

Hand­s-on lab

Lab: Inking

September 2015

Contents

[Overview 3](#_Toc431304032)

[Exercise 1: Drawing with the InkCanvas 4](#_Toc431304033)

[Task 1 – Create a blank Universal Windows app 4](#_Toc431304034)

[Task 2 – Create the InkCanvas 6](#_Toc431304035)

[Task 3 – Customize the drawing attributes 9](#_Toc431304036)

[Task 4 –Erase and Clear 12](#_Toc431304037)

[Task 5 – Save and Load 14](#_Toc431304038)

[Exercise 2: Handwriting Recognition 16](#_Toc431304039)

[Task 1 – Create a new solution for the Handwriting Recognition project 16](#_Toc431304040)

[Task 2 – Create the View 17](#_Toc431304041)

[Task 3 – Set up the ink recognizer 19](#_Toc431304042)

[Task 4 – Implement OnRecognizeAsync() 22](#_Toc431304043)

[Summary 25](#_Toc431304044)

Overview

Inking provides powerful tools to translate natural movement into pixels. As more devices become touch-enabled, inking plays a great role in user interactions. Traditionally, the stylus has been the main input device for ink. In UWP, stylus, touch, and mouse, and Xbox controller inking can be implemented with just a few lines of code. The InkCanvas has simplified the work that goes into creating the ink experience in your app.

In this lab, you will create an InkCanvas that accepts mouse, touch and stylus inputs. You will customize the ink color on the fly and erase, clear, save, and load strokes. In Exercise 2, you will implement handwriting recognition on an InkCanvas.

# Objectives

* 1. This lab will show you how to:
  + Create an InkCanvas in a UWP app
  + Enable stylus, touch and mouse inputs
  + Change the ink color based on user selection
  + Erase single strokes
  + Clear the canvas
  + Save and load strokes
  + Recognize handwriting

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Drawing with the InkCanvas
  3. Handwriting Recognition
  4. Estimated time to complete this lab:  **45 to 60 minutes**.

Exercise 1: Drawing with the InkCanvas

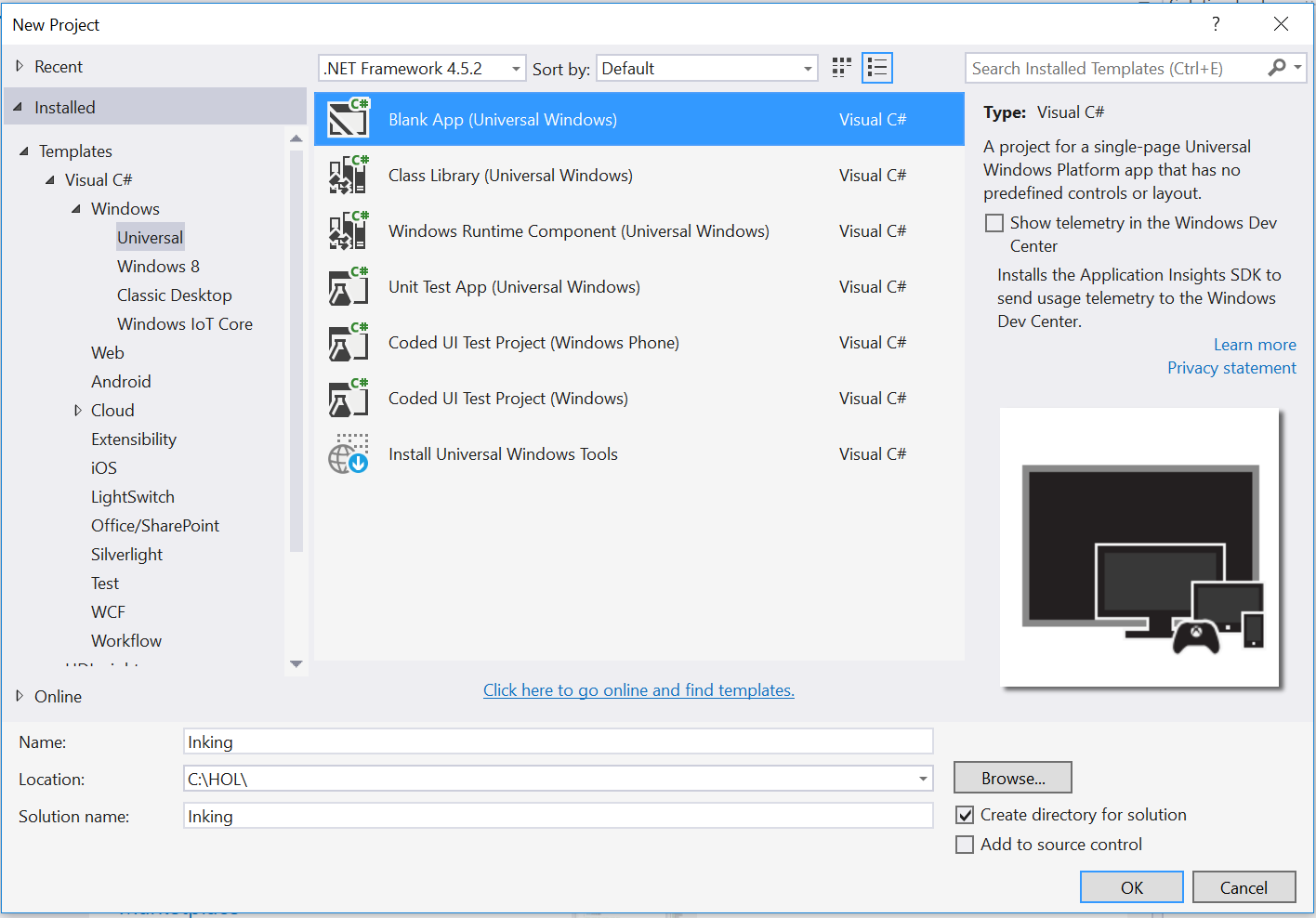
1. The InkCanvas is a great way to quickly implement ink in your UWP app. In this exercise, you will create an InkCanvas and modify its drawing attributes to produce strokes of different colors. You will also save, load, and erase strokes and clear the canvas.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > Visual C# > Windows > Universal** and select the **Blank App (Universal Windows)** template.
2. Name your project **Inking** and select the file system location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.

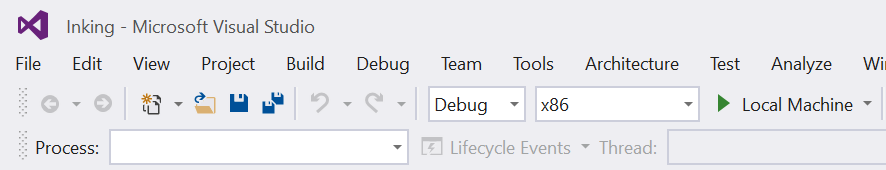
Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect both **Add to source control** and **Show telemetry in the Windows Dev Center** if you don't wish to version your work or use Application Insights. Click **OK** to create the project.



Figure

Create a new Blank App project in Visual Studio 2015.

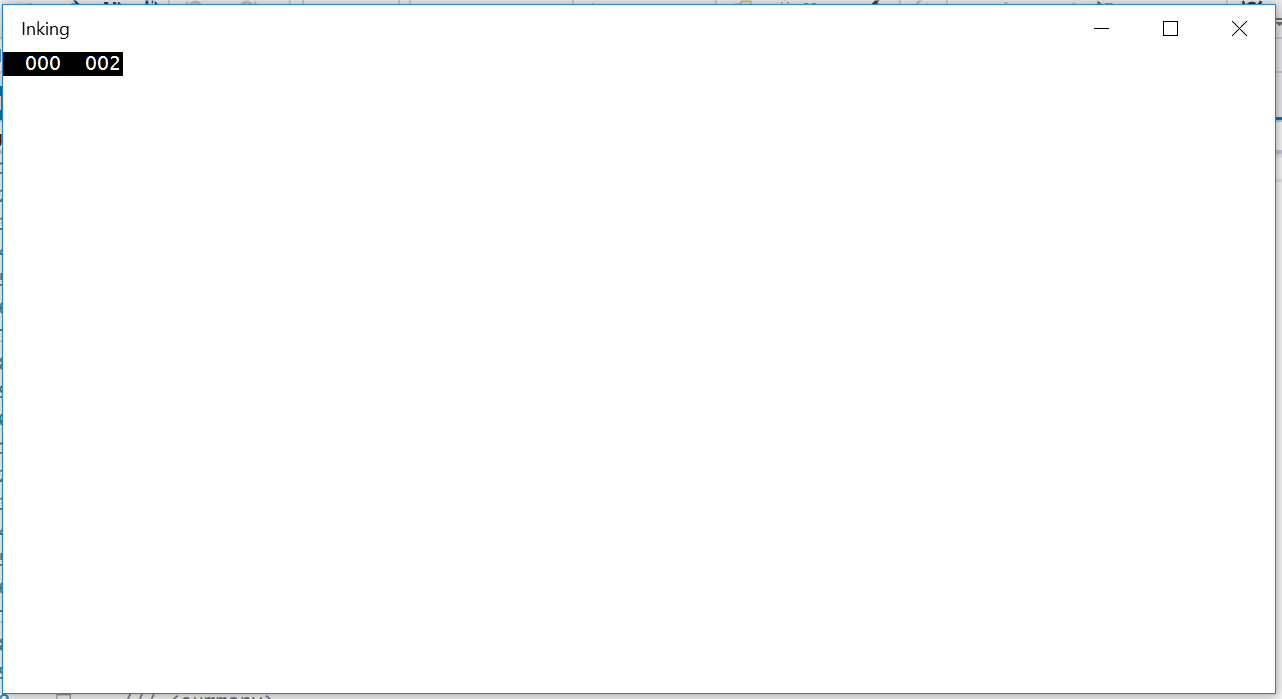
1. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.



Figure

* + 1. Configure your app to run on the Local Machine.

1. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.



Figure

The blank universal app running in Desktop mode.

* 1. **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
  2. In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

1. Return to Visual Studio and stop debugging.

Task 2 – Create the InkCanvas

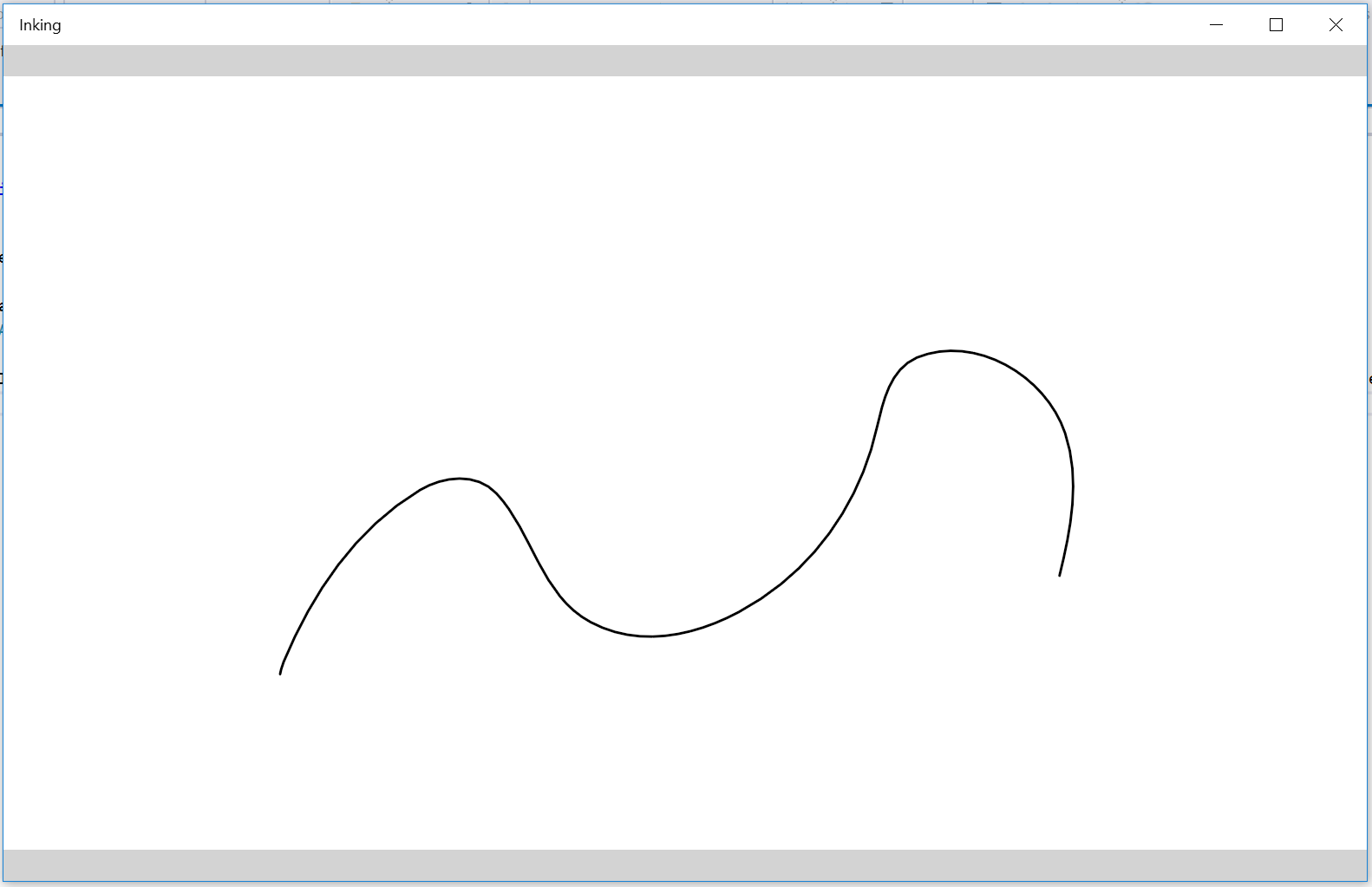
* 1. You can begin inking in your app in a few lines of code. In this task, we will set up a layout structure to support a large ink canvas with a rows above and below it to hold options and style buttons.

1. Open **MainPage.xaml**. Set the **Grid** background to **LightGray** and add **RowDefinitions** for three rows. The top and bottom rows will contain style options and buttons, and the large middle row will hold the InkCanvas.
   * 1. XAML
   1. <Grid Background="LightGray">
   2. <Grid.RowDefinitions>
   3. <RowDefinition Height="Auto" />
   4. <RowDefinition Height="2\*" />
   5. <RowDefinition Height="Auto" />
   6. </Grid.RowDefinitions>
   7. </Grid>
2. Add layouts to the three rows. The first row will contain two columns, each holding a horizontal StackPanel. The second row will contain the **InkCanvas**, and the third row will consist of another StackPanel. Name your InkCanvas with an **x:Name** attribute.
   * 1. XAML
   1. <Grid Background="LightGray">
   2. <Grid.RowDefinitions>
   3. <RowDefinition Height="Auto" />
   4. <RowDefinition Height="2\*" />
   5. <RowDefinition Height="Auto" />

</Grid.RowDefinitions>

* 1. <Grid Grid.Row="0" Margin="12">
  2. <Grid.ColumnDefinitions>
  3. <ColumnDefinition Width="1\*" />
  4. <ColumnDefinition Width="1\*" />
  5. </Grid.ColumnDefinitions>
  6. <StackPanel Grid.Column="0" Orientation="Horizontal">
  7. </StackPanel>
  8. <StackPanel Grid.Column="1" Orientation="Horizontal" HorizontalAlignment="Right">
  9. </StackPanel>
  10. </Grid>
  11. <Grid Grid.Row="1" Background="White" Grid.ColumnSpan="2">
  12. <InkCanvas x:Name="InkCanvas" />
  13. </Grid>
  14. <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
  15. </StackPanel>
  16. </Grid>

1. At this point, your app is ready for simple inking with a stylus. In case your development machine doesn’t support a stylus, we will enable other input devices as well.
2. In the MainPage code-behind, add the **Windows.UI.Input.Inking** namespace.
   1. **Note:** The InkCanvas only accepts stylus input by default. You can add Mouse and Touch as additional input device types.
3. Add **Mouse** and **Touch** as input device types for the **InkCanvas**.
   * 1. C#
   1. public MainPage()
   2. {
   3. this.InitializeComponent();
   4. InkCanvas.InkPresenter.InputDeviceTypes =
   5. Windows.UI.Core.CoreInputDeviceTypes.Mouse |   
       Windows.UI.Core.CoreInputDeviceTypes.Pen |   
       Windows.UI.Core.CoreInputDeviceTypes.Touch;
   6. }
   7. Note: The InkCanvas is a control that contains one inking property – the InkPresenter. The InkPresenter displays ink on the canvas and handles attributes for the strokes.
4. Build and run your app. Draw on the white **InkCanvas** with your mouse, stylus, or with touch. You will see the ink show with the default color and pen size attributes.



Figure

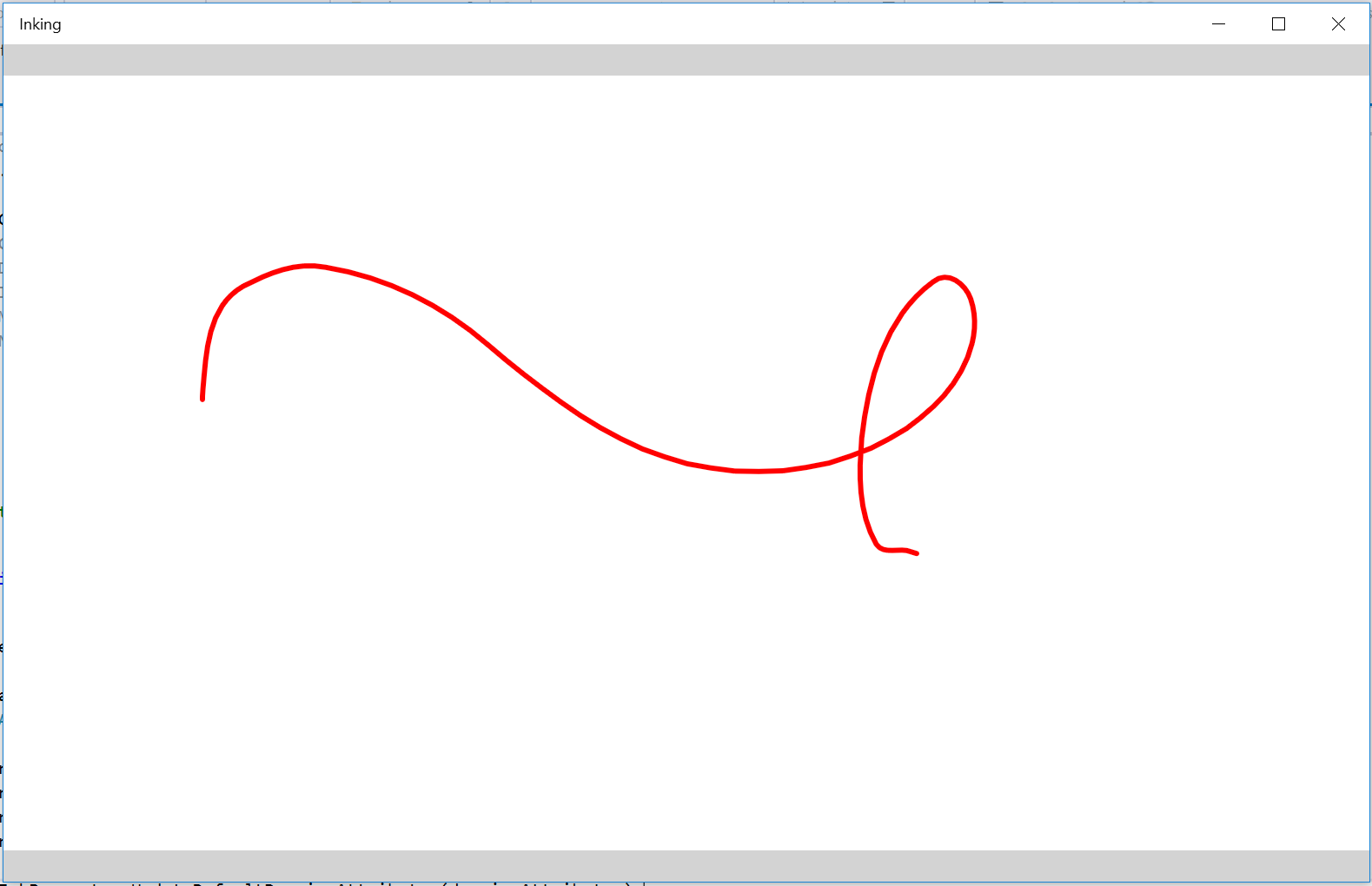
The default drawing attributes for the InkCanvas.

1. Stop debugging and return to Visual Studio.

Task 3 – Customize the drawing attributes

If you would like to use a single ink style, you can customize the drawing attributes for the **InkCanvas** when you create the **InkDrawingAttributes** instance. You can also programmatically update the attributes based on user selections. In this task, you will first add default attributes to the ink style, then add buttons to change the attributes on the fly.

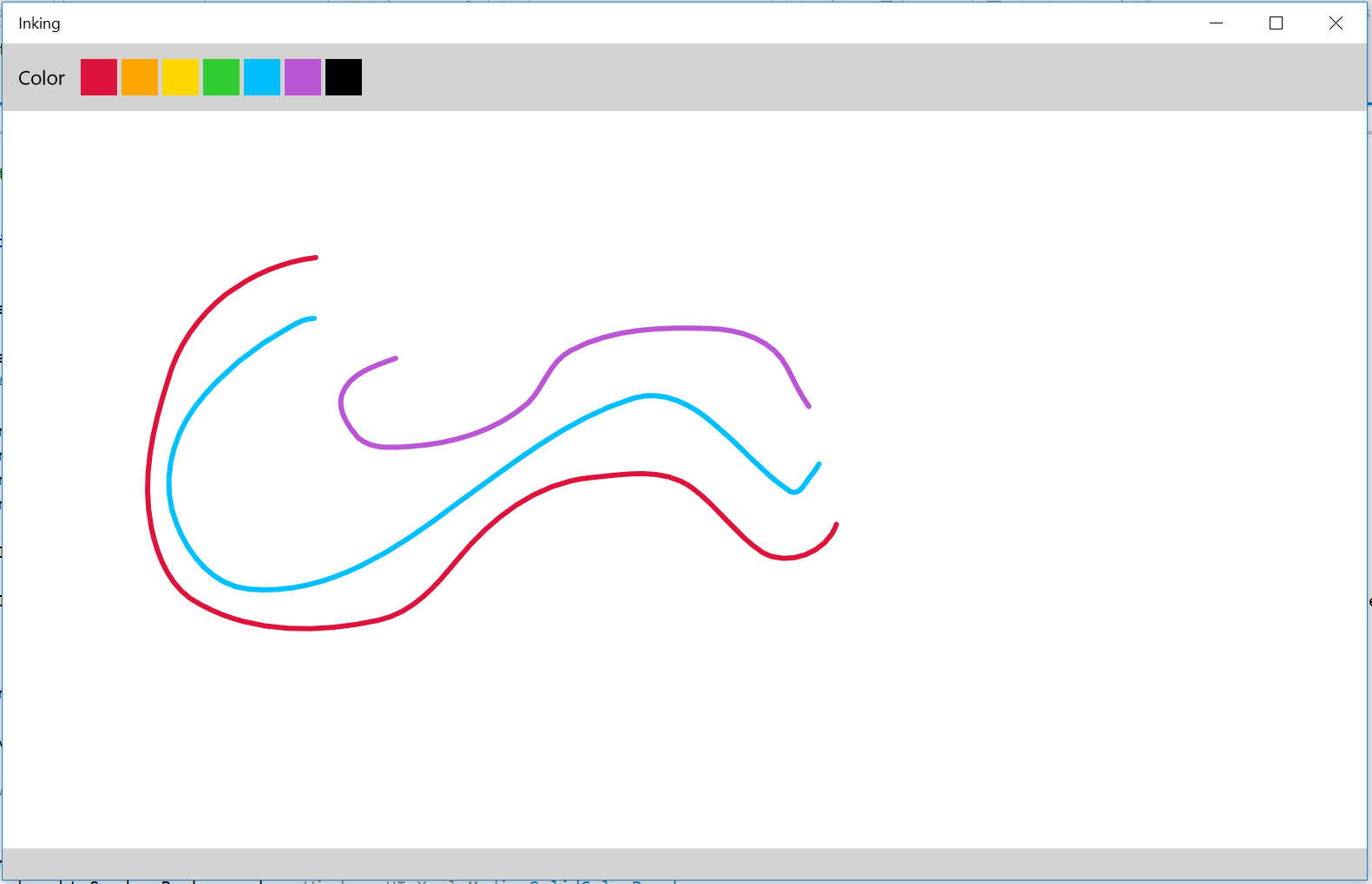
1. You can set attributes immediately when you create an instance of **InkDrawingAttributes**. Add **Color**, **Size**, **Pressure**, and **FitToCurve** attributes in the **MainPage** constructor. After setting the attributes, update the drawing attributes on the InkCanvas by passing in your instance of **drawingAttributes**.
   * 1. C#
   1. public MainPage()
   2. {
   3. this.InitializeComponent();
   4. InkDrawingAttributes drawingAttributes = new InkDrawingAttributes();
   6. drawingAttributes.Color = Windows.UI.Colors.Red;
   7. drawingAttributes.Size = new Size(4, 4);
   8. drawingAttributes.IgnorePressure = false;
   9. drawingAttributes.FitToCurve = true;
   10. InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes);
   11. InkCanvas.InkPresenter.InputDeviceTypes =
   12. Windows.UI.Core.CoreInputDeviceTypes.Mouse |   
        Windows.UI.Core.CoreInputDeviceTypes.Pen |   
        Windows.UI.Core.CoreInputDeviceTypes.Touch;
   13. }
   14. **Note:** Although IgnorePressure = false is an option in the drawing attributes, the **InkCanvas** does not currently display pressure-sensitive strokes. Instead, the pressure value at each point is recorded.
   15. Other drawing attributes include PenTip and DrawAsHighlighter. For more on the **InkDrawingAttributes** class, visit <https://msdn.microsoft.com/en-us/library/windows.ui.input.inking.inkdrawingattributes.aspx>
2. Build and run your app. This time, your ink will display a wider stroke in red.



Figure

Wider ink stroke in red.

1. Stop debugging and return to Visual Studio.
2. Add a set of colored buttons inside the StackPanel that is on the first row of the MainPage grid. Set the Click event for each to **OnPenColorChanged** and the style to **StaticResource** **ColorButtonStyle**. You will create the event handlers and the ColorButtonStyle in the next steps.
   * 1. XAML
   1. <StackPanel Grid.Column="0" Orientation="Horizontal">
   2. <TextBlock Text="Color" Margin="0,0,12,0" VerticalAlignment="Center" />
   3. <Button Background="Crimson" Foreground="Crimson" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   4. <Button Background="Orange" Foreground="Orange" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   5. <Button Background="Gold" Foreground="Gold" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   6. <Button Background="LimeGreen" Foreground="LimeGreen" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   7. <Button Background="DeepSkyBlue" Foreground="DeepSkyBlue" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   8. <Button Background="MediumOrchid" Foreground="MediumOrchid" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   9. <Button Background="Black" Foreground="Black" Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
   10. </StackPanel>
3. Add styles in a **Grid.Resources** element to set the height, width, margin, font size, and padding for the buttons.
   * 1. XAML
   1. <Grid Background="LightGray">
   2. <Grid.Resources>
   3. <Style x:Key="ColorButtonStyle" TargetType="Button">
   4. <Setter Property="MinWidth" Value="28"/>
   5. <Setter Property="MinHeight" Value="28"/>
   6. <Setter Property="Width" Value="28"/>
   7. <Setter Property="Height" Value="28"/>
   8. <Setter Property="Margin" Value="0,0,0,0"/>
   9. <Setter Property="FontSize" Value="0"/>
   10. <Setter Property="Padding" Value="0,0,0,0"/>
   11. </Style>
   12. </Grid.Resources>
4. In the code-behind, add the **OnPenColorChanged** event handler. This method copies the default drawing attributes from the InkCanvas and sets the ink color to the background color of the button.
   * 1. C#
   1. private void OnPenColorChanged(object sender, RoutedEventArgs e)
   2. {
   3. if (InkCanvas != null)
   4. {
   5. InkDrawingAttributes drawingAttributes = InkCanvas.InkPresenter.CopyDefaultDrawingAttributes();
   6. // Use button's background to set new pen's color
   7. var btnSender = sender as Button;
   8. var brush = btnSender.Background as SolidColorBrush;
   9. drawingAttributes.Color = brush.Color;
   10. InkCanvas.InkPresenter  
        .UpdateDefaultDrawingAttributes(drawingAttributes);
   11. }
   12. }
5. Build and run your app. Use the color buttons to change the ink color as you draw.



Figure

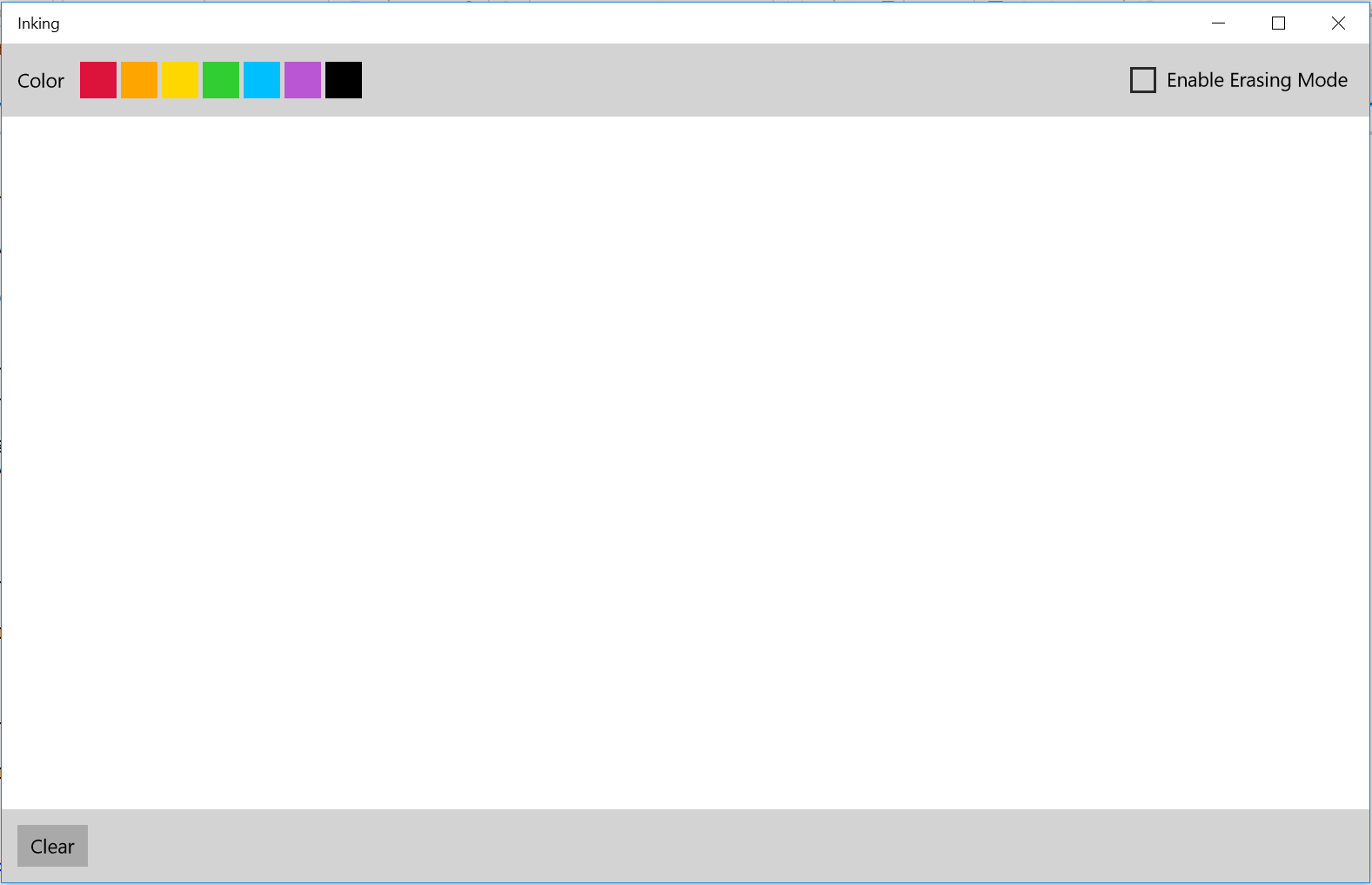
Change the ink color to user-selected colors.

1. Stop debugging and return to Visual Studio.
   1. **Note:** The InkToolbar UI used in the Edge Browser Web Notes feature is available to you as a Visual Studio extension. The toolbar provides a color picker, pen tip choices, and ink, highlighter, and eraser modes out of the box. For more information on the Ink Toolbar, visit <https://visualstudiogallery.msdn.microsoft.com/58194dfe-df44-4c4e-893a-1eca40675269>

Task 4 –Erase and Clear

Now that you can customize the strokes you are drawing, you may wish to erase errors. You can provide options to eliminate single strokes or clear the entire canvas at once.

1. In **MainPage.xaml**, add a checkbox inside the **StackPanel** that is in the second column of the main grid. This checkbox will appear to the right of the color buttons to toggle erasing mode. Set the **Checked** event to ErasingModeCheckBox\_Checked and the Unchecked event to ErasingModeCheckBox\_Unchecked. You will create the handlers in the next step.
   * 1. XAML
   1. <StackPanel Grid.Column="1" Orientation="Horizontal" HorizontalAlignment="Right">
   2. <CheckBox Content="Enable Erasing Mode" Margin="20,0,4,0"   
       Checked="ErasingModeCheckBox\_Checked"   
       Unchecked="ErasingModeCheckBox\_Unchecked"/>
   3. </StackPanel>
2. Create the handlers for the checked and unchecked states in the code-behind. The checked state sets the **InputProcessingConfiguration**.Mode to **Erasing**.
   * 1. C#
   1. private void ErasingModeCheckBox\_Checked(object sender, RoutedEventArgs e)
   2. {
   3. InkCanvas.InkPresenter.InputProcessingConfiguration.Mode =   
       InkInputProcessingMode.Erasing;
   4. }
   5. private void ErasingModeCheckBox\_Unchecked(object sender, RoutedEventArgs e)
   6. {
   7. InkCanvas.InkPresenter.InputProcessingConfiguration.Mode =   
       InkInputProcessingMode.Inking;
   8. }
   9. **Note:** A stroke made in Erasing mode will erase any Ink stroke it intersects. The entire stroke will be erased.
3. Build and run your app. Draw while in Inking mode, then enable Erasing mode with the checkbox. Intersect one of your strokes to see it disappear.
4. Stop debugging and return to Visual Studio.
5. In **MainPage.xaml**, add a **Clear** button to the **StackPanel** in the third row. Set the **Click** event to **OnClear**. You will create the event handler in the next step.
   * 1. XAML
   1. <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
   2. <Button Content="Clear" Click="OnClear" />
   3. </StackPanel>
6. Add the **OnClear()** handler to the **MainPage** code-behind.
   * 1. C#
   1. void OnClear(object sender, RoutedEventArgs e)
   2. {
   3. InkCanvas.InkPresenter.StrokeContainer.Clear();
   4. }
7. Build and run your app. Draw a few strokes and use the **Clear** button to remove them from the InkCanvas in one step.



Figure

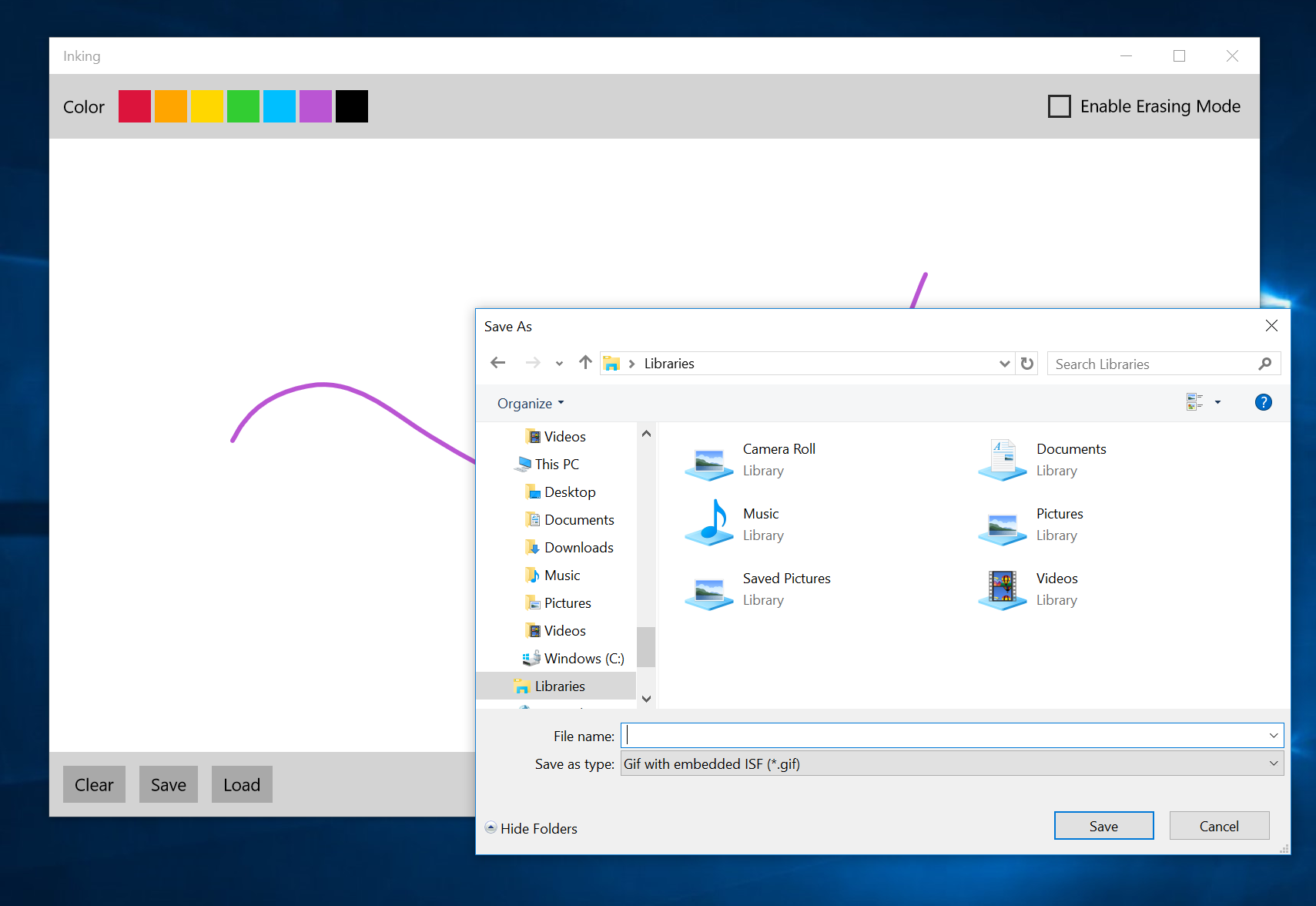
Erasing mode allows you remove single strokes, while the Clear button removes all.

1. Stop debugging and return to Visual Studio.

Task 5 – Save and Load

Inking is useful because it allows you translate physical interactions into a pixel representation on the device. To reuse the information created, you can save strokes from an InkCanvas and reload them later. In this task, you will save your ink strokes to a GIF file and reload them onto the canvas.

1. Add **Save** and **Load** buttons next to the **Clear** button in MainPage.xaml. Set the click events to **OnSaveAsync** and **OnLoadAsync**, respectively.
   * 1. XAML
   1. <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
   2. <Button Content="Clear" Click="OnClear" />
   3. <Button Content="Save" Margin="12,0,0,0" Click="OnSaveAsync"/>
   4. <Button Content="Load" Margin="12,0,0,0" Click="OnLoadAsync"/>
   5. </StackPanel>
2. In the MainPage code-behind, add the **Windows.Storage** and **Windows.Storage.Streams** namespaces.
   * 1. C#
   1. using Windows.Storage;
   2. using Windows.Storage.Streams;
3. Add the **OnSaveAsync** and **OnLoadAsync** methods to the code-behind. The **OnSaveAsync** method will save the strokes as a GIF file. The load method will read those strokes back onto the InkCanvas.
   * 1. C#
     2. async void OnSaveAsync(object sender, RoutedEventArgs e)
     3. {
     4. // We don't want to save an empty file
     5. if (InkCanvas.InkPresenter.StrokeContainer.GetStrokes().Count > 0)
     6. {
     7. var savePicker = new Windows.Storage.Pickers.FileSavePicker();
     8. savePicker.SuggestedStartLocation =   
         Windows.Storage.Pickers.PickerLocationId.PicturesLibrary;
     9. savePicker.FileTypeChoices.Add(  
         "Gif with embedded ISF",   
         new System.Collections.Generic.List<string> { ".gif" });
     10. Windows.Storage.StorageFile file =   
          await savePicker.PickSaveFileAsync();
     11. if (null != file)
     12. {
     13. using (IRandomAccessStream stream =   
          await file.OpenAsync(FileAccessMode.ReadWrite))
     14. {
     15. await InkCanvas.InkPresenter.StrokeContainer.SaveAsync(  
          stream);
     16. }
     17. }
     18. }
     19. }
     20. async void OnLoadAsync(object sender, RoutedEventArgs e)
     21. {
     22. var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
     23. openPicker.SuggestedStartLocation =   
          Windows.Storage.Pickers.PickerLocationId.PicturesLibrary;
     24. openPicker.FileTypeFilter.Add(".gif");
     25. openPicker.FileTypeFilter.Add(".isf");
     26. Windows.Storage.StorageFile file = await openPicker.PickSingleFileAsync();
     27. if (null != file)
     28. {
     29. using (var stream = await file.OpenSequentialReadAsync())
     30. {
     31. await InkCanvas.InkPresenter.StrokeContainer.LoadAsync(stream);
     32. }
     33. }
     34. }
4. Build and run your app. Draw on the canvas and save your strokes to a GIF file on your file system. With the app still running, clear the canvas and use the **Load** button to open the file browser. Select your GIF file to reload the strokes back onto the InkCanvas.



Figure

Save your strokes to a GIF file.

1. Stop debugging and return to Visual Studio.

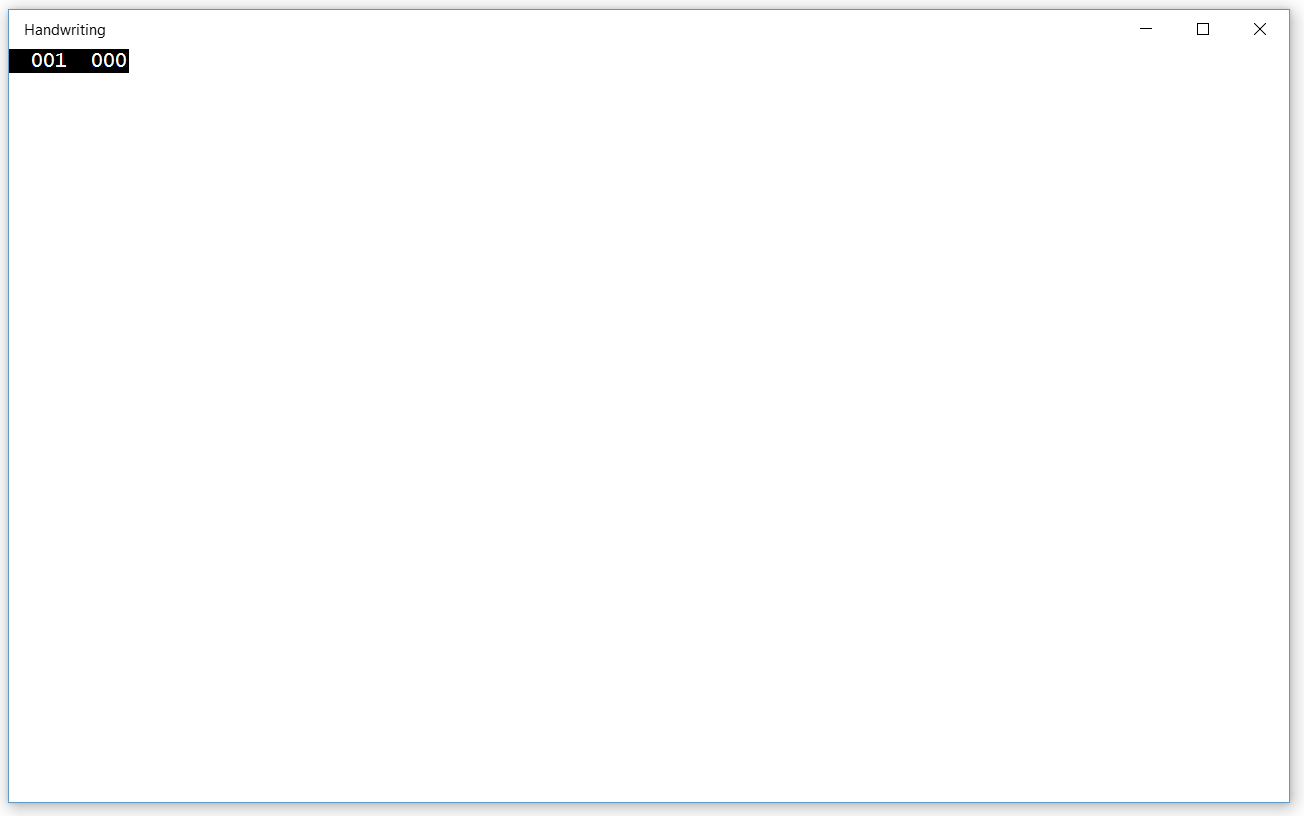
Exercise 2: Handwriting Recognition

* 1. Applications for Ink include drawing, annotation of images, and handwriting recognition. In this exercise, we will explore Handwriting recognition with an InkCanvas using the language-specific recognizers available on your device.

Task 1 – Create a new solution for the Handwriting Recognition project

Create a new project for Handwriting Recognition.

1. In a new instance of Visual Studio 2015, choose **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > Visual C# > Windows > Universal** and select the **Blank App (Universal Windows)** template. Name the project **Handwriting**. Save the project to the folder where you store your Hands-on Labs.
2. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.
3. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.



Figure

The blank universal app running in Desktop mode.

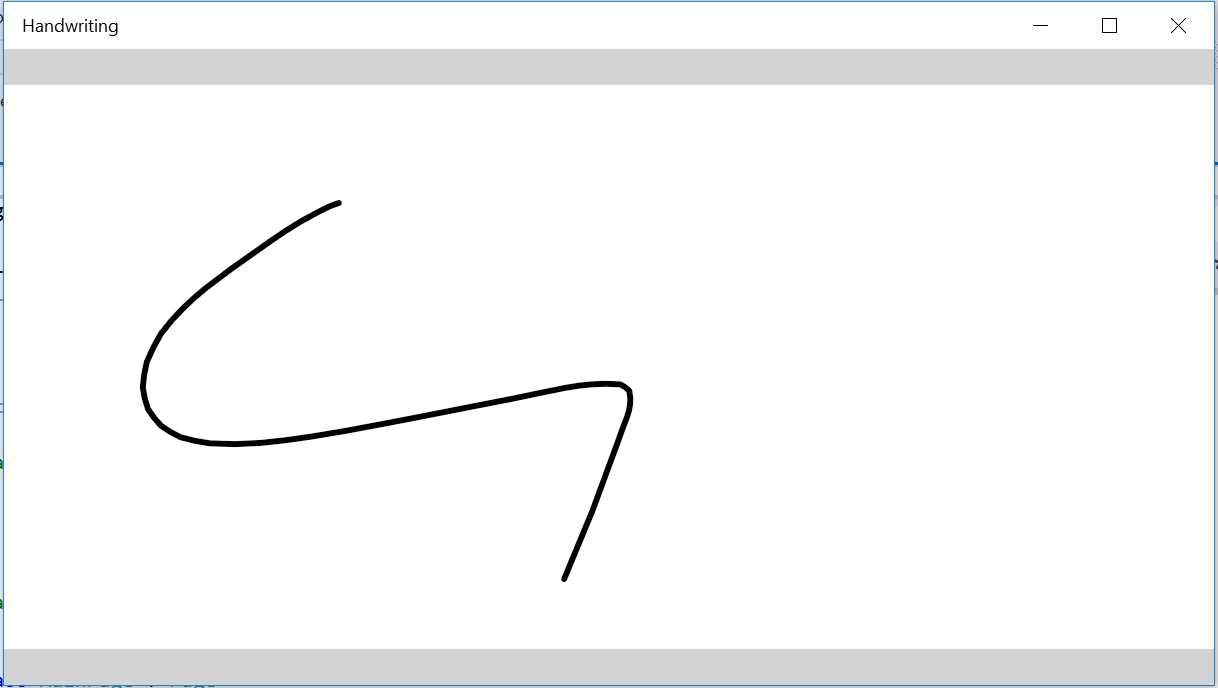
* 1. **Note:** The preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of this lab, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

1. Return to Visual Studio and stop debugging.

Task 2 – Create the View

The MainPage for this exercise will have a similar layout as the project in Exercise 1. In this task, we will create the layout and set up the InkCanvas.

1. Change the Grid background to LightGray. Create the same layout that you used in Exercise 1 using row definitions, column definitions, and layout controls. Add an InkCanvas to the second row with **x:Name InkCanvas**.
   * 1. XAML
   1. <Grid Background="LightGray">
   2. <Grid.RowDefinitions>
   3. <RowDefinition Height="Auto" />
   4. <RowDefinition Height="2\*" />
   5. <RowDefinition Height="Auto" />
   6. </Grid.RowDefinitions>
   7. <Grid Grid.Row="0" Margin="12">
   8. <Grid.ColumnDefinitions>
   9. <ColumnDefinition Width="2\*" />
   10. <ColumnDefinition Width="1\*" />
   11. </Grid.ColumnDefinitions>
   12. <StackPanel Orientation="Horizontal" Grid.Column="0">
   13. </StackPanel>
   14. <StackPanel Grid.Column="1" HorizontalAlignment="Right"   
        Orientation="Horizontal">
   15. </StackPanel>
   16. </Grid>
   17. <Grid Grid.Row="1" Background="White" Grid.ColumnSpan="2">
   18. <InkCanvas x:Name="InkCanvas" />
   19. </Grid>
   20. <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
   21. </StackPanel>
   22. </Grid>
2. Add the **Windows.UI.Input.Inking** namespace to the code-behind.
   * 1. C#
   1. using Windows.UI.Input.Inking;
3. Initialize the drawing attributes in the MainPage constructor.
   * 1. C#
   1. public MainPage()
   2. {
   3. this.InitializeComponent();
   4. InkDrawingAttributes drawingAttributes = new InkDrawingAttributes();
   5. drawingAttributes.Color = Windows.UI.Colors.Black;
   6. drawingAttributes.Size = new Size(4, 4);
   7. drawingAttributes.IgnorePressure = false;
   8. drawingAttributes.FitToCurve = true;
   9. InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(  
       drawingAttributes);
   10. InkCanvas.InkPresenter.InputDeviceTypes =   
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |   
        Windows.UI.Core.CoreInputDeviceTypes.Pen |   
        Windows.UI.Core.CoreInputDeviceTypes.Touch;
   11. }
4. Build and run your app. You will see simple inking capabilities similar to the previous exercise.



Figure

The start of the Handwriting recognition app.

1. Stop debugging and return to Visual Studio.

Task 3 – Set up the ink recognizer

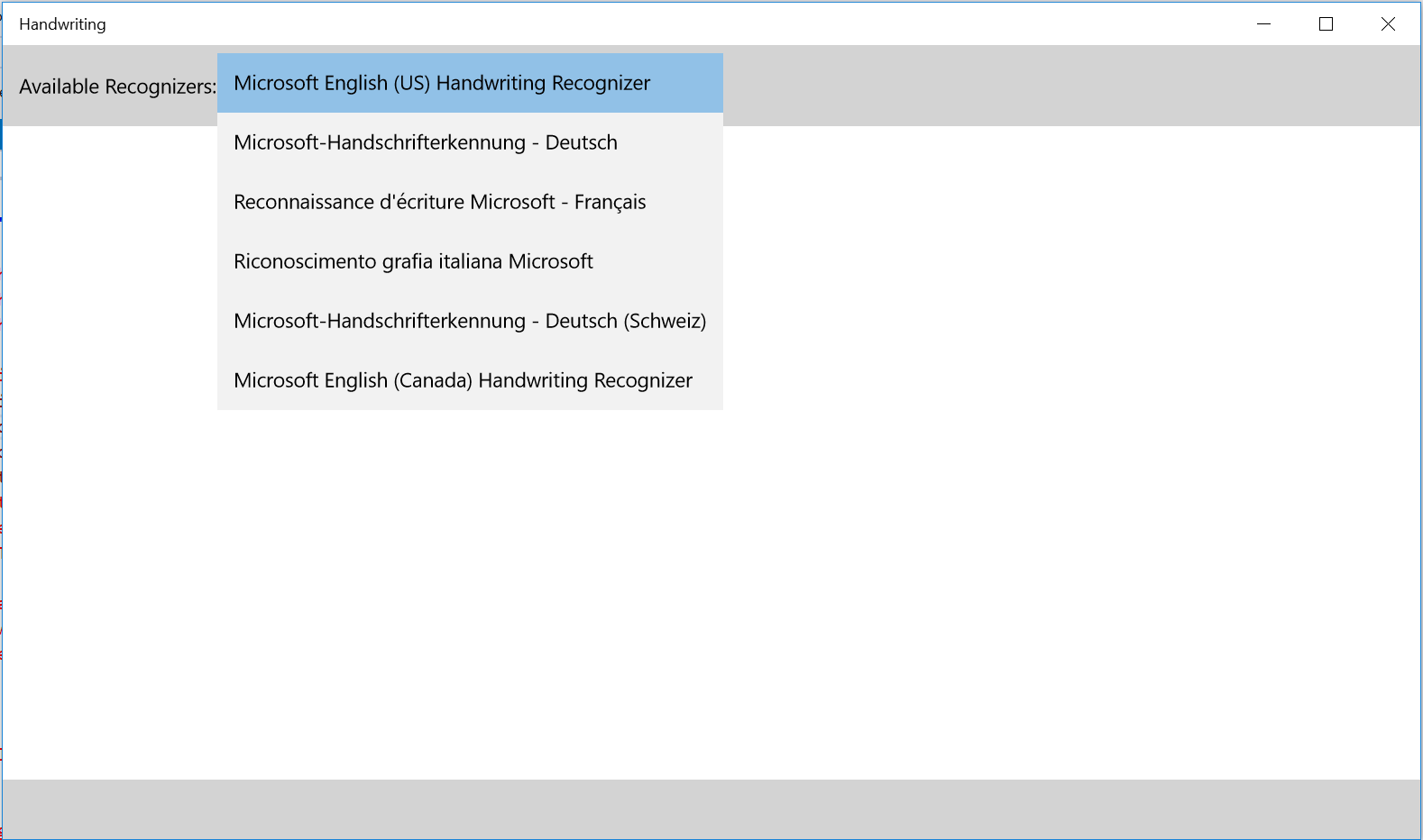
* 1. The InkRecognizer handles all aspects of handwriting recognition. In this task, you will set up a RecognizerHelper dictionary and the InkRecognizer.

1. Right-click on the **Handwriting** project name and choose **Add > Existing Item**. Browse to the Hands-on labs **Lab Assets** folder and add **RecognizerHelper.cs**.
2. Open **RecognizerHelper.cs**. This helper provides a dictionary of recognizers that may be installed on your device. It translates the recognizer language tags into a longer, human-readable names to help users identify which recognizer they have selected. Save and close the helper.
3. Add a TextBlock label and a ComboBox to the **StackPanel** in row 0, column 0 of MainPage.xaml.
   * 1. XAML
   1. <StackPanel Orientation="Horizontal" Grid.Column="0">
   2. <TextBlock Text="Available Recognizers:" Margin="0,8"/>

<ComboBox  
 x:Name="RecoName"  
 MaxWidth="500"  
 SelectionChanged="OnRecognizerChanged">  
 </ComboBox>

* 1. </StackPanel>

1. Create a **Status** TextBlock in the third row. The status text will display handwriting recognition results or a message if no recognizers were found on the system.
   * 1. XAML
   1. <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
   2. <TextBlock x:Name="Status" Margin="20,0,0,0" />
   3. </StackPanel>
2. Add the **Windows.Globalization** **namespace** to the MainPage code-behind.
   * 1. C#
   1. using Windows.Globalization;
3. Create an instances of the **InkRecognizerContainer** and a read-only list of **InkRecognizers** in the code-behind.
   * 1. C#
   1. namespace Handwriting
   2. {
   3. /// <summary>
   4. /// An empty page that can be used on its own or navigated to within a Frame.
   5. /// </summary>
   6. public sealed partial class MainPage : Page
   7. {
   8. InkRecognizerContainer inkRecognizerContainer = null;
   9. private IReadOnlyList<InkRecognizer> recoView = null;
4. Load the available system recognizers in the MainPage constructor.
   * 1. C#
   1. public MainPage()
   2. {
   3. this.InitializeComponent();
   4. InkDrawingAttributes drawingAttributes = new
   5. InkDrawingAttributes();
   6. drawingAttributes.Color = Windows.UI.Colors.Black;
   7. drawingAttributes.Size = new Size(4, 4);
   8. drawingAttributes.IgnorePressure = false;
   9. drawingAttributes.FitToCurve = true;
   10. inkRecognizerContainer = new InkRecognizerContainer();
   11. recoView = inkRecognizerContainer.GetRecognizers();
   12. if (recoView.Count > 0)
   13. {
   14. foreach (InkRecognizer recognizer in recoView)
   15. {
   16. RecoName.Items.Add(recognizer.Name);
   17. }
   18. }
   19. else
   20. {
   21. RecoName.IsEnabled = false;
   22. RecoName.Items.Add("No Recognizer Available");
   23. }
   24. RecoName.SelectedIndex = 0;
   25. InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(  
        drawingAttributes);
   26. InkCanvas.InkPresenter.InputDeviceTypes =   
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |   
        Windows.UI.Core.CoreInputDeviceTypes.Pen |   
        Windows.UI.Core.CoreInputDeviceTypes.Touch;
   27. }
5. Create the **OnRecognizerChanged** handler to set the ComboBox value to the selected recognizer, and the **SetRecognizerByName** method to keep track of which recognizers were found on the system.
   * 1. C#
   1. void OnRecognizerChanged(object sender, RoutedEventArgs e)
   2. {
   3. string selectedValue = (string)RecoName.SelectedValue;
   4. SetRecognizerByName(selectedValue);
   5. }
6. bool SetRecognizerByName(string recognizerName)
7. {
8. bool recognizerFound = false;
9. foreach (InkRecognizer reco in recoView)
10. {
11. if (recognizerName == reco.Name)
12. {
13. inkRecognizerContainer.SetDefaultRecognizer(reco);
14. recognizerFound = true;
15. break;
16. }
17. }
18. if (!recognizerFound)
19. {
20. Status.Text = "Could not find target recognizer.";
21. }
22. return recognizerFound;
23. }
24. Build and run your app. Use the ComboBox to view the list of recognizers installed on your system.



Figure

The ComboBox offers recognizers that are available on the device.

1. Stop debugging and return to Visual Studio.

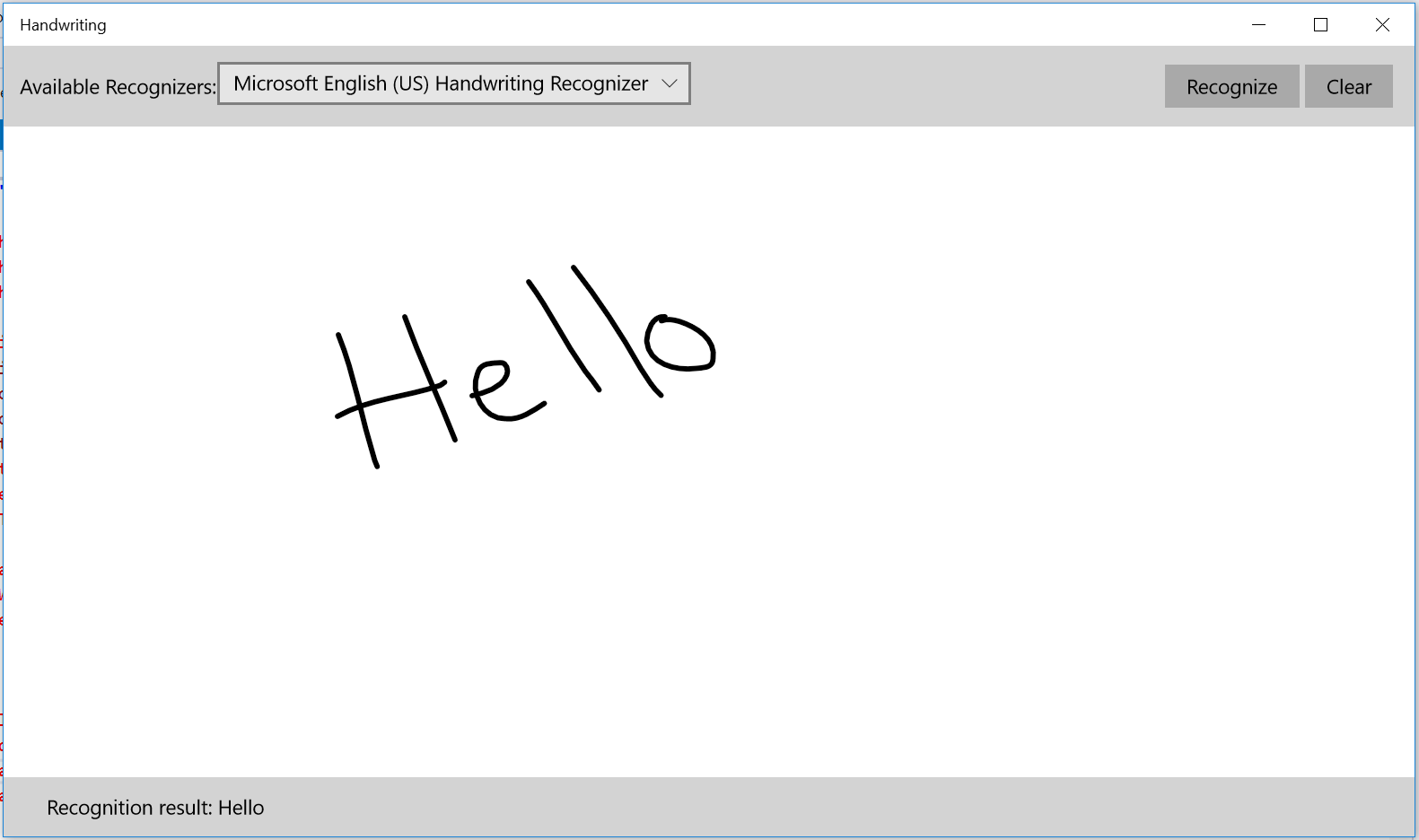
Task 4 – Implement OnRecognizeAsync()

Now that the recognizers are set up, you can implement handwriting recognition in your app.

1. Add Recognize and Clear buttons to the second column of the first row in the MainPage view.
   * 1. XAML
   1. <StackPanel Grid.Column="1" HorizontalAlignment="Right" Orientation="Horizontal">
   2. <Button x:Name="RecognizeBtn" Content="Recognize" Width="100"   
       Margin="0,0,4,0" Click="OnRecognizeAsync"/>
   3. <Button x:Name="ClearBtn" Content="Clear" Width="65" Margin="0,0,4,0"   
       Click="OnClear"/>
   4. </StackPanel>
2. Add the **OnClear()** handler to the MainPage code-behind. This method is identical to the **OnClear**() method we used in Exercise 1.
   * 1. C#
   1. void OnClear(object sender, RoutedEventArgs e)
   2. {
   3. InkCanvas.InkPresenter.StrokeContainer.Clear();
   4. }
3. Add the **OnRecognizeAsync()** handler to the code-behind. This method awaits results based on the strokes on the InkCanvas and the selected recognizer. If there are results, they will be displayed in the **Status** text block. If the results are inconclusive, the **Status** will report **No text recognized**.
   * 1. C#
   1. async void OnRecognizeAsync(object sender, RoutedEventArgs e)
   2. {
   3. IReadOnlyList<InkStroke> currentStrokes =   
       InkCanvas.InkPresenter.StrokeContainer.GetStrokes();
   4. if (currentStrokes.Count > 0)
   5. {
   6. RecognizeBtn.IsEnabled = false;
   7. ClearBtn.IsEnabled = false;
   8. RecoName.IsEnabled = false;
   9. var recognitionResults = await inkRecognizerContainer.RecognizeAsync(  
       InkCanvas.InkPresenter.StrokeContainer,   
       InkRecognitionTarget.All);
   10. if (recognitionResults.Count > 0)
   11. {
   12. // Display recognition result
   13. string str = "Recognition result:";
   14. foreach (var r in recognitionResults)
   15. {
   16. str += " " + r.GetTextCandidates()[0];
   17. }
   18. Status.Text=str;
   19. }
   20. else
   21. {
   22. Status.Text = "No text recognized.";
   23. }
   24. RecognizeBtn.IsEnabled = true;
   25. ClearBtn.IsEnabled = true;
   26. RecoName.IsEnabled = true;
   27. }
   28. else
   29. {
   30. Status.Text="Must first write something.";
   31. }

}

1. Build and run your app. Choose a recognizer from the list and write a word from that language on the canvas. Use the **Recognize** button to view your results.



Figure

Successful handwriting recognition.

1. Stop debugging and return to Visual Studio.

Summary

* 1. Ink is a natural and convenient way to interact with apps on Windows 10 devices that have mouse, touch, pen, or controller inputs. The implementation of ink is quick and easy in UWP apps. In this lab, we covered options for customizing the ink attributes and styles, clearing and erasing the canvas, and saving and loading strokes. We also implemented handwriting recognition for language packs installed on the device.