on the Surface Community Web site.
Presentation and Core Layers	The Microsoft Surface SDK informs applications when contacts appear on the Microsoft Surface screen over the application window. As users put contacts on the display and manipulate them, the Microsoft Surface SDK notifies applications and gives them a chance to update their user interface. For each contact, applications can determine the position, orientation, bounding box, and central ellipse. For contacts that are made with *tagged objects* (which have tags printed on the bottom of the objects), applications can also determine the contact tag value. The Microsoft Surface SDK exposes two sets of APIs: the Presentation layer and the Core layer. You can use only one layer when you are developing a Microsoft Surface application: • The Presentation layer integrates with Windows Presentation Foundation (WPF) and includes a suite of Microsoft Surface-enabled controls. • You can use the Core layer together with almost any user interface framework. For more information about the Presentation and Core layers, see Presentation and Core Layers.
Surface Shell	Surface Shell is the component that manages applications, windows, orientation, and user sessions and provides other functionality. Every Microsoft Surface application must integrate with Surface Shell.
Surface and Windows Integration	The integration between Microsoft Surface and the Windows operating system provides system-wide functionality on top of the Windows operating system. You must use this functionality to support unique aspects of the Microsoft Surface experience, such as managing user sessions, switching between the Windows user interface (administrator mode) and the user experience (user mode), monitoring critical Microsoft Surface processes, and handling critical failures.

Microsoft Surface
Software Development Kit

Presentation and Core Layers

The Microsoft Surface SDK includes two sets of APIs: the Presentation layer and the Core layer. Each layer is targeted toward a different development model. The most important planning decision for your Microsoft Surface application is whether to use the Presentation layer or the Core layer. You should choose the layer that is best suited to the requirements of your application.

The following sections describe each layer in more detail:

- Developing Applications with the Presentation Layer
- Developing Applications with the Core Layer

Microsoft Surface
Software Development Kit

Developing Applications with the Presentation Layer

The Presentation layer is one of the two layers of the Microsoft Surface SDK. This layer exposes the same functionality as the Core layer, but it is targeted towards a different development model.

The Presentation layer uses Microsoft Windows Presentation Foundation (WPF), which is the standard choice for developing touch-enabled applications because it facilitates many development needs, including custom controls, touch-enabled by default, and many standard user interface (UI) elements, such as buttons, labels, and scroll bars. WPF also enables you to use custom graphics to create content-rich applications. By building applications with the Presentation layer, you can focus on your application's needs while WPF and pre-built controls handle the common tasks.

The Presentation layer includes a suite of Microsoft Surface-enabled versions of standard WPF controls and supports the use of XAML files for rapid UI development. To minimize how much you have to learn if you have WPF experience, the Presentation layer exposes routed events and attached properties in a way that is consistent with the WPF model for providing you with information about mouse and stylus input. The Presentation layer also enables a multi-capture system so that one or more contacts on a Microsoft Surface unit can capture each UI element.

By using the Presentation layer, you can create an application UI by using XAML and the Microsoft Expression Blend 2 design software. When you combine the application UI with the corresponding C# code, you can quickly and easily develop touch-enabled applications.

To get started and learn how to create a Microsoft Surface application that uses the Presentation layer, see WPF Quick Start.

Microsoft Surface WPF Controls

The Presentation layer of the Microsoft Surface SDK includes the following touch-enabled WPF controls:

Used directly in applications

- **SurfaceWindow**
- **SurfaceButton**
- **SurfaceInkCanvas**
- **SurfaceSlider**
- **SurfaceScrollViewer**
- **SurfaceListBox**
- **SurfaceTextBox**
- **SurfacePasswordBox**
- **SurfaceMenu**
- **SurfaceContextMenu**
- **SurfaceCheckBox**
- **SurfaceRadioButton**

Components of other controls

- **SurfaceScrollBar**
- **SurfaceTrack**
- **SurfacePopup**
- **SurfaceThumb**
- **SurfaceRepeatButton**
- **SurfaceToggleButton**
- **SurfaceMenuItem**
- **SurfaceListBoxItem**

Base classes for custom controls

- **SurfaceControl**
- **SurfaceContentControl**
- **SurfaceUserControl**
- **SurfaceItemsControl**

Microsoft Surface-specific controls

- **ElementMenu**
- **LibraryBar**

- **LibraryContainer**
- **LibraryStack**
- **ScatterView**
- **ScatterViewItem**
- **TagVisualizer**
- **TagVisualization**

> **Important:** The **ScatterView** and **SurfaceListBox** controls are the most versatile controls that are available to you. For more information about these controls, see Manipulations and Inertia.

When to Use the Presentation Layer

Use the Presentation layer when your Microsoft Surface application does not require high-end graphics, such as complex 3-D animation. Use the Presentation layer when you want to take advantage of built-in Microsoft Surface controls that enable you to rapidly create a sophisticated application.

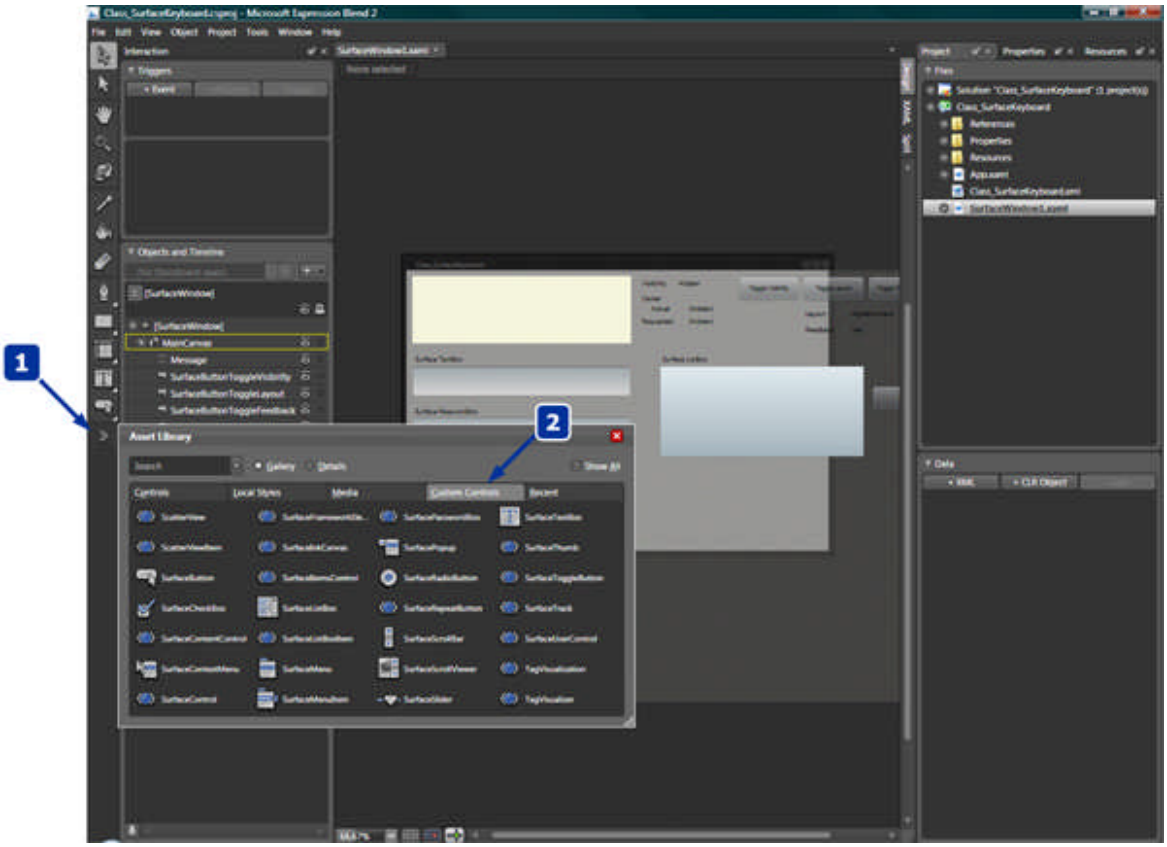
Designing by Using Expression Blend

Expression Blend 2 is an application that helps you create layout. All layout work that you complete in Expression Blend 2 is exported as a XAML file, which is the same file format that WPF-based Microsoft Surface applications use to define UI. You can access Microsoft Surface-enabled custom controls in Expression Blend 2 by importing a Microsoft Surface project from the Microsoft Visual C# 2008 Express Edition (or Microsoft Visual Studio 2008) development system.

To locate and access Microsoft Surface-enabled custom controls in Expression Blend

1. Create a Visual C# 2008 (or Visual Studio 2008) project by using the **Surface Application (WPF)** template, and then save the project.
2. Open Expression Blend 2, and then click **Open Project** on the startup screen.
3. Browse to your saved project, select your project's solution file (*.sln), and click **Open**.
4. In the toolbar on the left side of the screen, click the double arrows (labeled 1 in the following illustration). The **Asset Library** control opens**.**
5. Click the **Custom Controls** tab (labeled 2 in the following illustration), and select the appropriate Microsoft Surface-enabled control. You can now insert the control into Expression Blend.

When you load a Microsoft Surface project in Expression Blend 2, the Asset Library control displays only Microsoft Surface custom controls. The XAML code for the application reflects all UI layout that you complete in Expression Blend 2 after you save the Expression Blend 2 project.



Asset Library and custom controls in Expression Blend

Microsoft Surface
Software Development Kit

Developing Applications with the Core Layer

The Core layer is one of the two layers of the Microsoft Surface SDK. This layer exposes the same functionality as the Presentation layer, but it is targeted towards a different development model.

The Core layer exposes Microsoft Surface-specific contact data and events so you can create Microsoft Surface-enabled applications with any user interface (UI) framework that is based on **HWND**. The samples, project templates, and other components in the Microsoft Surface SDK use the Core layer by using the Microsoft XNA development platform. But you could also use other UI frameworks such as Microsoft Managed DirectX or Microsoft Windows Forms. XNA enables you to create dynamic and sophisticated graphics by supporting complex two-dimensional (2-D) and three-dimensional (3-D) rendering.

Because the Core layer does not depend on a specific UI framework, the Core layer is flexible in how you can use it. But the Core layer also does not have the pre-built controls that you can use with the Presentation layer of the Microsoft Surface SDK.

When to Use the Core Layer

The Presentation layer is powerful, but it does not meet all development needs. Use the Core layer when your Microsoft Surface application requires high-end graphics, such as complex 3-D animation or rich rendering with custom pixel shaders.

Use the Core layer if your application requires access to raw image data from the vision system.

To get started and learn how to create an Microsoft Surface application that uses XNA, see XNA Quick Start.

Microsoft Surface
Software Development Kit

Contacts

The main component of Microsoft Surface applications is touch and contact-enabled controls, events, and gestures. These elements enable you to develop applications that create natural and intuitive user experiences.

Contact Types and Events

The Microsoft Surface SDK recognizes three distinct kinds of contacts:

- *Finger:* One or more fingers are placed on, moved along, or lifted up from the Microsoft Surface screen.
- *Blob:* Any object is placed on, moved along, or lifted up from the Microsoft Surface screen.
- *Tagged object*: Any object that has an attached *tag* is placed on, moved along, or lifted up from the Microsoft Surface screen.

Gestures

Gestures are user interactions that applications recognize without the context of what user interface elements they occur over. Typically, gestures are interactions that do not involve changes in the {X, Y} position of the user input. The Microsoft Surface SDK exposes events for two gestures:

- Tap
- Press-and-hold

Manipulations

Manipulations are user interactions that an application can properly interpret only in the context of user interface elements. For example, an application could interpret two fingers that are moving away from each other as a "stretch" manipulation, as two separate "move" manipulations, or as several other actions. The actual interpretation depends on what user interface elements appear beneath the interactions. The Microsoft Surface SDK exposes manipulations, in the form of controls, that respond to specific manipulations:

- Drag
- Pan (**SurfaceScrollViewer**, **SurfaceListBox**)
- Flick
- Move (**ScatterView**)
- Resize
- Rotate

If you develop your application in the Core layer, you must create all of these contact-enabled elements yourself. The Presentation layer enables you to create versatile and graphically rich experiences with the scroll viewer and scatter view controls.

Manipulations and Inertia Processors

A manipulations processor provides functionality to handle groups of contacts as a single composite. An inertia processor provides functionality for implementing real-world properties like movement and

friction as users move virtual objects around on a Microsoft Surface unit. These processors are implemented in the **Microsoft.Surface.Core.Manipulations** and **Microsoft.Surface.Presentation.Manipulations** namespaces. For more information, see Using Manipulations and Inertia.

Microsoft Surface
Software Development Kit

Tagged Objects

The Microsoft Surface Vision System captures and processes raw images of a Microsoft Surface screen and can "see" the actual outlines of physical objects that are placed on the screen by using cameras that operate in the near infrared light (specifically, 850 nm). (For an example of how to see object outlines, see Capturing and Rendering a Raw Image.)

However, if you want your Microsoft Surface application to be driven by objects, the Microsoft Surface Vision System also recognizes special *tagged objects* that are marked with a special pattern of dots called *tags*. A tag consists of a geometric arrangement of infrared reflective and absorbing areas. Tagged objects give your application several advantages over processing raw images:

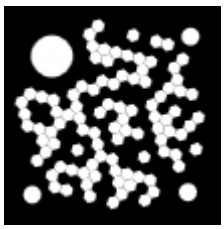
- Tagged objects are more computation-efficient. The tags are small and well-defined, so specialized code in the Vision System can locate and track them quickly, accurately, and efficiently.
- The Microsoft Surface platform inherently recognizes tags, so it is easy to write an application that uses tags. (In contrast, an application that uses raw images must include its own image-recognition algorithms.)
- Each tag stores a distinct binary code value, so an application can distinguish one object from another.

Microsoft Surface supports two types of tags:

- *Byte tags* store 8 bits of data (1 byte), so there are 256 unique tag values. The value of a byte tag is represented in code by a **ByteTag** structure. For more information about byte tags, see Byte Tags.

An example byte tag

- *Identity tags* store 128 bits of data (two 64-bit values), so there is an effectively unlimited range of unique tag values. The value of an identity tag is represented in code by an **IdentityTag** structure. For more information, see Identity Tags.



An example identity tag

Your application's needs determine which type of tag that you should use. For more information, see Designing a Microsoft Surface Application for Tagged Objects. To learn about the requirements for creating and printing tags, see Creating Tagged Objects.

When you use tagged objects, you can use as many instances of the same tag as an application requires. However, if a user moves a tagged object too fast, the Microsoft Surface Vision System

cannot track the object successfully. If you're writing a Microsoft Windows Presentation Foundation (WPF) application, you can create an on-screen visual object that tracks the tag motion by using the **TagVisualizer** control.

How to Use Tagged Objects

A Microsoft Surface application can use tagged objects in several ways:

- **Recognize objects**: An application can use tags to recognize an object or distinguish among a collection of objects. For example, a board game might use different tags to represent the players and trace their positions on the Microsoft Surface screen. Similarly, a chess application might use eight instances of one tag to represent the white pawns, two instances of another tag for the two rooks, two instances of another tag for the knights, two instances of another tag for the bishops, one tag for the queen, and another tag for the king. Therefore, the application would use six unique tags for the white pieces, and another set of six unique tags for the black pieces.
- **Start a command**: An application might use a tag to start a command or action. For example, placing a tagged object on the Microsoft Surface screen might open a menu or start a video.
- **Start or display an application**: Applications can register a particular byte tag value or identity tag series. When a user places the relevant tag on the Microsoft Surface screen, it displays a menu that enables the user to start or display any application that has registered the tag. If a user selects an application by using a tagged object, the application designer can use that tag contact as the starting point for the user's experience with that application. This feature is called *object routing*.
- **Point and orient applications**: An application might require precise movement with absolute orientation. For example, an application might enable a user to move in sequence through a series of video clips, turn the pages in a virtual book, or adjust the volume of a video or audio clip. To adjust the volume, the application might enable the user to turn a tagged object in a circular motion, like the dial on a radio or television. (Microsoft Surface applications enable users to move and rotate an object with their fingers. But tags provide a more precise absolute location than fingers, so applications can use tagged objects as pointing devices.)

Because identity tags have a greater range of possible values, they also enable additional scenarios:

- Casino rewards promotions: A casino marketing department produces an offer for $100 in free slot play. The printed offer includes the identity tag with the customer ID and a unique offer code, such as a casino chip. The customer brings this offer to the venue and places it on a Microsoft Surface screen. The customer code and offer are validated, and a $100 credit is associated with the customer account as downloadable credits. The next time the customer inserts their loyalty card into a slot machine, they receive the $100 credit to the machine.
- Theme park photo pass: Passes are issued to guests of the park each day. The pass has an identity tag ID on it. As photos are taken throughout their visit to the park, the digital pictures are associated with the ID on the photo pass. The guests then place the pass on a Microsoft Surface screen and all the photos that are associated with the pass appear on the screen for the guests to view, manipulate, and purchase.
- Hotel loyalty cards: Identity tags are placed on loyalty cards to identify customers. Premier customers have access to more services and applications on a Microsoft Surface unit than typical customers or those without a loyalty card.
- Fast food kids' toys: Identity tags are placed on kids' toys. A user places the toy on a Microsoft Surface screen and the application responds to the toy in some way.

Note: Any application that uses tagged objects should register the tags that it uses. For more information, see Registering a Microsoft Surface Application.

Security Considerations for Tagged Objects

Do not use tagged objects to access private data. Tags are not sufficiently secure to use as a method of authentication. You can use tagged objects to identify a user, but you should require the user to

enter a password, personal identification number (PIN), or other secure data for authentication purposes.

When you use tags, consider the following important security issues:

- An identity tag does not have enough bits to be cryptographically secure. Do not use an identity tag as a password equivalent.
- Tag bits are not encrypted when they are in memory. For example, they are not equivalent to the **System.Security.SecureString** class in the .NET Framework. When you write code that works with tags, imagine that you are working with strings. That is, in your application, is **System.String** sufficiently secure, or would you need to use **SecureString**? If you would use **SecureString**, do *not* use an identity tag for the data.
- Depending on how tags are printed, photocopiers, digital cameras, and other devices can easily capture the tags so they are easy to duplicate.

Overall, we recommend that you can use tags for identification but you should not use them authentication. An identity tag is simply a way to enter shorthand, intended to save you from having to type something into a text box. It's the equivalent of having a hyperlink that you can click on, instead of forcing you to enter a URL, or the equivalent of pressing CTRL+R to respond to an e-mail, instead of entering the recipient's address manually. The contents of an identity tag are not more secure than a URL or an e-mail address.

The remaining topics in this sections provide more information about how to create and use tagged objects:

- Byte Tags
- Identity Tags
- Designing Applications for Tagged Objects
- Creating Tagged Objects
- Using Object Routing with Tagged Objects
- Specifying Physical Attributes of Tagged Objects

Microsoft Surface
Software Development Kit

Sample Applications

The Microsoft Surface SDK contains the following samples:

Samples That Use the Presentation Layer

- Controls Box shows how to build simple application behavior, such as updating a text box when a user touches button, from touch-enabled controls that the Presentation layer provides.
- Data Visualizer shows contact properties that are exposed in the Presentation layer (such as x, y, height, width, major, minor axis, and orientation) and how you can read and use these properties in a Microsoft Surface application.
- Fractal Browser displays an H-fractal. A user can zoom the image to be larger than the Microsoft Surface screen and can rotate and pan the content.
- Grand Piano demonstrates how to integrate sound into Microsoft Surface applications based on the Presentation layer.
- Item Compare represents a simple tool that lets a user compare and contrast the properties of two "items" (tagged objects).
- Paddle Ball uses WPF controls to implement contact tracking and movable user interface elements as part of a four-person game.
- Photo Paint uses the **SurfaceInkCanvas** control to implement drawing and painting over pictures and video.
- Scatter Puzzle shows an implementation of the **ScatterView** and **SurfaceListBox** controls to create a simple puzzle game. The **ScatterView** and **SurfaceListBox** controls automatically provide some powerful Microsoft Surface-related features.
- Shopping Cart shows how to implement drag-and-drop functionality in a retail application.
- Tag Visualizer Events shows how to incorporate hit-testing in the **TagVisualizer** control to let UI elements react when tagged objects move over them.

Samples That Use the Core Layer and the XNA Framework

- Finger Fountain draws small images for every contact at every frame. This sample emphasizes multiple touches and shows how to use the Microsoft XNA APIs.
- Cloth is an XNA-based application that demonstrates how to use the Core Interaction Framework.
- RawImage Visualizer shows how to use the **RawImage** APIs for XNA applications. This sample displays captured normalized (8 bit per pixel) images that are flipped vertically.
- XNA Scatter demonstrates how to use the manipulations and inertia APIs to to move graphical user interface (GUI) components in a Microsoft Surface application around in a natural and intuitive way.

Samples That Use the Surface Shell API

- Activate Application from an Application demonstrates how to activate one application from another Microsoft Surface application.

- Attract Application demonstrates a basic Microsoft Surface attract application, including the necessary XML file needed to register as an attract application.
- Notifications demonstrates how to display messages to the user, whether your application is currently active or is running in the background.

Installing the Samples

The samples are included in the Surface Code Samples.zip file that is installed into the %ProgramFiles%\Microsoft SDKs\Surface\v1.0\Samples folder. A link to this ZIP file also appears on the Start menu under **Microsoft Surface SDK 1.0 SP1** -> **Samples**. Extract the contents of ZIP file to a directory where you have write permissions.

A batch file called InstallSamples.bat is included in the ZIP file. In Windows Explorer, right-click the batch file, and then click **Run as administrator**. The batch file builds each SDK sample and creates the XML file needed to register the application with Surface Shell. It then copies the XML file to the correct directory so it will show up in Launcher.

After the batch file is finish, do one of the following:

- On a Microsoft Surface unit: Start Surface Shell by using the desktop shortcut. Browse the samples by using Launcher. If the samples are in a public directory (that is, any user can access them on the system), samples will also be available in user mode.
- On a computer: Start Surface Simulator and use Launcher to browse the samples.

If the batch file fails, check to see that it is not being run directly from the ZIP directory and that it is being run as administrator. Alternatively, open SDKSamples.sln in Microsoft Visual C# 2008 Express Edition (or Microsoft Visual Studio 2008) and install the built samples manually to the Surface Shell.
