**Contoso Jobs**

**Lab 2. Adding Ink**

In this Lab you will add Ink support to the Job to enable the Surface Hub stylus to Ink directly in a Job being created.

To work with this Lab you will need a device that has an Active Stylus, either a Surface Hub, a Surface Pro, or a Surface Book would work, along with numerous other tablet devices that support active stylus input, as opposed to touch with a projected capacitive pen.

This Lab continues from the previous Lab, or you can start from the start point in the folder named *2. Start Contoso Jobs Inking*

1. Open the existing **Contoso Jobs.sln** solution file in **Visual Studio 2015**
2. In Solution Explorer expand the **Views** folder and open the **CreateJobContentDialog.xaml** file.
3. Find the **TextBox** named *descriptionTextBox* and place it in a **Grid.** Add an **InkCanvas** control below the **TextBox** in the **Grid**

Increase the Height of the **TextBox** to 150 to provide more space for writing or drawing.

The **InkCanvas** also has the **IsHitTestVisable** attribute set to **False.** This means that it will not receive touch events, yet it will continue to receive input from an active stylus.

<Grid>

<TextBox Name="descriptionTextBox" Header="Description" TextWrapping="Wrap"

Height="150" />

<InkCanvas Name="Annotation" Height="150" IsHitTestVisible="False" />

</Grid>

1. Build and Run the App (F5). When you create a new job you can Ink inside the description. You will need a device with an Active Stylus for this to work.

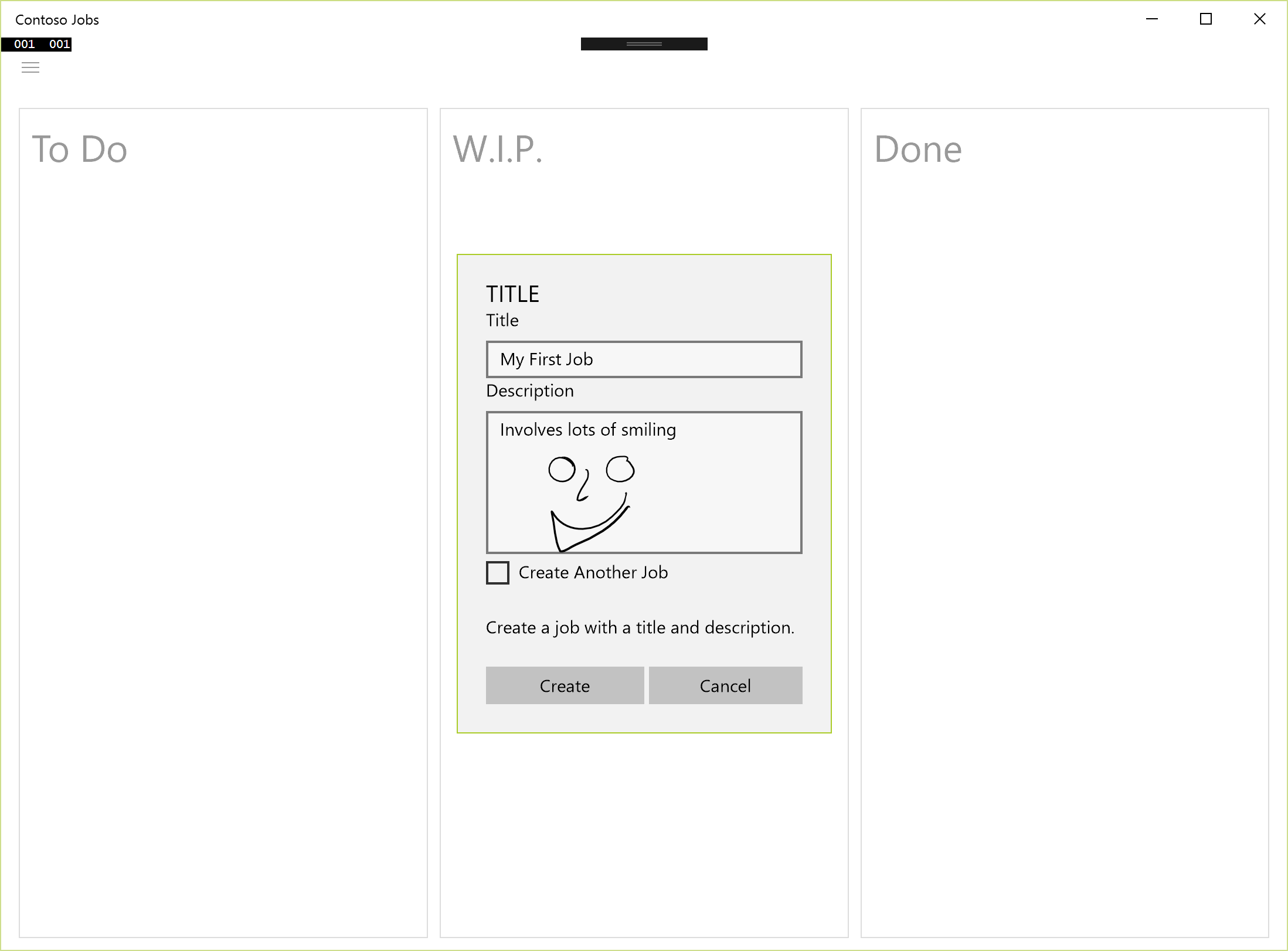


Figure Ink in a new Job

1. Once you create the Job the Ink is lost. The Job class needs to maintain a copy of the Ink strokes so they can be displayed where the Job is rendered.

Open the **Job.cs** code file in the **Models** folder and add a new property named **Strokes** to contain a collection of **InkStrokes**

Note: the **XmlIgnore** attribute is used to prevent the Strokes from being serialized, you will discover more about this later in this Lab.

[XmlIgnore]

public IReadOnlyList<InkStroke> Strokes { get; set; }

you will also need to add two namepsaces to the file for this to compile

using System.Xml.Serialization;

using Windows.UI.Input.Inking;

1. When the Job is created the Ink Strokes need to be added to the Job object. Open the **CreateJobContentDialog.xaml.cs** file and edit the **CreateButtonClick** event handler.

private void ContentDialog\_CreateButtonClick(ContentDialog sender,

ContentDialogButtonClickEventArgs args)

{

IReadOnlyList<InkStroke> strokes =

Annotation.InkPresenter.StrokeContainer.GetStrokes();

jobsViewModel.CreateJob(titleTextBox.Text, descriptionTextBox.Text,

strokes.ToList());

CreateAnother = createAnotherCheckBox.IsChecked.Value;

}

You will need to add a using statement for the Inking namespace.

using Windows.UI.Input.Inking;

1. You need to update the **CreateJob** method in the **JobsViewModel** class to take the collection of InkStrokes and use it to initialize a Job object.

internal void CreateJob(string title, string description, List<InkStroke> strokes)

{

Job j = new Job() { Title = title, Description = description,

Status = JobStatus.Backlog, Strokes= strokes };

jobs.Add(j);

SaveJobs();

OnPropertyChanged("Backlog");

}

You will need to add a using statement for the Inking namespace to the JobsViewModel.cs file.

using Windows.UI.Input.Inking;

1. Edit the XAML in **JobControl.xaml** to add an **InkCanvas** to display the **InkStrokes** from the Job object. Add the DescriptionTextBox TextBlock into a **Grid** and then add an **InkCanvas** after it. This is similar to the changes you made to the CreateJobContentDialog

<Grid Grid.Row="1">

<TextBlock x:Name="DescriptionTextBlock" Margin="0" TextWrapping="Wrap"

Text="{Binding Description}" VerticalAlignment="Center"

Height="150"

FontSize="{StaticResource TextFontSize}"/>

<InkCanvas x:Name="Annotation" Height="150" />

</Grid>

1. Open the **JobControl.xaml.cs** code file and edit the **DataContextChanged** method to redraw the InkStrokes

private void JobControl\_DataContextChanged(FrameworkElement sender,

DataContextChangedEventArgs args)

{

Job j = DataContext as Job;

if (j != null)

{

switch (j.Status)

{

case JobStatus.Backlog:

ProgressButtonText = "\uE768";

break;

case JobStatus.WIP:

ProgressButtonText = "\uE71A";

break;

default:

ProgressButtonText = string.Empty;

break;

}

Annotation.InkPresenter.StrokeContainer.Clear();

if (j.Strokes != null)

{

foreach (InkStroke stroke in j.Strokes)

{

Annotation.InkPresenter.StrokeContainer.AddStroke(stroke.Clone());

}

}

}

}

You will need to add a using statement for the Inking namespace.

using Windows.UI.Input.Inking;

1. Build and run the App (F5). You will see that you can create a Job with ink and then when it is moved around the Ink is maintained.

The Ink annotations are not saved between sessions though, remember you excluded the InkStroke collection from the XmlSerialization.

1. To add serialization of the strokes implement the IXmlSerializable interface on the Job class. Open the Job.cs file and add the following code to load and save a Job object to Xml

using System.Xml;

using System.Xml.Schema;

using System.Xml.Serialization;  
using Windows.Storage.Streams;

public class Job : IXmlSerializable  
{

…

public XmlSchema GetSchema()

{

return null;

}

public async void ReadXml(XmlReader reader)

{

reader.Read();

Title = reader.ReadElementContentAsString("Title", "");

Description = reader.ReadElementContentAsString("Description", "");

string s = reader.ReadElementContentAsString("Status", "");

Status = (JobStatus)Enum.Parse(typeof(JobStatus), s);

InMemoryRandomAccessStream ms = new InMemoryRandomAccessStream();

DataWriter dw = new DataWriter(ms.GetOutputStreamAt(0));

byte[] tempBytes = new byte[1024];

int bytesRead = 0;

do

{

bytesRead = reader.ReadElementContentAsBinHex(tempBytes, 0, 1024);

if (bytesRead > 0)

{

dw.WriteBytes(tempBytes);

}

} while (bytesRead == 1024);

await dw.StoreAsync();

if (ms.Size > 0)

{

InkStrokeContainer inkCont = new InkStrokeContainer();

await inkCont.LoadAsync(ms);

Strokes = inkCont.GetStrokes().ToList();

}

reader.Skip();

}

public async void WriteXml(XmlWriter writer)

{

writer.WriteElementString("Title", Title);

writer.WriteElementString("Description", Description);

writer.WriteElementString("Status", Status.ToString());

writer.WriteStartElement("Stokes");

if (Strokes != null && Strokes.Count > 0)

{

using (InMemoryRandomAccessStream ms = new InMemoryRandomAccessStream())

{

InkStrokeContainer inkCont = new InkStrokeContainer();

foreach (InkStroke stroke in Strokes)

{

inkCont.AddStroke(stroke.Clone());

}

await inkCont.SaveAsync(ms);

await ms.FlushAsync();

byte[] bytes = new byte[ms.Size];

//var dataWriter = new DataWriter(ms);

var reader = new DataReader(ms.GetInputStreamAt(0));

await reader.LoadAsync((uint)ms.Size);

reader.ReadBytes(bytes);

writer.WriteBinHex(bytes, 0, (int)ms.Size);

}

}

writer.WriteEndElement();

}

1. In order to test these changes you will need to delete any previously saved Jobs.xml file as it is no longer in the format expected. You will find the file somewhere like this C:\Users\<username>\AppData\Local\Packages\55d006ef-be06-4019-bc6d-ec38f28a5304\_xn317nnjv2k9e\LocalState\Jobs.xml

(the highlighted text will be different in your environment)

1. Build and Run the App (**F5**). Create a new Job and add Ink and move it around. Close the App and reopen it, the Ink should be saved between sessions.
2. You might have noticed that you can now add Ink annotations to the Job objects directly, while they are in the column, yet those annotations are not saved. Try it now, if you have not already.
3. Open the **JobControl.xaml.cs** file to add code to capture annotations made to a Job while it is in place. Add an event handler for the **StrokesCollected** event in the constructor.

public JobControl()

{

this.InitializeComponent();

DataContextChanged += JobControl\_DataContextChanged;

Annotation.InkPresenter.StrokesCollected += InkPresenter\_StrokesCollected;

}

private void InkPresenter\_StrokesCollected(InkPresenter sender,

InkStrokesCollectedEventArgs args)

{

Job j = DataContext as Job;

if (j != null)

{

IReadOnlyList<InkStroke> strokes =

Annotation.InkPresenter.StrokeContainer.GetStrokes();

j.Strokes = strokes.ToList();

}

}

1. Build and Run the App (F5). You can add Ink annotations to the JobControl in place and they are captured and saved in the Job object.

1. All the Ink drawn is currently black, in order to support different colors you will now add a color selection. Start by adding a red colored **Ellipse** to the **JobControl.xaml** file.

<Grid Grid.Row="1">

<TextBlock x:Name="DescriptionTextBlock" Margin="0" TextWrapping="Wrap"

Text="{Binding Description}" VerticalAlignment="Center"

Height="150" FontSize="{StaticResource TextFontSize}"/>

<InkCanvas x:Name="Annotation" Height="150" />

<Ellipse Width="20" Height="20" VerticalAlignment="Top" HorizontalAlignment="Right"

Fill="Red" Margin="5, 0" PointerPressed="Ellipse\_PointerPressed" />

</Grid>

1. In the JobControlxaml.cs code file add the event handler code for the Ellipse PointerPressed event.

private void Ellipse\_PointerPressed(object sender, PointerRoutedEventArgs e)

{

InkDrawingAttributes drawingAttribs =

Annotation.InkPresenter.CopyDefaultDrawingAttributes();

drawingAttribs.Color = Colors.Red;

Annotation.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttribs);

e.Handled = true;

}

1. Build and Run the App (F5). You can now change the color of the ink you add.
2. Add two more colored Ellipse objects for Black and Blue selections to change the Ink color.

In this Lab you have added Ink annotations to the JobControl objects, allowing multiple people to Ink in different Jobs on the screen at the same time. The use of the stylus enables a far richer way of expressing yourself within the Job description.

Extra exercises:

1. If you are using the Windows 10 Anniversary Update then try the InkToolbar
2. The Edit Job dialog still doesn’t support Ink, you could fix that. Why should you not fix it ?