

Microsoft Surface

Software Development Kit

XNA Quick Start

This section shows how you can create a Microsoft Surface touch-enabled application in the [Core layer](#) by demonstrating how to create a touch-enabled user interface element within a Microsoft Surface application. The application displays a **Hello, World!** message when a user touches the Microsoft Surface screen.

To complete this tutorial, read the following topics, in this order:

1. [Starting the Visual Studio Project for XNA Applications](#)
 2. [What the XNA Template Creates](#)
 3. [Editing the C# File for the XNA Template](#)
 4. [Running and Extending Your XNA Application](#)
-

[© 2009 Microsoft Corporation. All rights reserved.](#)

Microsoft Surface

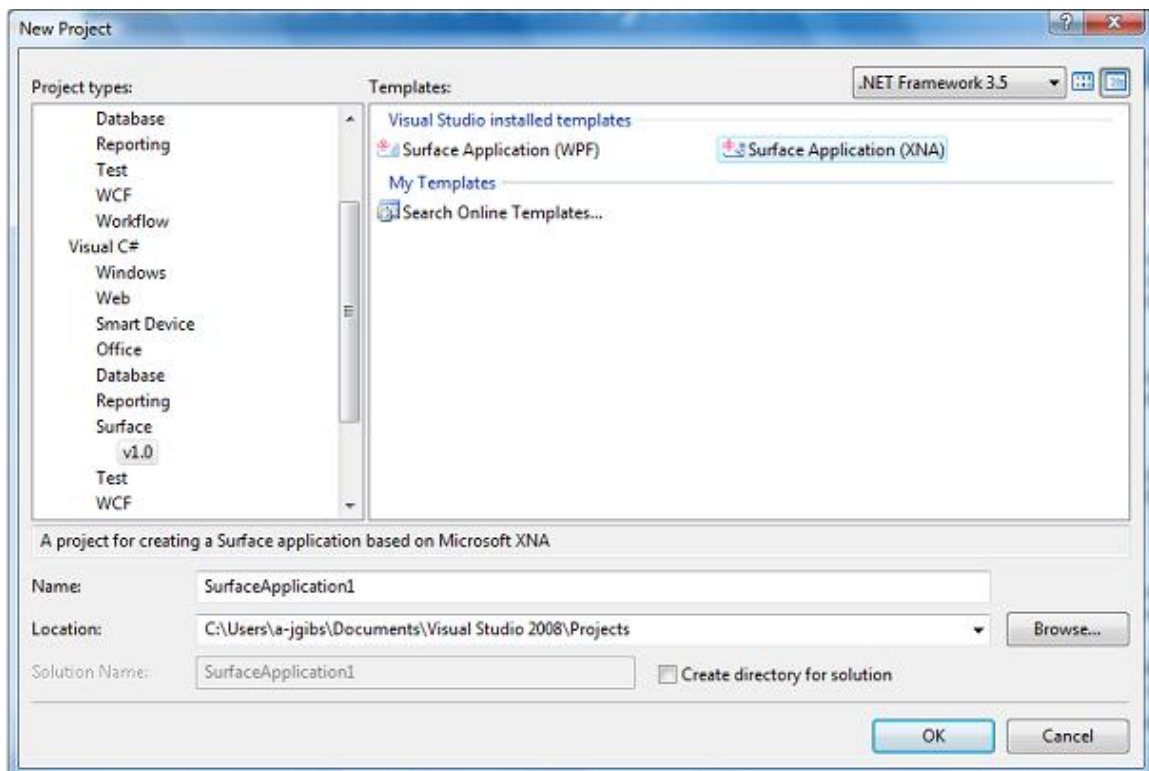
Software Development Kit

Starting the Visual Studio Project for XNA Applications

The Microsoft Surface SDK contains Microsoft XNA templates for Microsoft Visual C# 2008 (and Microsoft Visual Studio 2008) that you can use to create your Microsoft Surface application.

To start the Visual Studio project

1. Open Visual C# 2008 (or Visual Studio 2008).
2. On the **File** menu, click **New**.
3. Click **Project**, expand **Visual C#**, expand **Surface**, and then click **v1.0**. This project type contains the Microsoft Surface-specific project templates.
4. In the **Templates** pane, under **Visual Studio installed templates**, select **Surface Application (XNA)**.



5. Change the application name in the **Name** box, if you want to.
6. Click **OK**.
7. Learn [what the template creates for you](#).

© 2009 Microsoft Corporation. All rights reserved.

Microsoft Surface

Software Development Kit

What the XNA Template Creates for You

The Microsoft Surface XNA template for Microsoft Visual C# 2008 Express Edition (and Microsoft Visual Studio 2008) creates the basic code and framework that you must have to initialize and build a touch-enabled application for a Microsoft Surface unit or an application that will work with the [Surface Simulator](#) application.

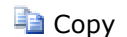
Template-Generated C# Code

If you have developed by using Microsoft XNA Game Studio, you should be familiar with the template-generated code because of its similarities to the XNA Game Studio-generated code. (All XNA Game Studio-generated methods are included in the Microsoft Surface XNA template-generated code). The Microsoft Surface XNA main type (App1) derives from the [Microsoft.Xna.Framework.Game](#) class.

The C# code that the Microsoft Surface template generates also includes the following elements:

- Includes Microsoft Surface Core and XNA **using** statements and their associated references.
- Includes identifiers and properties that App1 methods use to facilitate interaction between the Microsoft Surface and XNA frameworks, including:

C#



```
private readonly GraphicsDeviceManager graphics;
private ContactTarget contactTarget;
private UserOrientation currentOrientation = UserOrientation.Bottom;
private Color backgroundColor = new Color(81, 81, 81);
private bool applicationLoadCompleteSignalled;
private Matrix screenTransform = Matrix.Identity;

// The application state: Activated, Previewed, Deactivated,
// Start in the Activated state.
private bool isApplicationActivated = true;
private bool isApplicationPreviewed;

/// <summary>
/// The graphics device manager for the application.
/// </summary>
protected GraphicsDeviceManager Graphics
{
    get { return graphics; }
}

/// <summary>
/// The target that receives all Microsoft Surface input for the application.
/// </summary>
protected ContactTarget ContactTarget
{
    get { return contactTarget; }
}
```

- Overrides the **Initialize** method to include the following initialization duties:
 - Call the generated **SetWindowOnSurface** method that moves and sizes the window to cover the input surface.
 - Call the generated **InitializeSurfaceInput** method that initializes the Microsoft Surface Vision

System.

- Set the application's orientation based on the current Launcher orientation.
- Register Microsoft Surface activation events.
- Set up the user interface to transform if the user interface is rotated.
- Creates the **SetWindowOnSurface** method that is called during initialization to position the application window and make sure that the graphics device has the correct back buffer size.
- Creates the **InitializeSurfaceInput** method that is called during initialization to establish communication with the Microsoft Surface unit (or Surface Simulator).
- Builds application event handlers, including:
 - An event handler for the [ApplicationActivated](#) event.
 - An event handler for the [ApplicationPreviewed](#) event.
 - An event handler for the [ApplicationDeactivated](#) event.

You can manually add more event handlers to the C# code.

Template-Generated XML File

The template also generates an XML file that contains the information needed to install your application with the Surface Shell. This file is named from the project name such as "SurfaceApplication1.xml" or whatever name is provided when the project is created.

Next, [edit the C# file](#).

© 2009 Microsoft Corporation. All rights reserved.

Microsoft Surface

Software Development Kit

Editing the C# File for the XNA Template

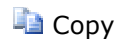
The Microsoft Surface XNA template adds code to the App1 class of the App1.cs file. To expand the App1 class into a "Hello World" application, this example uses a "Hello World" JPEG image file.

Note: This example does not use manipulations or inertia. To learn how to use those capabilities, see [Manipulating a Large Image by Using XNA](#) and [Manipulations and Inertia Using the Core Namespace Sample Application](#).

To add the logic to display the Hello World image file, make the following changes to the template code.

1. In **Solution Explorer**, open the App1.cs file.
2. Declare new App1 class identifiers (anywhere in the class private data area).

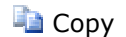
C#



```
private Texture2D helloWorldImage;  
private SpriteBatch spriteBatch;  
private Rectangle drawingArea;
```

3. Modify the **LoadContent** method to appear like the following code example.

C#

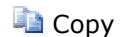


```
protected override void LoadContent()  
{  
    spriteBatch = new SpriteBatch(GraphicsDevice);  
    helloWorldImage =  
        Texture2D.FromFile(graphics.GraphicsDevice, @"Resources\HelloWorld.jpg");  
    drawingArea =  
        new Rectangle(graphics.GraphicsDevice.Viewport.X + 100,  
            graphics.GraphicsDevice.Viewport.Y + 100,  
            graphics.GraphicsDevice.Viewport.Width,  
            graphics.GraphicsDevice.Viewport.Height);  
}
```

Note: This example assumes that a HelloWorld.jpg file exists in the Resources directory off of the project executable directory (for example, bin\debug\Resources).

4. Add the following code to the **Draw** method.

C#



```
//TODO: Add your drawing code here.  
spriteBatch.Begin();  
Vector2 pos = new Vector2(drawingArea.Left, drawingArea.Top);  
spriteBatch.Draw(this.helloWorldImage, pos, Color.White);  
spriteBatch.End();
```

5. Optional. Add thread safety to the **Update** method:
 - a. Create a global class lock object.

C#



```
private object syncObject = new object();
```

- b. Wrap a lock block around the **Update** method code to prevent critical data from being modified while the **Update** method is using it. The modified **Update** method should look similar to the following code example.

C# Copy

```
protected override void Update(GameTime gameTime)
{
    lock (syncObject)
    {
        if (isApplicationActivated || isApplicationPreviewed)
        {
            if (isApplicationActivated)
            {
                // TODO: Process contacts.
                // Use the following code to get the state of all current contacts.
                // ReadOnlyContactCollection contacts = contactTarget.GetState();
            }
            // TODO: Add your update logic here.
        }
    } // Lock block end.

    base.Update(gameTime);
}
```

- c. Lock other critical areas of the application, but only where this lock is necessary because lock use can degrade application performance.

Next, [run and extend the application](#).

© 2009 Microsoft Corporation. All rights reserved.

Microsoft Surface

Software Development Kit

Running and Extending Your XNA Application

If you are running your application on a Microsoft Surface unit, double-click the **SurfaceInput** icon on the Windows desktop. If **SurfaceInput** is not running, your application will not receive **Contact** events. If you are running on a separate computer, start Surface Simulator from the Start menu. Surface Simulator starts **SurfaceInput** when it is launched.

The last step is to build your project by pressing the F5 key in Microsoft Visual C# 2008 Express Edition (or Microsoft Visual Studio 2008). This action also runs the application, which displays the "Hello, World" image in the Microsoft Surface window.

If you encounter an error during the build process, check to verify that all the elements and the attributes match the names that are contained in the Microsoft Surface SDK or inside corresponding C# constructs. Make sure that you have a "HelloWorld.jpg" image file and that it is in the appropriate folder (for example, debug\bin).

To extend this tutorial, try to including manipulations support (like the example in [Manipulating a Large Image by Using XNA](#)). You could further extend this tutorial by including manipulations and inertia support (like the sample application in [Manipulations and Inertia Using the Core Namespace Sample Application](#)).

© 2009 Microsoft Corporation. All rights reserved.