* 1. 

Demo Script

Office 2010 UI Customization

Lab version: 1.0.0

Last updated: 1/5/2011

* 1. 

1. Contents

[Overview 3](#_Toc281991190)

[Key Messages 3](#_Toc281991191)

[Key Technologies 3](#_Toc281991192)

[Time Estimates 4](#_Toc281991193)

[Setup and Configuration 4](#_Toc281991194)

[Demo Flow 4](#_Toc281991195)

[Opening Statement 5](#_Toc281991196)

[Step-by-Step Walkthrough 6](#_Toc281991197)

[Segment #1 – Exploring the Add-In 7](#_Toc281991198)

[Segment #2 – Exploring the Add-In Implementation 12](#_Toc281991199)

[Segment #3 – Checking Group Elements and Common UI Controls 15](#_Toc281991200)

[Segment #4 – Discovering Callbacks definitions 17](#_Toc281991201)

[Segment #5 – Analyzing Ribbon structure 19](#_Toc281991202)

[Segment #6 – Analyzing Backstage structure 20](#_Toc281991203)

[Segment #7 – Analyzing Context Menus structure 22](#_Toc281991204)

[Segment #8 – Allowing access to the Add-In on an specific range of cells 24](#_Toc281991205)

[Summary 25](#_Toc281991206)



Overview

* 1. This document provides setup documentation, systematic instructions, and a written script for showing Office 2010 UI Customization features and code. This document can also serve as a tutorial or walkthrough of the exposed technologies.
  2. Since Microsoft Office 2007, the applications that compose this suite like Word, Excel, PowerPoint, etc. have had new and improved look & feel. The Microsoft Office Fluent user interface (UI) replaced the layered menus, toolbars, and task panes with a simpler system optimized for efficiency and discoverability. The Fluent UI has improved context menus, Enhanced ScreenTips, a Mini toolbar, and keyboard shortcuts that help to improve user efficiency and productivity.
  3. Developers have taken advantage of the tools and programming structures in earlier versions of Office to extend the Fluent UI in creative ways. For example, the command bars object model enabled developers to build rich solutions in their custom Office applications. Continuing in that tradition, UI extensibility introduces an innovative model that you can use to enhance the user experience. You use extensible markup language (XML) and one of several conventional programming languages to manipulate the components that make up the Fluent UI. Because XML is plain text, you can create customization files in any text editor, which simplifies work with the Fluent UI. You can also reuse custom Fluent UI files with a minimum of adjustments because each application uses the same programming model.
  4. This demo shows a set of customizations that you can use on your Office Applications (Add-Ins) taking advantage of the Fluent UI of Office 2010. The demo consists of several scenarios showing a single feature in the easiest possible way.

### Key Messages

* 1. Microsoft Office Fluent user interface (UI) is one of the mayor enhancements since Microsoft Office 2007 and it allows UI improvements on the different Office Applications with the objective of enhancing the User Experience (UX).
  2. Developers can easily extend Microsoft Office Fluent UI (Ribbon, Backstage and Context menus) because it is based on XML and can be implemented in various programming languages. They will use it to create Office Applications (Add-Ins) and implement whatever they imagine. Also, Visual Studio 2010 provides templates and tools to make this process easier.

### Key Technologies

This demo uses the following technologies:

* 1. [Microsoft Visual Studio 2010](http://www.microsoft.com/visualstudio/en-us/)
  2. [Office 2010](http://www.microsoft.com/office/2010/en/default.aspx)

### Time Estimates

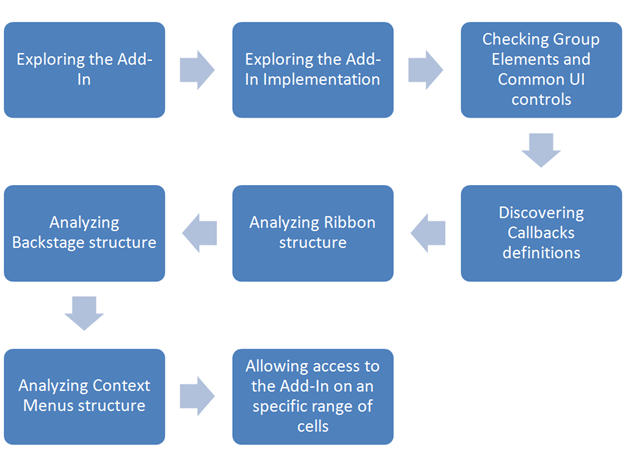
* + Estimated time for setting up and configuring the demo: 3 min
  + Estimated time to complete the demo: 45 min

Setup and Configuration

## System Requirements

* + [Microsoft Visual Studio 2010](http://www.microsoft.com/visualstudio/en-us/)
  + [Office 2010](http://www.microsoft.com/office/2010/en/default.aspx)

Demo Flow

1. The following diagram illustrates the high-level flow for this demo and the steps involved:
   1. 

Opening Statement

* 1. Today I would like to walk you through a demo application built to show the UI customizations available on Office 2010. This demo is based on an Excel 2010 Add-in functionality that you will use to explore the different options that you can implement to define your application’s look and feel.
  2. This demo takes advantage of the new Microsoft Office Fluent user interface (UI) available since Office 2007. The new UI replaced the system of layered menus, toolbars, and task panes with a simpler system optimized for efficiency and discoverability for the end user of our Office applications.

In this demo you will specifically see these things:

* 1. The three main extension points of the Office 2010 UI:
     1. The Ribbon: It replaces the old menus at the top of the screen for big beautiful icons that represents the different actions or buttons.
     2. The backstage: It is new in Office 2010 and it is accessible from the File tab. There you will implement tasks related with the document in general.
     3. Finally, the well-known Context Menus, that they have an improved UI and allow more variants thanks to the Fluent UI.
  2. How your Add-In can extend these points using xml files and the templates and tools provided by Visual Studio 2010 to make this process easier.
  3. Understand the common features of the extensible points and how to implement each of them specifically.

Step-by-Step Walkthrough

1. This demo is composed of the following segments:
   1. Exploring the Add-In
   2. Exploring the Add-In Implementation
   3. Checking Group Elements and Common UI controls
   4. Discovering Callbacks definitions
   5. Analyzing Ribbon structure
   6. Analyzing Backstage structure
   7. Analyzing Context Menus structure
   8. Allowing access to the Add-In on an specific range of cells

### Segment #1 – Exploring the Add-In

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open **Microsoft Visual Studio 2010** from Start | All Programs.   2. Open the **ExcelRibbonX.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference). |  |  |
| * 1. Press **F5** to debug the Add-In. | * + This demo consists of an Excel Add-In. that adds a Ribbon, a Backstage tab and Context Menu.   + When you run the solution, an instance of Excel opens, with the Add-In automatically enabled. |  |
| * 1. Select the row **C5** inside the spreadsheet.   2. Check the “**Contoso Parts**” tab in the Ribbon at the top of the screen.   3. Show the different buttons of the Tab. | * + The new toolbar at the top of the screen, which is the base of the Fluent UI, is called the Ribbon.   + This demo solution defines a mocked tab for the Ribbon showing some of the controls available for it.   + There is specific code to show the “**Contoso Parts**” tab only when you select a row inside the **C5:G8** range. We will explore that part of the code during the last segment of this demo.   + There are different controls and elements that you can define in your Ribbon tab that can be grouped into different categories.   + We will go through these features in the following segments. |  |
| * 1. Press the **File** tab.   2. Select the “**Contoso Process**” menu. | * + This section is called the Backstage. Here developers can combine document-level tasks and actions.   + The backstage was introduced in office 2010 and enhances the functionality provided by the Microsoft Office button in previous releases.   + The “**Contoso Process**” backstage page is added by this demo Add-In to show a possible usage of it.   + As the Ribbon tab, there are different controls and elements available for the Backstage that can be grouped in different categories and sections. |  |
| * 1. Click the “**Request Extension**” button.   2. Check that the UI is modified based on the performed action. | * + It is possible to define behavior when the user interacts with the different elements using something called **callbacks**, this will be explained more deeply below. |  |
| * 1. Exit the **File** tab and get back to the spreadsheet.   2. Right-click any cell inside the **C5:G8** range. | * + Context menus are shown when you right-click a section of a spreadsheet.   + This demo defines two menus to be shown in the Context menu when you right-click a cell inside the valid range. |  |
| * 1. Navigate inside the “Recalculate” and “Assets Risks” menus.   2. Close **Excel**.   3. Get back to **Visual Studio 2010** | * + New controls and elements with improved UI can be shown inside the new Context Menus of Office 2010. |  |

### Segment #2 – Exploring the Add-In Implementation

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Inside the Solution Explorer, expand the **ExcelRibbon** project node and explore its content. | * + This demo is based on an Excel Add-In project. All these kinds of project have a similar structure and can be created using the Excel Add-In project template. |  |
| * 1. Double-click the **ThisAddIn.cs** file (for C# projects) or the **ThisAddIn.vb** file (for Visual Basic projects) inside the **Excel** folder. | * + All Add-Ins have a specific class defined as entry point where initialization and recycling code can be defined.   + The default name of that class is **ThisAddIn**.   + This class should inherit from the **Microsoft.Office.Tools.AddInBase** abstract class (inherited definitions can be found inside the hidden ThisAddIn.Designer.xml file).   + The demo implementation overrides the CreateRibbonExtensibilityObject method to create a new instance of the RibbonX class. |  |
| * 1. Double-click the **RibbonX.cs** file (for C# projects) or the **RibbonX.vb** file (for Visual Basic projects).   2. Find the **GetCustomUI** method inside the **RibbonX** class. | * + RibbonX is a class created using the Ribbon template.   + It provides a way to extend the different sections of the Office UI and define the behavior I showed previously on this demo.   + The GetCustomUI method defines the name of the XML file where the UI elements and their structure are defined. By default, the XML file has a similar name to the Ribbon class but ends with an .xml extension. |  |
| * 1. Double-click the **RibbonX.xml** file. | * + All the controls and elements that are displayed in the different extensible points of the Office Fluent UI are defined here.   + This file has three sections: Ribbon, Backstage and Context Menus. Each of them contains the elements shown at the beginning of this demo. |  |

### Segment #3 – Checking Group Elements and Common UI Controls

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. If the **ribbon**, **backstage** or **contextmenu** elements in the **RibbonX.xml** file are collapsed, expand them | * + UI Sections are defined using XML tags. Most of the elements are common to the three sections, and have similar behavior or representations.   + Groups and UI Controls will be the most common elements that you will use when you create your UI, no matter if you are creating a tab for the Ribbon or a menu for the Context Menu. |  |
| * 1. Highlight a group element inside the ribbon element. | * + Grouping similar elements is a common task when creating an Add-In UI; since it allows organizing controls and elements in sections improving the User Experience.   + To define a group or category you must add the **group** XML element.   + Elements inside the same group are defined within the group element.   + Different attributes can be added to the group element, for example:     - **id**: indicates the group identifier     - **label**: indicates the group display name     - **autoscale**: sets if the group is allowed to collapse elements inside it when the ribbon is resized horizontally.     - **imageMso**: defines the image that should be shown when the ribbon is too small to show the icons inside the group. |  |
| * 1. Check the different UI controls inside the xml. | * + UI Controls are the atomic elements that you will use to create your UI.   + If you used Visual Studio before, they are as the controls for a Desktop application, or the Web Controls for an ASP.NET Application.   + The most common controls are:     - **button**     - **toggleButton** (A button that can be pressed or released)     - **editBox** (A textbox to edit values)     - **dropdown** (A classical combobox)     - **imageControl** (A control to show images)     - **hyperlink** (A control to create links to other documents or perform specific actions)   + Elements also share common attributes like:     - **id**: indicates the control Identifier     - **label**: indicates the control display mame     - **imageMso**: sets the image associated with the control |  |

### Segment #4 – Discovering Callbacks definitions

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. If the backstage element is collapsed, expand it.   2. Highlight the button with the **RequestExtensionButton** id. | * + On this markup, you can see the definition for the button we clicked before, which had associated behavior.   + To link a button with the behavior, you must define the appropriate callbacks. This is the most common way to add functionality to an Add-Ins.   + Callbacks allow defining a method name inside the xml file which will be invoked when the user executes an action on the element.   + In this demo, you can see that the button has the **onAction** attribute defined and it is set to **ExtendDesignDueDate**. |  |
| * 1. Double-click the **RibbonX.cs** file (for C# projects) or the **RibbonX.vb** file (for Visual Basic projects).   2. Find the **ExtendDesignDueDate** method inside the **RibbonX** class. | * + The behaviors are implemented inside the class’ code, related with the xml file.   + The method names have to match the value defined in the callback attribute, in this case, the one defined in the **onAction** attribute of the control: **ExtendDesignDueDate**.   + The method has to receive as unique parameter a **Microsoft.Office.Core.IRibbonControl** object, which is the instance of the element against which the action was performed. |  |
| * 1. Add a breakpoint to the first line of the **ExtendDesignDueDate** method.   2. Press **F5** to debug the Add-In again.   3. Open the **File** tab, and select the “**Contoso Process**” menu.   4. Click the “**Request Extension**” button.   5. Press **Shift + F5** or click **Stop Debugging** on the **Debug** menu. | * + Visual Studio 2010 provides debugging of the Microsoft Office Add-Ins.   + You only have to add breakpoints and perform the required actions to raise the callbacks. |  |

### Segment #5 – Analyzing Ribbon structure

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Double click on the RibbonX.xml file to open it.   2. Highlight the tab element, inside tabs. | * + The Ribbon is composed mainly of a collection of Tabs containing controls to interact with your application. To define a new tab you must add a new <tab> element inside the <tabs> collection in the Ribbon configuration file.   + In this demo, we only included one tab named “Contoso Parts”. |  |
| * 1. Press **F5**, and go to **D5** in the Excel instance opened.   2. Show the name of the tab.   3. Play with the tab controls to show there is no functionality implemented.   4. Close Excel. | * + You will be able to check the new added tab by starting a new instance of the application.   + Since, there is code in the application to filter the mocked functionality of the tab to certain cells; you should click in a specific range in order to enable the Tab.   + Now, notice that the “Contoso Parts” Tab has several groups of controls. These controls have no functionality, but they help you realize the different ways in which you can customize the Office UI. |  |
| * 1. In Visual Studio, double click on the **RibbonX.xml** file to open it if it is not already open.   2. Highlight the **GroupDrive** group. | * + Inside a Tab, you can define controls and group of controls.   + As you can see, there are several types of controls to choose from and several properties to play with.   + Later on this demo, you will see how to control the tab visibility from custom code using callbacks. |  |

### Segment #6 – Analyzing Backstage structure

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Collapse the **<ribbon>** element in order to ensure the **<backstage>** element is visible.   2. Highlight the Backstage element. | * + The main difference you will notice in the <backstage> element is the presence of a single <tab> element instead of a <tabs> collection.   + Even though, the creation of controls remains similar to the Ribbon, you group and order elements in columns. |  |
| * 1. Highlight the **firstColumn** element.   2. Scroll down and highlight the **secondColumn** element. | * + You can see that this demo defines two different columns to group the items.   + To show this behavior, in the sample we placed all the controls that the user can interact with, in the first column.   + While on the second one, a column-graph image will be displayed. |  |
| * 1. Highlight the **primaryItem** and **topItems** definition inside the group element.   2. Press F5 to start the solution.   3. In Excel, point to File and select the Contoso Process Backstage tab   4. Show the primaryItem and the topItems. | * + To order the elements in each column, each group has the ability to define the position of the items.   + In this demo, you can see that a primary Item containing a button is defined   + Additional items are rendered inside the topItems element.   + To understand how this positioning works, let us run the demo again. |  |
| * 1. Close excel.   2. Highlight the layoutContainer in Visual studio | * + Within the topItems element, you can define one or more layout containers,   + These containers are not only useful for grouping controls, but also allows us to describe the layout in which the will be rendered, for example, horizontal. |  |

### Segment #7 – Analyzing Context Menus structure

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Collapse the backstage element.   2. Highlight the contextMenu item. | * + Prior to Microsoft Office 2010, the only way to customize context menus in the Microsoft Office Fluent Ribbon user interface (UI) was by using CommandBars solutions.   + In Microsoft Office 2010, you can customize built-in context menus just as any other component of the Ribbon UI.   + In this demo, two different context menu items are defined. |  |
| * 1. Highlight the gallery element declaration   2. Highlight the getItemImage callback declaration. | * + The gallery element is a context menu item that contains a set of images as sub-item.   + These images are populated with the getItemImage callback declaration. |  |
| * 1. Highlight the splitButton element declaration   2. Highlight the menu items declaration. | * + The splitButton is a button that contains other buttons within it.   + These buttons are defined inside the menu element declared in the xml. |  |
| * 1. Highlight the insertAfterMso attribute declaration | * + The insertAfterMso attribute defines the position of the element in the context menu.   + In this demo, the elements are positioned after the PasteSpecialAlt element. |  |
| * 1. Highlight the last two button elements. | * + Additionally, the Add-in binds the FormatCellsDialog and HyperlinkInsert context menu element visibility to the GetNegatedProcessToolsVisibility   + We will see how the visibility binding works on the next segment.   + Office 2010 provides these two elements by default. |  |

### Segment #8 – Allowing access to the Add-In on an specific range of cells

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Double-click the **RibbonX.cs** file (for C# projects) or the **RibbonX.vb** file (for Visual Basic projects).   2. Locate the Ribbon\_Load method.   3. Highlight the **SheetSelectionChange** event handler declaration. | * + This demo toggles the Contoso ribbon and context menus visibility, for a specific cells range.   + This behavior implemented using cutom code using the SheetSelectionChange event of the ExcelApplication class. |  |
| * 1. Locate the excelApplication\_SheetSelectionChange method. | * + When the event handler gets in control, it verifies if the selected cell is within the defined range and sets the InsideData value to true, which consecutively activates the **TabParts**.   + If the selected cell is not valid, setting the InsideData value to false deactivates the **TabHome**. |  |
| * 1. Locate the GetProcessToolsVisibility and GetNegatedProcessToolsVsibility methods.   2. Double click on the RibbonX.xml file to open it.   3. Locate the TabParts and highlight the getVisible attribute. | * + The ribbon and context menu elements visibility is restricted to the same behavior through the GetProcessToolsVisibility and GetNegatedProcessToolsVsibility methods. |  |

Summary

* 1. In this demo, you saw how you can take advantage of the Microsoft Office Fluent user interface (UI) to create Office Applications with a professional look & feel. Moreover, you learn how easy is to construct the UI in a declarative way using xml, and how the controls behavior can be written in the language of your preference.