DEVELOPING APPLICATIONS FOR MICROSOFT SURFACE 2.0

June 2011

CONTENTS

Introduction	**3**
Microsoft Surface Software Architecture	**4**
Surface Hardware Components	4
Surface Software Components	5
Introduction to Surface Application Development	**7**
The Surface Version 2.0 SDK	8
Surface Application Development Overview	12
Application Responsibilities	12
Adding Tag Support to your Applications	13
Special Applications	13
Creating an Application by Using the ScatterView Control	14
Creating an Application by Using the TagVisualizer Control	16
Adapting Surface Applications to Run on Windows 7	17
Testing, Marketing, and Deploying Your Applications	**18**
Testing Applications	18
Creating Tagged Objects for Your Applications	19
Certifying Your Application	20
Microsoft Provided Marketing and Partner Support	21
Summary	**21**
For More Information	**21**
Legal Notice	**22**

INTRODUCTION

Microsoft Surface enables you to deliver new kinds of experiences for the people who step through your customers' doors in the hospitality, retail, healthcare, financial, business, education, and IT service industries. Ordering drinks from the bar? Choosing features for a new automobile? Evaluating different mortgage options? Surface enables these experiences to come to life in new, innovative ways. The world of Surface is one of intuitive touch interactions and familiar objects recognized and understood.

With a thin 40" high definition screen placed horizontally or vertically, the Samsung SUR40 for Microsoft Surface is large enough to serve as the centerpiece of engagements that involve several people working together, playing together, and exploring together. PixelSense™ enables Microsoft Surface to see and respond to touch and objects. It is optimized for over 50 unique points of simultaneous touch, affording unprecedented opportunities to bring people together.

People can use the 360° interface from all sides for face-to-face collaboration, cooperation, and trust building. It's intimate enough to facilitate the close back-and-forth of a consultative sale. Surface makes intuitive sense to everyone, young and old. Touch. Push. Pull. Turn. Surface creates opportunities for social interaction where barriers existed before.

So why Surface? Simply put, there is no easier way to bring people together to connect, learn, have fun, and reach decisions. Surface is what computing always promised to be – intuitive, informative, engaging, and helpful – and with Surface you enable the people who matter most to your customers to have an experience they will long remember. Surface touches people, even as they touch Surface.

As a developer you must familiarize yourself with the Surface architecture and the features that are available in the Surface 2.0 SDK. It is particularly important to understand how to program your applications to respond to touch, leverage the unique capabilities of vision based multitouch hardware, and the implications of creating applications that engage your users with a 360° interface.

SURFACE HARDWARE COMPONENTS

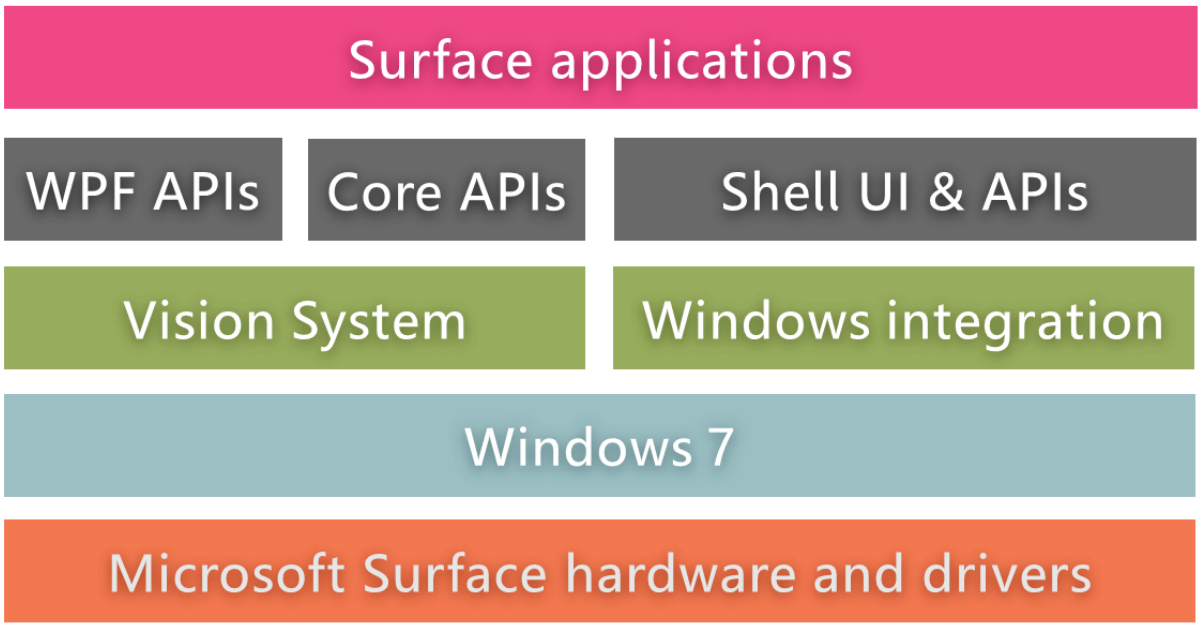
The Microsoft Surface 2.0 platform is optimized for the Samsung SUR40 for Microsoft Surface. This vision based input display is a 4 inch thin LCD screen that utilizes PixelSense™, which makes the screen see invisible infrared light. This capability enables the device to recognize and instantly respond to touch and objects placed on the screen. The thin form factor makes it easy to deploy the device in horizontal and vertical positions, and fit it in more locations than ever before.

The Samsung SUR40 has an integrated Windows 7 PC providing powerful input and graphics processing. The device also includes USB ports, digital audio out, HDMI in and out, Bluetooth and WiFi connectivity.

The Samsung SUR40 for Microsoft Surface – shown with optional legs

SURFACE 2.0 SOFTWARE ARCHITECTURE

The following figure represents the entire Surface platform, including the hardware layer and software components that you must be aware of when you develop Surface applications.



SURFACE 2.0 SOFTWARE COMPONENTS

The Surface software components not only enable you to respond to multitouch and tagged objects in your applications, but also provide the basics of the Surface user experience. These include integration with Windows 7, the Launcher application, and the suppression of user interface (UI) messages from the Windows operating system and other programs running on the platform.

Windows 7

Microsoft Surface 2.0 runs on the Windows 7 64-bit operating system. Windows 7 provides all the administrative, security, and directory functionality of the Surface unit.

Developers and administrators working with a Microsoft Surface unit have full access to Windows functionality (in Windows Mode). However, when people interact with Microsoft Surface applications, the Windows user interface is suppressed (in Surface Mode).

Integration with the Vision System

The Vision System uses PixelSense™ to process captured touch data by converting raw visual information into lightweight, easily useful application data you can access through Surface SDK

APIs. PixelSense™ enables each pixel in the Surface display to detect when a person touches it or when someone moves a finger, tagged object, or untagged object over it. It does this without the use of cameras, which is what makes newer Surface hardware much thinner than previous versions. The Surface 2.0 SDK also provides APIs that enable you to create advanced applications that access raw images from the vision system.

Presentation layer

The Presentation layer integrates with Windows Presentation Foundation (WPF) and includes a suite of interaction controls designed for Microsoft Surface enabling you to quickly and easily build touch enabled applications.

Core layer

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. Though the Visual Studio 2010 templates are based on the XNA Game Studio 4.0, you can use the Core layer with almost any user interface framework.

Windows integration

The tight 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 standard Windows user interface (Windows Mode) and the deployment experience (Surface mode), monitoring critical Microsoft Surface processes, and handling critical failures.

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.

INTRODUCTION TO SURFACE APPLICATION DEVELOPMENT

There are four features of the Microsoft Surface software platform that are different from traditional PC environments that you must take into account when you create your applications:

1. **People use the 360° degree Surface interface from all sides**
   People will use your application from all angles around the Surface unit. You must design applications that face content to those individuals.
2. **Surface is optimized for over 50 unique points of simultaneous touch**
   You must create applications that elegantly embrace the Surface multitouch hardware capabilities, regardless of how many people work with your application at the same time.
3. **Surface uses UI suppression to enhance the experience**
   The Surface environment suppresses standard Windows dialog boxes, such as update messages seen in Windows Mode. You should develop your applications so that they do not raise such messages in Surface Mode.
4. **Surface recognizes physical objects placed on the screen**
   Objects are used to simplify the experience, let people share application control, and can provide identification and location of the people using the device.

When you develop Microsoft Surface applications, you can:

- **Develop directly on a Microsoft Surface unit.** If you choose to develop on a Surface unit, you can run the unit in two modes. A device administration mode, **Windows Mode**, lets you access all of the Windows 7 functionality. In contrast, the primary deployment mode, **Surface Mode**, demonstrates how the unit appears in a commercial venue. That is, the unit suppresses the Windows 7 user interface, automatically starts required applications, and makes sure that those applications are restarted if they need to be. This enables you to run tests on the Surface environment as you develop the application. Typically, you develop applications in Windows Mode and test and run them in Surface mode.
- **Develop on a separate workstation.** On a separate workstation, you can install the Microsoft Surface 2.0 SDK. If you develop on a Windows 7 touch enabled PC, you can test how your application responds to touch as you develop the application. If you develop on a Windows 7 PC that is not touch enabled you can use the **Surface Input Simulator** tool to simulate Microsoft Surface finger, tag and blob touch inputs. There are significant differences between using the Surface Input Simulator and an actual Surface unit. For example, raw vision input is not available. If you write an application that uses raw vision input, you cannot test it by using Surface Simulator. Also, the Surface Input Simulator does not enable testing Shell functionality or integration. Thoroughly testing your applications on Surface hardware is always a must.

THE SURFACE 2.0 SOFTWARE DEVELOPMENT KIT (SDK)

While many of the techniques and approaches used to create Surface 1.0 applications are the same, there are a number of significant differences when developing for Surface 2.0. This section will touch on those differences and describe some of the background required to develop for the Surface 2.0 platform.

The following are the two most significant changes to the Surface 2.0 software environment that affect how you code your applications:

- **The Surface computer now runs the Windows 7 64-bit operating system.** The benefits of this for you and your organization are twofold. First, you get to take advantage of all the built-in security and manageability of Windows 7 for your Surface applications. Second, with proper planning and minor design modifications, you can target your applications to run on Surface units and Windows 7 touch enabled PCs; immediately expanding your application's reach. For more information about designing your Surface applications to run on Windows 7, see the Adapting Surface Applications to Run on Windows 7 section in this whitepaper.
- **WPF based Surface applications now use WPF 4.0 touch and manipulation processing.** Microsoft Surface 1.0 used its own custom manipulation processor and you had to use classes unique to the Surface SDK to create inertia and track movements if a person flicked and released an object on the Surface display. Windows Presentation Foundation (WPF) 4.0 now includes the manipulation and touch events that the Surface 2.0 SDK uses to calculate touch position and simplify development.

Requirements

To install and develop applications using the Microsoft Surface 2.0 SDK, you must you have the following software installed on your Windows 7 PC workstation, Windows 7 touch PC, or Surface unit that you are using for development:

- Windows 7 operating system (64-bit recommended)
- .NET Framework 4.0
- Visual Studio 2010 or Visual C# 2010
- XNA 4.0 Redistributable or XNA Games Studio 4.0
- Microsoft Expression Blend 4.0 (recommended)

Additionally, you must create your applications to run on hardware designed for Surface 2.0, such as the Samsung SUR40 for Microsoft Surface, or Windows 7 touch PCs.

The SurfaceWindow class

The **SurfaceWindow** class provides the base control on which you create your WPF based Surface application. When you create a Surface application using the WPF 4.0 templates, Visual Studio 2010 automatically creates a control that provides a base implementation of a **SurfaceWindow** control.

The **SurfaceWindow** class derives from the **System.Windows.Window** class and activates the necessary event handlers to make sure that it captures touch events and displays touch visualizations.

The Core layer

The Core layer exposes Microsoft Surface specific touch 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 Game Studio 4.0 development platform. XNA Game Studio 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, you can use it in a variety of ways. However, the Core layer also does not provide pre-built controls that you can use with the Presentation layer of the Microsoft Surface SDK.

Although there are no controls associated with the Core layer, the Surface SDK does provide the Core Interaction Framework. If you develop your Microsoft Surface application by using the Core layer, the Core Interaction Framework enables you to implement user interface (UI) controls that emulate the interactive behaviors of Microsoft Windows Presentation Foundation (WPF) controls in the Presentation Layer. The Core Interaction Framework also provides several other benefits:

- Uses the Model-View-Controller (MVC) design pattern so that the framework does not depend on a specific UI framework and it separates development and maintenance of components.
- Supports event-based and loop-based usage models. In the event-based model, events are raised on an application-requested thread on the application-defined time frame.
- Does not place restrictions on application features, such as scene graphics, collision, or render engine design. Your application controls all visual aspects of the user interface.

The Presentation layer

The Presentation layer is a Surface specific extension of Microsoft Windows Presentation Foundation (WPF). The Presentation layer is the typical choice for developing touch enabled applications for Surface because it provides support for many development needs, including

support for custom controls, a default touch enabled framework, and many standard interface elements, such as buttons, labels, and scroll bars. WPF also enables you to use custom graphics and media to create content oriented applications. Building applications with the Presentation layer, keeps focus on your application's needs while WPF and pre-built controls handle the common tasks.

Using the Presentation layer, you can design the application user interface (UI) using XAML and the Microsoft Expression Blend 4.0 design software. Combine the application UI with the corresponding C# code to quickly and easily develop touch enabled applications.

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

Additionally, the Presentation layer in Surface 2.0 uses the touch and manipulation events provided through WPF 4.0 to respond to multitouch. It also relies on the WPF 4.0 manipulation processor to process the information provided by those events. For more information, see the **Touch and Manipulation** section of the Input Overview topic on MSDN.

Surface SDK controls

The Presentation layer includes a suite of Microsoft Surface optimized versions of standard WPF controls, and also 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.

Surface controls provide an easy way to incorporate functionality into your applications, but the Surface Design and Interaction Guidelines encourage you to use controls only on an as needed basis. Indirect touch controls should not take the place of direct touch as the primary way for people to interact with application content. When controls are used in your application, be certain to size them correctly for touch. For more information, see the Surface Design and Interaction Guidelines.

The following table contains a list of the Surface specific SDK controls:

Control	Description
ElementMenu	Provides a hierarchical menu with menu nodes that your application can associate with various commands.
LibraryBar	Enables you to display UI content items horizontally, organize items into groups, and scroll the groups.
LibraryContainer	Provides a dual view control that arranges items in a horizontal bar or in a vertical stack and enables you to switch back and forth between the two views.
LibraryStack	Enables you to display UI content items that are stacked on one another.
ScatterView	Enables you to display one or more UI content elements that you want users to be able to move, rotate, or resize freely within a fixed area.
SurfaceButton	Enables you to add a region of the screen that is labeled with a command. When a person touches the control, the button command runs.
SurfaceCheckBox	Enables people to choose between two clearly opposite choices. The **SurfaceCheckBox** label indicates the selected state, and the cleared state indicates the direct opposite of the selected state.
SurfaceListBox	Enables people to select from a set of values that are presented in a list that is always visible.
SurfacePassowrdBox	Enables you to include a text box that is specific for people to enter a password into the Surface system.
SurfaceRadioButton	Enables you to include a button that people can select, but not clear.
SurfaceScrollBar	Enables you to include a scroll bar that lets people move to a particular portion of a list, display,

	or other viewing area.
SurfaceScrollViewer	Enables you to create a scrollable area that can contain other visible elements.
SurfaceTextBox	Enables you to include a text box in a Surface application. When people touch the control, an on-screen keyboard appears to enable them to enter text.
TagVisualizer	Enables you to add a control that enables your application to respond to tagged objects.

SURFACE APPLICATION DEVELOPMENT OVERVIEW

Now that you know that basic features the Surface 2.0 SDK offers you as a programmer, this paper will discuss aspects of using those features to create compelling applications for people using Surface.

Application responsibilities

When you develop a Surface application, there are five responsibilities you must ensure your application handles.

1. **Start and orient the application.** No matter which layer you use for your application code, your application should call certain Shell APIs that control how the application starts.
2. **Indicate the loading is complete.** You application must signal to the Microsoft Surface Shell that it has completed loading so that the loading screen can be dismissed.
3. **Register your application with the Microsoft Surface Shell so that it appears in Launcher.** This means that you must include information in the application registration XML file that is created when you create your application in Visual Studio 2010.
4. **Do not block the UI thread.** The Surface 2.0 SDK uses the WPF 4.0 threading model. In this model, the UI thread receives user input, handles events, paints the screen, and runs application code. The UI thread queues work items inside an object call a Dispatcher. The Dispatcher selects work items on a priority basis and runs each to completion. When creating Surface applications, it is important to keep each work item small so that other work items do not sit in the Dispatcher queue waiting to be processed. Surface sets a time limit of 5 seconds for the Dispatcher to process a work item. If a work item takes longer than that to process, the Surface system will close and restart the unresponsive application twice. If the system must close the application a third time for unresponsiveness, it automatically returns to the Launcher and does not attempt to start the application again.
5. **Localize the application content, when appropriate.** Storing UI text strings in a separate XML file will help you quickly address localization needs – giving you the ability to rapidly adapt your application for another world language.

For more information about these responsibilities, see the Surface 2.0 SDK documentation.

Adding tag support to your applications

When people place a tagged object (an item that contains a special printed pattern) on a Microsoft Surface screen, the Microsoft Surface software interprets the value that tag represents. You can program your application to respond to particular values by displaying user interface (UI) elements. These elements can take any form that you need for your application. For example, your application might respond to a tag that has a particular value by displaying a form that contains additional information about the item that the tagged value represents.

The easiest way to do this in a WPF based Surface application is to add a **TagVisualizer** control to your application. The **TagVisualizer** is hosted within your application's **SurfaceWindow** control. Depending on your application's design, the **TagVisualizer** control might use the entire window, or only a portion of it.

A person must place a tagged object within the boundaries of the **TagVisualizer** control to generate a response. Your application defines which tag values that the **TagVisualizer** control recognizes by creating objects that are derived from the **TagVisualizationDefinition** class (**ByteTagVisualizationDefinition**, **IdentityTagVisualizationDefinition**, or your custom class definition) and adding them to the **Definitions** property of your **TagVisualizer** control. Two or more definitions can share the same **TagVisualization** user interface.

When a person places a tagged object on a Microsoft Surface screen within the boundaries of a **TagVisualizer** control, the Microsoft Surface platform begins a process that can result in the display of additional user interface elements. During this process, your application can make runtime changes by handling various events.

SPECIAL APPLICATIONS

There are two types of special applications that you can develop to run on Surface units. These are service applications and applications that you design to run alone on a Surface unit.

Service applications

Service applications run in the background and interact with the Surface Shell component. A service application does not provide user interface elements. For example, you can create a service application that initiates user notifications, such as printing complete messages or network messages received by the Surface unit. A user notification can also give people the option of starting a different application. You can also use service applications for continuous media operations or system monitoring.

Single Application Mode applications

Applications that run in Microsoft Surface Single Application Mode have different requirements than applications that run in standard Surface Mode. Single Application Mode enables your customers to configure a Microsoft Surface unit to run only one application, all the time. For example, a hotel might put a Microsoft Surface unit in the lobby and want to run one application that lets customers explore the amenities and features of the hotel. This application could:

- Describe the hotel's restaurants and enable people to make reservations.
- Describe spa packages and exercise and pool facilities.
- Show photos of the hotel's banquet and function
- Link to the hotel's other properties in different geographical locations.
- Describe special offers.

When you set up a Microsoft Surface unit to run in Single Application Mode, the unit does not show an Attract Application or Launcher. When you start the unit, it briefly displays a simple screen with the Microsoft Surface name and logo while the application loads, and then the application starts. Single Application Mode applications do not close or move to the background. However, you can include close or timeout functionality into the application itself. For example, by using the preceding example, you could add a reset button so people can reset the application to its original state if someone walks away and leaves the application showing restaurant information. You could also add an automatic reset if the Microsoft Surface unit does not get a contact after 20 seconds. The automatic reset could also clear any personal information that a guest entered, such as a spa appointment.

CREATING AN APPLICATION USING THE SCATTERVIEW CONTROL

The Microsoft Surface 2.0 SDK includes several controls that are designed specifically for creating experiences that are only possible by using the Microsoft Surface vision based multitouch input capabilities. Of these, the **ScatterView** control enables you to quickly create an application with a full 360° interface for your application content. The **ScatterView** enables you to have multiple interface elements within a fixed area on the Surface unit. It also enables people to move, rotate, or resize each element freely within the application.

When you create a Surface application using a **ScatterView** control, you include **ScatterViewItem** controls in the XAML for the **ScatterView** control to define the items to display. The Surface SDK provides the following ways for you to accomplish this:

- **Adding ScatterViewItem controls at design time.** You can do this either by adding XAML to the **ScatterView** control for each item, or you can drag and drop the item controls from the Expression Blend or Visual Studio toolbox onto the design surface.
- **Adding ScatterViewItem controls programmatically.** You can create event handler code that adds items to a **ScatterView** control when a user touches a region of the display. The following XAML is a sample **ScatterView** control:

```
<s:ScatterView x:Name="scatter"/>
```

The following sample code adds a specific JPEG file to the **ScatterView** control's Items collection when added to the event handler:

```
Image img = new Image();
img.Source =
     new BitmapImage(new Uri("Desert Landscape.jpg", UriKind.Relative));
scatter.Items.Add(img);
```

- **Adding ScatterViewItem controls by using databinding.** You can also populate a **ScatterView** control by using databinding. The following XAML includes a **ScatterView.ItemTemplate** element, a **DataTemplate** element, and an Image element that is ready for databinding.

```
<s:ScatterView x:Name="scatter">
    <s:ScatterView.ItemTemplate>
        <DataTemplate>
          <Image Source="{Binding}"/>
        </DataTemplate>
      </s:ScatterView.ItemTemplate>
</s:ScatterView>
```

You must also include code in the constructor for the window that hosts the **ScatterView** control to obtain the location of the content you want to display in the control. You then set the **ItemSource** property to the path and file type to display in the **ScatterView**.

```
public SurfaceWindow1()
{
    InitializeComponent();
    // Add handlers for application-activation events.
    AddActivationHandlers();
    // Standard path to sample images in Windows 7.
    string path = @"C:\Users\Public\Pictures\Sample Pictures";
    scatter.ItemsSource = System.IO.Directory.GetFiles(path, "*.jpg");
}
```

Using one of these approaches, you can take advantage of the built-in support that the **ScatterView** control provides for moving, rotating, and resizing the items that it contains.

You can also extend the **ScatterView** control by creating custom classes that you use in place of **ScatterViewItem** controls. This enables you greater control in the types of data each item displays and how users interact with it.

Regardless of which technique you use, it is important that your team fully designs how the item controls display in the **ScatterView** control, and how each item or item group responds to touch, objects, and tag values.

CREATING AN APPLICATION USING THE TAGVISUALIZER CONTROL

If you want to create an application that enables the use of tags, the quickest way to do so is to use the **TagVisualizer** control. A **TagVisualizer** reacts to one or more tagged physical objects that are placed on the Surface screen by creating and displaying **TagVisualization** objects. A **TagVisualization** object represents the user interface that users see. A **TagVisualizer** control automatically tracks the motion of a tagged object and moves the **TagVisualization** object with the physical object.

Generally, use the following steps you take to create an application based on a **TagVisualizer** control:

1. Create a WPF based Surface application in Visual Studio 2010.
2. Choose where you want to host the tag visualizations for the application. By default, Surface uses the **TagVisualizationCanvas** class to host visualizations. However, you can host visualizations in a **ScatterView** control, or in a custom visualization host control.
3. Create **TagVisualization** objects for the visualizations that you plan to include in your application. Though you can associate visualizations with specific values on tagged objects, this is not required.
4. Create a **TagVisualizationDefinition** element to associate the visualizations you have created with the **TagVisualizer** control.
5. Create event handler methods to process changes in tag values or in the location of the tagged object on the surface display.

For more information about how to create applications using the **TagVisualizer** control, see the Surface version 2 SDK documentation.

PREPARING SURFACE APPLICATIONS TO RUN ON WINDOWS 7

Because the Surface 2.0 platform runs on the Windows 7 operating system, it is easy to prepare any applications you develop for deployment on Windows 7 touch PCs.

There are a number of steps you should take to design your applications so they will run on Surface units and Windows 7 touch PCs.

- **Design the application for horizontal and vertical deployments.** To do this you can write code that uses the **InteractiveSurfaceDevice.Tilt** property to determine whether the orientation of the display is vertical or horizontal. You can modify the behavior of your application for the display tilt.
- **Check the hardware capabilities of the system on which the application is running.** The **InteractiveSurfaceDevice** class not only provides information about whether the display is tilted, it gathers lots of other information about the hardware your applications run on. This information includes whether the display supports touch input, finger and tag recognition, and the maximum number of touches points supported by the hardware.
- **If necessary, remove icons and images that would display upside down in a vertical deployment of your application.** Typically you could do this by checking the value of the Tilt property as mentioned previously.

Finally, be sure to thoroughly test your application on a Windows 7 touch PC if you plan to deploy it to that environment.

TESTING, MARKETING, AND DEPLOYING YOUR APPLICATIONS

There are a number of steps beyond coding your Surface application that you need to take to ensure that they are ready for your customers and reach the people using your application as quickly and efficiently as possible. These steps include the following:

- Testing your application in Surface Mode on a Surface unit.
- Creating tagged objects to distribute with your applications, if necessary.
- Certifying your applications.
- Taking advantage of marketing and partner support from Microsoft and the device manufacturer.

Testing Applications

The testing process occurs throughout application development. The Surface 2.0 SDK and Visual Studio 2010 provide an environment for you to test and debug your applications throughout the development lifecycle. When you are ready to stress test your Surface applications, you can use the Surface Stress tool included with the SDK to test their stability and robustness.

You can test and debug in the following ways:

- **On a developer workstation**. If your workstation is a Windows 7 touch PC, you can test some touch capabilities with that device. Regardless of whether your Windows 7 PC is touch enabled, you will probably use the Surface Input Simulator and the Visual Studio debugger to test as you develop your application.
- **On a Surface unit running in Windows Mode**. This is a good way to test how your application interacts with fingers and objects touching the Surface display, but you should not use this for final application testing.
- **On a Surface unit running in Surface Debug Mode**. This enables you to use an external keyboard and start other applications on the Surface unit as you test your application.

While testing in Surface Debug Mode can be useful for troubleshooting, you should not use it to develop your applications.

- **On a Surface unit running in Surface Mode.** Because Surface application developers and testers frequently overlook this critical step, the following paragraphs concentrate on this topic.

You should perform your final application testing steps in Surface Mode, as it replicates the deployed experience that people will use. Testing in Surface Mode enables you to check whether the Surface UI suppression works properly with your application, whether your application integrates properly with the Surface Shell, and whether your application relies on elevated privileges that will cause the application to break when your customers deploy it on their units.

Additionally, be sure to test the application thoroughly with several people interacting on all sides of the Surface unit. If you intend your application to be used only in vertical deployments, test the application using a unit that is vertically deployed. Otherwise, you should test your application from all four sides of the Surface unit with at least six people to make sure that content orients correctly to each person when they touch the application, and that the additional processing load does not degrade your application's performance.

Creating tagged objects for your applications

When you create tagged objects to use with your applications, there are a number of factors that you must consider:

Reflectance

- White dots must reflect between 73% and 86% of 850 nanometer (nm) infrared light.
- The dark region of the tag must absorb 93% or more of 850 nanometer (nm) infrared light.
- A single tag on a physical object with a black or non-IR reflective background provides the best tag recognition performance.

Materials and Placement

- Whenever possible, place tags on infrared absorbent objects. If you place a tag on a highly infrared reflective surface, the Microsoft Surface Vision System might have trouble recognizing the tag.
- Place tags on a solid, consistent background (for example, with no patterns or stripes). Use a dark-colored or non-IR reflective background instead of a light background.

- Place tags on flat surfaces that help the tag maintain flush contact with the tabletop. Also, tags work better on physical objects that do not easily tip over while they move.

Printers

- Ink jet printers that use a mix of cyan, yellow, and magenta to create black generally do not work. The three-color combination is typically invisible in the infrared spectrum. You should test all printing methods for their appearance in the infrared 850 nm wavelength.
- Be careful that the tags cannot be rubbed off. Tags that are printed by using ink jet or laser printers can wear off rapidly when they are used. If the tag printing wears off, you cannot use the tag anymore and the tag will smudge the Microsoft Surface screen. The tags that are included with the unit are printed on self-adhesive vinyl to protect them from wearing off.

For more specifics on the requirements to create effective tags for your application, see the Microsoft Surface 2.0 SDK documentation.

Certifying Your Application

After you complete testing of your application, you can submit your application to the Microsoft Surface certification authority. Certification sets the quality bar for applications across the Microsoft Surface ecosystem – ensuring that they are compatible, reliable, and secure when running on Microsoft Surface. Achieving the certification requirements also ensures that people using your application have a consistent, high quality experience. If you application meets all of these requirements, it enables you to include the Certified for Microsoft Surface logo on your packaging and marketing materials that your organization creates for your application. Meeting the certification requirements requires advanced planning, and you must plan for certification from the beginning of the application design process.

For your application to be certified, it must meet requirements in five areas:

- Security and compatibility
- Installation
- Reliability and performance
- Configuration and API usage
- User experience

For more information about the Surface certification process, see the Certifying Applications for Microsoft Surface 2.0 whitepaper on the Microsoft Surface Developer's Center. You can also visit the LionBridge Web site and search for Certified for Microsoft Surface.

Microsoft Provided Marketing and Partner Support

As a Microsoft Surface application developer or service provider, Microsoft provides you with a variety of benefits to help your business be successful. These benefits include training on the Microsoft Surface platform, support for creating marketing campaigns for your business and applications, and sales support for your applications. It also provides sales support for your organization if you sell Microsoft Surface units along with applications. For more information, visit the Microsoft Partner Network.

SUMMARY

With Surface, you can create amazing applications that connect people to each other and your customer's content in brand new ways. If you are a developer interested in creating highly engaging, social experiences for several people at the same time, Microsoft Surface provides a unique platform for you to target your applications. With experience in WPF or XNA and touch application design, you can quickly learn to create applications that will engage people's senses.

The Surface software platform is ideal for any scenario where people need to interact together on a single device. The Surface focus is on creating real connections – connections with customers and one another.

MORE INFORMATION

The Surface product website	http://www.surface.com
Microsoft Surface Design and Development Center	http://www.msdn.com/windows/surface
Microsoft Partner Network	http://partner.microsoft.com
LionBridge Surface Certification	http://www.lionbridge.com/surface

LEGAL NOTICE

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.