Revit API UI Labs

Lab 1 – Ribbon

May, 2011 by A. Nagy

Last Updated, Date : March 03, 2018

<C#>C# Version**</C#>**

Objective: In this lab, we will learn how to create our own Ribbon tab and panels then populate them with controls. We'll learn how to:

- Create a new Ribbon tab and panel
- Create various Ribbon controls

The following is the breakdown of step by step instructions in this lab:

1. Create a new External Application
2. Create new Ribbon tab and panel
3. Create push button
4. Create split button
5. Create combo box
6. Create other controls
7. Summary

1. Define A New External Application

We'll create a new project and add an external application to it.

1.1 You can follow the same steps as in Revit API Intro Labs >> Lab1 >> 5, but this time name the Class Library project UiVb (or UiCs depending on the programming language you chose)

1.2 Rename the Class1.vb (or cs) file

- File name: **1_Ribbon.vb (or .cs)**
- Application class name: **UIRibbon**

Required References:
Additional references needed for this lab are:

- PresentationCore – used for handling bitmap images
- WindowsBase – used for handling bitmap images
- IntroCs/Vb – we will be using commands defined in the Intro labs.

Required Namespaces:

Namespaces needed for this lab are:

- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.ApplicationServices
- Autodesk.Revit.Attributes
- System.Collections.Generic
- System.Xaml
- System.Diagnostics – used for debug
- System.IO – used for reading folders
- System.Windows.Media.Imaging – used for bitmap images

1.3 Declare some variables that will contain certain paths that will be useful e.g. to retrieve images for our ribbon controls:

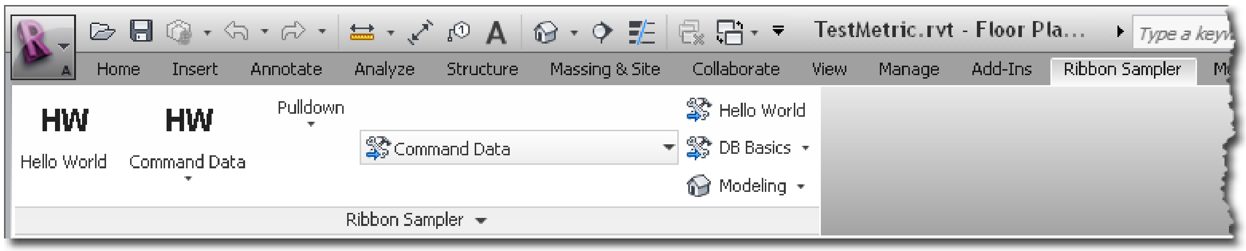
<C#>
```
    /// <summary>
    /// This is both the assembly name and the namespace
    /// of the external command provider.
    /// </summary>

    const string _introLabName = "IntroCs";
    const string _uiLabName = "UiCs";
    const string _dllExtension = ".dll";

    /// <summary>
    /// Name of subdirectory containing images.
    /// </summary>

    const string _imageFolderName = "Images";

    /// <summary>
    /// Location of managed dll where we have defined the commands.
    /// </summary>

    string _introLabPath;

    /// <summary>
    /// Location of images for icons.
    /// </summary>

    string _imageFolder;
```
</C#>

1.4 Let's also add a couple of utility functions, which will help us find our image folder or create a BitmapImage object from a specific image file

<C#>
```
    /// <summary>
    /// Starting at the given directory, search upwards for
    /// a subdirectory with the given target name located
    /// in some parent directory.
    /// </summary>
    /// <param name="path">Starting directory, e.g.
GetDirectoryName( GetExecutingAssembly().Location ).</param>
    /// <param name="target">Target subdirectory name, e.g. "Images".</param>
    /// <returns>The full path of the target directory if found, else
null.</returns>

    string FindFolderInParents( string path, string target )
    {
      Debug.Assert( Directory.Exists( path ),
        "expected an existing directory to start search in" );

      string s;

      do {
        s = Path.Combine( path, target );
        if( Directory.Exists( s ) )
        {
          return s;
        }
        path = Path.GetDirectoryName( path );
      } while( null != path );

      return null;
    }

    /// <summary>
    /// Load a new icon bitmap from our image folder.
    /// </summary>

    BitmapImage NewBitmapImage(string imageName)
    {
      return new BitmapImage(new Uri(
        Path.Combine(_imageFolder, imageName)));
    }
```
</C#>

We'll also need to copy the 'Images' folder from the 'Labs\2_Revit_UI_API' folder to our project's folder.

2. Create new Ribbon tab and panel



Since Revit 2012 we can also add our own Ribbon tab using CreateRibbonTab(). Depending on which override of CreateRibbonPanel() we are using, the newly created panel will either be added to the Ribbon tab we specify or to the default 'Add-Ins' tab.
Let's create a function that will create our Ribbon tab and add a panel to it.

<C#>

```
    public void AddRibbonSampler( UIControlledApplication app )
    {
      app.CreateRibbonTab( "Ribbon Sampler" );

      RibbonPanel panel =
        app.CreateRibbonPanel( "Ribbon Sampler", "Ribbon Sampler" );
    }
```

</C#>

Before going any further let's initialize the helper variables in the class inside the OnStartup() function.

<C#>

```
      // External application directory:

      string dir = Path.GetDirectoryName(
        System.Reflection.Assembly
        .GetExecutingAssembly().Location );

      // External command path:

      _introLabPath = Path.Combine( dir, _introLabName + _dllExtension );

      if( !File.Exists( _introLabPath ) )
      {
        TaskDialog.Show( "UIRibbon", "External command assembly not found: "
          + _introLabPath );
        return Result.Failed;
      }

      // Image path:

      _imageFolder = FindFolderInParents( dir, _imageFolderName );

      if( null == _imageFolder
        || !Directory.Exists( _imageFolder ) )
      {
        TaskDialog.Show(
          "UIRibbon",
          string.Format(
            "No image folder named '{0}' found in the parent directories of
'{1}'.",
          _imageFolderName, dir ) );

        return Result.Failed;
      }
```

</C#>

3. Create push button

Now let's add some controls to the Ribbon panel we've just created. First we'll create a push button. Controls can be created through the various [Control]Data classes which will specify the properties of the control we are creating and the actual [Control] will be created once we called AddItem() – e.g. you use PushButtonData when creating a PushButton.
In case of the various buttons we need to specify the assembly path and full class name of the ExternalCommand that should be invoked when the button is clicked.
Note: If you specify an external command in the *.addin manifest file then the command will be available through the 'Add-Ins tab >> External >> External Tools'. By creating various Ribbon controls you can make your commands available through those controls as well. If the latter is enough for you then no need to declare your external commands in the *.addin manifest file.

```
<C#>
    public void AddPushButton(RibbonPanel panel)
    {
      // Set the information about the command we will be assigning
      // to the button

      PushButtonData pushButtonDataHello =
        new PushButtonData( "PushButtonHello", "Hello World",
                            _introLabPath, _introLabName + ".HelloWorld" );

      // Add a button to the panel

      PushButton pushButtonHello =
        panel.AddItem( pushButtonDataHello ) as PushButton;

      // Add an icon
      // Make sure you reference WindowsBase and PresentationCore,
      // and import System.Windows.Media.Imaging namespace.

      pushButtonHello.LargeImage = NewBitmapImage( "ImgHelloWorld.png" );

      // Add a tooltip

      pushButtonHello.ToolTip = "simple push button";
    }
</C#>
```

4. Create split button

A split button is basically just a button that groups push buttons together. So you'll need to use PushButtonData's and add them to a SplitButtonData object to create the control. Here as well, each button will be hooked up to a specific external command.

```
<C#>
    public void AddSplitButton(RibbonPanel panel)
    {
      // Create three push buttons for split button drop down
      // #1
      PushButtonData pushButtonData1 =
        new PushButtonData( "SplitCommandData", "Command Data",
          _introLabPath, _introLabName + ".CommandData" );
      pushButtonData1.LargeImage = NewBitmapImage( "ImgHelloWorld.png" );

      // #2
      PushButtonData pushButtonData2 =
        new PushButtonData( "SplitDbElement", "DB Element",
          _introLabPath, _introLabName + ".DbElement" );
      pushButtonData2.LargeImage = NewBitmapImage( "ImgHelloWorld.png" );

      // #3
      PushButtonData pushButtonData3 =
        new PushButtonData( "SplitElementFiltering", "ElementFiltering",
          _introLabPath, _introLabName + ".ElementFiltering" );
      pushButtonData3.LargeImage = NewBitmapImage( "ImgHelloWorld.png" );

      // Make a split button now
      SplitButtonData splitBtnData =
        new SplitButtonData( "SplitButton", "Split Button" );

      SplitButton splitBtn = panel.AddItem( splitBtnData ) as SplitButton;
      splitBtn.AddPushButton( pushButtonData1 );
      splitBtn.AddPushButton( pushButtonData2 );
      splitBtn.AddPushButton( pushButtonData3 );
    }
</C#>
```

5. Create combo box

In case of e.g. the combo box, the functionality comes from handling selection events instead of hooking up each combo item to a specific external command. Once we created the combo box, we need to subscribe to the CurrentChanged event to handle selection changes.

```
<C#>
    public void AddComboBox( RibbonPanel panel )
    {
      // Create five combo box members with two groups

      // #1
      ComboBoxMemberData comboBoxMemberData1 =
        new ComboBoxMemberData( "ComboCommandData", "Command Data" );
      comboBoxMemberData1.Image = NewBitmapImage( "Basics.ico" );
      comboBoxMemberData1.GroupName = "DB Basics";

      // #2
      ComboBoxMemberData comboBoxMemberData2 =
        new ComboBoxMemberData( "ComboDbElement", "DB Element" );
      comboBoxMemberData2.Image = NewBitmapImage( "Basics.ico" );
      comboBoxMemberData2.GroupName = "DB Basics";

      // #3
      ComboBoxMemberData comboBoxMemberData3 =
        new ComboBoxMemberData( "ComboElementFiltering", "Filtering" );
      comboBoxMemberData3.Image = NewBitmapImage( "Basics.ico" );
      comboBoxMemberData3.GroupName = "DB Basics";

      // #4
      ComboBoxMemberData comboBoxMemberData4 =
        new ComboBoxMemberData( "ComboElementModification", "Modify" );
      comboBoxMemberData4.Image = NewBitmapImage( "Basics.ico" );
      comboBoxMemberData4.GroupName = "Modeling";

      // #5
      ComboBoxMemberData comboBoxMemberData5 =
        new ComboBoxMemberData("ComboModelCreation", "Create");
      comboBoxMemberData5.Image = NewBitmapImage("Basics.ico");
      comboBoxMemberData5.GroupName = "Modeling";

      // Make a combo box now
      ComboBoxData comboBxData = new ComboBoxData("ComboBox");
      ComboBox comboBx = panel.AddItem(comboBxData) as ComboBox;
      comboBx.ToolTip = "Select an Option";
      comboBx.LongDescription = "select a command you want to run";
      comboBx.AddItem(comboBoxMemberData1);
      comboBx.AddItem(comboBoxMemberData2);
      comboBx.AddItem(comboBoxMemberData3);
      comboBx.AddItem(comboBoxMemberData4);
      comboBx.AddItem(comboBoxMemberData5);

      comboBx.CurrentChanged += new
EventHandler<Autodesk.Revit.UI.Events.ComboBoxCurrentChangedEventArgs>(comboB
x_CurrentChanged);
    }

    void comboBx_CurrentChanged(object sender,
      Autodesk.Revit.UI.Events.ComboBoxCurrentChangedEventArgs e)
    {
      // Cast sender as TextBox to retrieve text value
      ComboBox combodata = sender as ComboBox;
      ComboBoxMember member = combodata.Current;
      TaskDialog.Show("Combobox Selection", "Your new selection: " +
        member.ItemText);
    }
```
</C#>

6. Create other controls

There are other controls as well that you could use. Look through the Autodesk.Revit.UI namesapce to find them and try to create some of them yourself.

Now call all the control creation functions from the AddRibbonSampler() function.

</C#>
```
    public void AddRibbonSampler(UIControlledApplication app)
    {
      app.CreateRibbonTab("Ribbon Sampler");
      RibbonPanel panel =
        app.CreateRibbonPanel("Ribbon Sampler", "Ribbon Sampler");

      AddPushButton(panel);

      AddSplitButton(panel);

      AddComboBox(panel);
    }
```
</C#>

Then call AddRibbonSampler() at the end of OnStartup()

You could also place some of the controls on the slide out part of the panel – this part is only visible if the user clicks on the bottom strip of the Ribbon panel.
All controls added to the panel after a call to panel.AddSlideOut() will be added to the slide out part of the panel.

7. Summary

In this lab, we learned how to create your own Ribbon tab and panels and will them with controls. We have learned how to:

- Create a new Ribbon tab and panel
- Create various Ribbon controls