

SOLUTIONS FOR DEVELOPING AND EXTENDING RICH GRAPHICAL USER INTERFACES FOR OFFICE APPLICATIONS

Dana-Mihaela PETROȘANU *¹

Alexandru PÎRJAN ²

ABSTRACT:

In this paper, we analyze solutions for developing and extending rich Graphical User Interfaces (GUI) for Office applications through Component Object Model Add-Ins (COM Add-Ins). We begin by presenting the Visual Studio Tools for Office (VSTO), its previous versions and the main features that it offers to the developers. In the second section, we make a survey of the most important Ribbon controls offered by the latest version of VSTO, launched in 2013, depicting their functionality, properties and possible applications. In the third section, we develop an example for extending rich GUI for Word 2013 through COM Add-Ins. The developed extension implements a new customized tab, three new customized groups within it containing the main customized Office controls: CheckBox, Button, ToggleButton, vertical separators, ToggleButton, ComboBox, DropDown, EditBox. Through this example, we highlight the main VSTO controls and their properties.

Keywords: VSTO 2013, COM Add-ins, Ribbon (XML), Ribbon controls, developer.

1. Introduction

VSTO is a core component of the development environment Visual Studio that facilitates the development of custom Office applications. VSTO represents an useful tool for the developers and a viable, improved alternative for those that were previously familiar with Visual Basic for Applications. VSTO helps developers to customize Office starting from simple Add-Ins, up to complex customized documents, improving and extending the existing functionality, facilitating the integration and assembly of different Office applications, of documents and databases. VSTO helps developers create custom Word documents designed for complex tasks, to enhance Outlook through new add-ins useful in managing and scheduling more efficiently e-mails, to customize the Ribbon interface, reaching up to developing new Ribbons according to the user's specific needs.

VSTO extends some of the existing objects in the Office object model, supports the Microsoft .NET Framework and is a component of the Visual Studio .NET tools suite. It helps developers to program and integrate applications targeting most of the Office suite applications. VSTO represents a collection of useful tools available as project templates and a runtime that enables Office applications to make use of the advantages offered by the .NET Framework. Therefore, a developer can program complex Office solutions that extend the features of the Office suite, using the Common Language Infrastructure programming languages like C# and Visual Basic.

¹ * Corresponding author. PhD, Department of Mathematics-Informatics, University Politehnica of Bucharest, 313, Splaiul Independentei, district 6, code 060042, Bucharest, Romania, E-mail: danap@mathem.pub.ro

² PhD, Faculty of Computer Science for Business Management, Romanian-American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania, E-mail: alex@pirjan.com

Before the launching of Visual Studio .NET and of the .NET Framework in 2002, the developers of Office applications that needed to create COM Add-ins had a limited set of development tools available. Afterwards, the VSTO 2005 was launched as an Add-In to Visual Studio Professional, supporting the Office versions 2003 and 2007. The latest version of this software is VSTO 2013, compatible with Office 2007 and with more recent versions (2010, 2013).

Similarly to the Visual Basic for Applications code, a specific virtual machine, a component of the .NET framework, runs the VSTO software code. In the VBA applications, the code is stored right inside the document file, while in the VSTO programs the code is stored in a compiled code library, the Common Language Infrastructure (CLI) assemblies. These are associated through custom properties with the documents and the Office environment makes use of the Common Language Runtime, the Common Language Infrastructure assembly being loaded into a separate application domain, a mechanism used within the CLI to isolate executed software applications in order to avoid interacting with each other. Comparing VSTO and VBA, one can observe that the development of VSTO software is performed through Visual Studio, while the VBA code is developed right within the Office application, without requiring specific tools. A VBA specific tool is the macro recorder that can automatically transform the user's actions in VBA code.

Compared with other development solutions, like Apps for Office or VBA, the VSTO proves to be the best solution for the developers that want to extend or improve the Office applications. Its main advantages are the fact that it allows to customize the user interface, to improve the automation of specific tasks, to add or edit charts or shapes, to manipulate documents, to interact with other programs hosted on the same computer (or not). Thus, VSTO is allowed to use the power and resources stored in the host computer, interacting with the file system, benefiting from the developer's skills.

The paper has the following structure: in the second section, we make a survey of the most important *Ribbon* controls offered by the latest version of VSTO, launched in 2013, depicting their functionality, properties and possible applications. In the third section, we develop an example for extending rich GUI for Word 2013 through COM Add-Ins. The developed extension implements a new customized tab, three new customized groups within it containing an implementation of the main customized Office controls: CheckBox, Button, ToggleButton, vertical separators, ToggleButton, ComboBox, DropDown, EditBox. Through this implementation, we highlight the main VSTO controls along with their properties.

2. A survey of the most important Ribbon controls offered by Visual Studio Tools for Office 2013

VSTO 2013 offers the possibility to extend the default features of the Office suite by developing customized solutions suited for each activity. VSTO 2013 offers the following template types for developing Office solutions: Component Object Model (COM Add-ins) at the application level; Programmatic control and customization at the document level.

VSTO offers the possibility to execute source code only when a certain document is opened or every time an application runs. In order to extend the functionality of the

existing *Ribbon* bar or to repurpose the built-in commands, we must take into account the controls that the Office Ribbon provides, along with their object and event model. All the *Ribbon* controls have a set of common properties that is complemented by specific properties corresponding to the particularities of each control. The set of common properties for all the *Ribbon Controls* consists in [1]: *Enabled* property that is of type *bool* and allows to enable or disable a control; *Id* property that is of type *string* and represents the unique identifier of a control; *Name* property that is of type *string* and represents the name of the control; *Tag* property that is of type *object* and offers the possibility to specify custom data associated with the control at runtime.

The *Box* control is useful for grouping other *Ribbon* controls horizontally or vertically. These controls are invisible when being displayed on screen (their border is not displayed), their purpose being that of grouping other Ribbon controls like: *Box*, *Button*, *Button Group*, *Check Box*, *Combo Box*, *Drop Down*, *Edit Box*, *Gallery*, *Label*, *Menu*, *Split Button* and *Toggle Button* controls. *RibbonBox* represents the object associated within the object model with a *Box* control. The *Box* object provides an *Items* collection that returns a *RibbonComponentCollection* object. Using this object, one can iterate over the existing controls contained by the *Box* control without being able to add other controls or remove the existing ones. The controls that are contained by the *Box* control can be specified only at design time. Using the *BoxStyle* property of type *RibbonBoxStyle*, the *BoxStyle* can be set to horizontal (*RibbonBoxStyle.Horizontal*) or vertical (*RibbonBoxStyle.Vertical*).

In VSTO 2013, one of the most commonly used controls is the *Button* control. *RibbonButton* represents the object associated within the object model with a *Button* control. In addition to the set of common properties, the *RibbonButton* exposes the following properties:

- *ControlSize* is a property having the type *RibbonControlSize* that allows the developer to specify or retrieve the *RibbonButton's* size as *RibbonControlSizeLarge* or *RibbonControlSizeRegular*.
- *Description* property that is of type *string*, is used when the *ControlSize* is *RibbonControlSizeLarge* and the *RibbonButton* is contained within a *Menu* or a *SplitButton* control.
- *Image* property that is of type *Image*, allows the developer to specify or retrieve the *RibbonButton's* image that is associated with the control.
- *ImageName* property that is of type *string*, allows the developer to enable a different method for loading images. If the developer did not set the properties *Image* and *OfficeImageId* but he has set the *ImageName* property, the Office environment will raise the *LoadImage* event through which the developer is passed the *ImageName* and he can load images in the handling of this event.
- *KeyTip* property that is of type *string*, allows the developer to specify a keyboard shortcut for the *RibbonButton*. The property must be one to three uppercase characters long and must not contain spaces, tabs or new-line characters. The key tips are being shown when the user presses the Alt key.
- *Label* property that is of type *string*, allows the developer to specify or retrieve the *RibbonButton's* label that will be displayed on the screen to the user.
- *OfficeImageId* property that is of type *string*, allows the developer to specify or retrieve the identification of a built-in Office image.

- *Position* property that is of type *RibbonPosition*, allows the developer to position the *Button* before or after a built-in Office control.
- *ScreenTip* property that is of type *string*, allows the developer to specify or retrieve the single-line text displayed when the mouse hovers the *Button* control.
- *ShowImage* property that is of type *bool*, allows the developer to specify or retrieve if an image is being displayed on the control. *ShowImage* is always true if the property *ControlSize* has been set to *RibbonControlSizeLarge*.
- *ShowLabel* property that is of type *bool*, allows the developer to specify or retrieve whether a label is shown on the control. *ShowLabel* is always true if the property *ControlSize* has been set to *RibbonControlSizeLarge*.
- *SuperTip* is a property having the type *string* that allows the developer to specify or retrieve the multi-line text displayed when the mouse hovers the *Button* control.
- In addition to the above-mentioned properties, the *RibbonButton* also provides a *Click* event that receives a parameter of type *RibbonControlEvents*.

The *ComboBox* can be regarded as a fusion between an edit box and a drop-down control. It provides a label and an image, the developer being able to choose if he wants to display the image, the label or both of them. The developer can set the width for the edit box but not for the drop-down list. *RibbonComboBox* represents the object associated within the object model with a *ComboBox* control. The *RibbonComboBox* exposes an *Items* collection that returns a *RibbonDropDownItemCollection*.

The collection can be used by developers to add, delete or update *RibbonDropDownItem* controls at runtime. The properties *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *ScreenTip*, *ShowImage*, *ShowLabel* and *SuperTip* are also offered by a *ComboBox* control and they have similar functionality as in the case of the *Button* control analyzed before.

In addition, *RibbonComboBox* exposes several properties regarding the edit box. The *MaxLength* property that is of type *int*, allows the developer to specify or retrieve the maximum length of input allowed in the edit-box of the *ComboBox*. The *Text* property of type *string* allows the developer to specify or retrieve the text in the editbox. The *ShowItemImage* property that is of type *bool* allows specifying or retrieving if the images are displayed for the *RibbonDropDownItem* controls in the *Items* collection.

If the user edits the text in the edit box or he exits the edit box by pressing the Enter key or changes the focus to another part in the graphical interface the *RibbonComboBox* fires a *TextChanges* event. The *TextChanges* event is also fired when the user selects another value from the drop-down list leading to a change of the text. VSTO also offers an *ItemsLoading* event that is triggered before the dropdown list is displayed, thus offering to the developer a chance to alter the content of the *Items* collection before this would have been displayed to the user.

The *DropDown* control consists in a drop-down list from which the user has the possibility to choose a value. The developer can set to a *DropDown* control a label, an image or both. The text of the item that has been selected from the dropdown list is displayed in the drop-down list when the list is collapsed. *RibbonDropDown* represents the object associated within the object model with a *DropDown* control and contains a *Buttons* collection of type *RibbonButtonCollection* that contains *RibbonButton* objects. The developer is not allowed to add or delete buttons from the collection at runtime, he can only show or hide the existing buttons within the collection.

The *RibbonDropDown* also contains an *Items* collection of *RibbonDropDownItems* containing *RibbonDropDownItem* controls. In this case, the collection can be altered at runtime. VSTO also offers an *ItemsLoading* event that is fired before the dropdown list of the *DropDown* control is displayed, thus offering to the developer a chance to alter the content of the *Items* collection before this would have been displayed to the user. *RibbonDropDown* provides some properties for customizing the display of the *Items* collection.

The *ItemImageSize* property is of type *Size* and allows the developer to specify or retrieve the size of the images that are being displayed in the *Gallery* control. The *SelectedItem* property is of type *RibbonDropDownItem* and allows specifying or retrieving the selected item from the *Items* collection. The *SelectedItemIndex* property that is of type *int* allows the developer to specify or retrieve the selected item from the *Items* collection by using its index. The *ShowItemImage* property that is of type *bool*, offers the possibility to specify or retrieve if images are to be displayed for the items within the drop-down list.

The *ShowItemLabel* property that is of type *bool* allows specifying or retrieving if labels are to be displayed for the items in the drop-down list. The properties *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *ScreenTip*, *ShowImage*, *ShowLabel* and *SuperTip* are also offered by a *RibbonDropDown* control and they have similar functionality as in the case of the *Button* control that we have analyzed before.

Regarding the events provided by the *RibbonDropDown*, the developer has access to three events: *ButtonClick* that is fired according to the button instance when one of the buttons from the *Buttons* collection is clicked; *SelectionChanged* that is fired when a *RibbonDropDownItem* from the *Items* collections is selected, thus leading to a change of the currently selected *RibbonDropDownItem*; *ItemsLoading* is triggered before the drop-down list is being displayed to the user, so the developer has an opportunity to alter the contents of the *Items* collections before the *Gallery* drop-down list is displayed.

The *EditBox* control offers the user the possibility to input a value. The *EditBox* provides a label, an image or both of them. The developer can also specify a default value for the *EditBox*. *RibbonEditBox* represents the object associated within the object model with an *EditBox* control. The *RibbonEditBox* control provides the following properties that we have already analyzed before: *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *ScreenTip*, *ShowImage*, *ShowLabel* and *SuperTip*. The *MaxLength* property that is of type *int* offers the possibility to the developer of specifying or retrieving the maximum length of input that can be entered in the edit box. The *Text* property that is of type *string* makes it possible to specify or retrieve the text from the edit box. The *SizeString* property is of type *string* and accepts a sample string based on which the width of the edit box is set. The *RibbonEditBox* provides a *TextChanged* event that is fired when the text in the edit box has been changed by the user who afterwards exits the edit box either by using the Enter key or by moving the focus to another part of the user interface.

The *Gallery* control offers the possibility to display a drop-down list of items arranged in a grid and a supplementary area for inserting additional buttons at the bottom of the list. The *Gallery* control allows the developer to keep always the highlight over a selected item or to turn it off. The developer can specify the items of a *Gallery* control, either at design time or at runtime, by adjusting them dynamically. *RibbonGallery* represents the object associated within the object model with a *Gallery* control. The *RibbonGallery* contains a *RibbonButtonCollection* of *RibbonButton* objects. The collection cannot be

altered at runtime, it does not allow to add or delete buttons, the developer can only show and hide the existing buttons within the collection. The *RibbonGallery* also contains an *Items* collection of *RibbonDropDownItems* containing *RibbonDropDownItem* controls. In this case, the collection can be altered at runtime. VSTO also offers an *ItemsLoading* event that is fired before the dropdown list of the *Gallery* control is displayed, thus offering to the developer a chance to alter the content of the *Items* collection before this would have been displayed to the user.

RibbonGallery provides some properties for customizing the display of the *Items* collection. The *ColumnCount* property is of type *int* and it allows the developer to specify or retrieve the number of columns that are being displayed in the *Gallery* control. The *RowCount* property is of type *int* and allows specifying or retrieving the number of rows displayed in the *Gallery*. The *ItemImageSize* property is of type *Size* and allows the developer to specify or retrieve the size of the images that are being displayed in the *Gallery* control. The *SelectedItem* property is of type *RibbonDropDownItem* and allows specifying or retrieving the selected item from the *Items* collection. The *SelectedItemIndex* property that is of type *int* allows the developer to specify or retrieve the selected item from the *Items* collection by referring it, using its index. The *ShowItemImage* property that is of type *bool*, offers the possibility to retrieve or specify if the images are to be displayed for the items within the *Gallery*. The *ShowItemLabel* property that is of type *bool*, allows specifying or retrieving if labels are to be displayed for the items in the *Gallery*. The *ShowItemSelection* property that is of type *bool*, allows specifying or retrieving if an item is highlighted in the *Gallery* or not.

Likewise, the *RibbonGallery* has other properties that have been previously analyzed: *ControlSize*, *Description*, *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *Position*, *ScreenTip*, *ShowImage*, *ShowLabel*, *SuperTip*. *RibbonGallery* also offers three events: *ButtonClick* event that is fired when one of the buttons from the *Buttons* collection is clicked; *Click* event that is fired when a *RibbonDropDownItem* from the *Items* collections is clicked; *ItemsLoading* event that is triggered before the *Gallery* drop-down list is displayed to the user, so the developer has an opportunity to alter the *Items* collections before the *Gallery* drop-down list is displayed.

The *Group* control can be regarded as a container for most of the *Ribbon* controls. A developer cannot add controls directly to a tab within the *Ribbon*. The tab must contain a *Group* control in which the elements will be added. The developer has the possibility to specify a label for a *Group* control and also a *DialogLauncher* icon in the bottom-right position of the *Group*. When this button is clicked, an event is triggered that offers the developer the possibility to create a custom dialog box. The developer has also the possibility to assign a label and an image to the *Dialog Launcher* that will be displayed if the *Dialog Launcher* is added to the *Quick Access* toolbar. *RibbonGroup* represents the object associated within the object model with a *Group* control. The *RibbonGroup* also contains an *Items* collection of *RibbonComponentCollection* type containing the controls within the *Group* control. The *DialogLauncher* property is of type *DialogLauncher* and allows the developer to specify or retrieve the settings of the *DialogLauncher* for the *Group* control. The *RibbonGroup* control provides the following properties that we have already analyzed before: *KeyTip*, *Label* and *Position*. The *RibbonGroup* also offers the *DialogLauncherClick* event that is triggered if the *RibbonGroup* contains a

DialogLauncher and the user clicks its icon from the bottom right position of the *Group*. A developer is not allowed to add custom controls to a built-in Office group. When one desires to label a certain part of the *Ribbon*, he can use a *Label* control. *RibbonLabel* represents the object associated within the object model with a *Label* control. The *RibbonLabel* offers access to the following properties: *Label*, *ScreenTip*, *ShowLabel* and *SuperTip*. Their functionality has been analyzed along with the previous controls.

Using a *Menu* control, the developer can incorporate more controls in the list that is dropped when the menu is accessed. A *Menu* control can include the following controls: *Button*, *Check Box*, *Gallery*, *Separator*, *Toggle Button* and *Split Button*. A *Menu* control can also include other *Menu* controls. A developer can insert the items within a *Menu* either at design time or dynamically, at runtime. A *Menu* control can have a normal or a large size in order to accommodate the items. *RibbonMenu* represents the object associated within the object model with a *Menu* control. The *RibbonMenu* also contains an *Items* collection of *RibbonComponentCollection* type containing the controls associated with the *Menu*. The *RibbonMenu* also offers an *ItemsLoading* event that is triggered before the menu is displayed to the user, so the developer has an opportunity to alter the contents of the *Items* collections before the menu is displayed. In order to change the *Items* collection at runtime, the *Dynamic* property, which is of *Boolean* type, must have the value *true*. The *ItemSize* property is of type *RibbonControlSize* and allows the developer to specify or retrieve the size of the *Items* that the *Menu* contains.

The *RibbonMenu* control provides the following properties: *ControlSize*, *Description*, *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *Position*, *ScreenTip*, *ShowImage*, *ShowLabel* and *SuperTip*. The *Title* property represents the text that is being displayed at the top of the *Menu* when its list of items is displayed.

OfficeRibbon represents the object associated within the object model with the *Ribbon* bar. The *Context* property is of type *object* so it can be cast to an object of type *Inspector* in Outlook. The developer is able to retrieve the *Inspector* corresponding to the *Ribbon* using the *Context* property. The *Context* property returns the *null* value in other Office applications. The developer can set the *false* value to the *Global* property in order to be certain that each of the *Inspector* windows corresponds to a unique *Ribbon* instance. The developer has the possibility to hide all the built-in tabs and display only his custom developed ones by setting the *StartFromScratch* property that is of type *bool*, to the *true* value. All the *RibbonTab* objects contained within the *Ribbon* are included in the *RibbonComponentCollection* that can be retrieved using the *Tabs* property. The *OfficeRibbon* exposes the following events: the *Load* event, that is triggered when the *Ribbon* is loaded; the *Close* event, that is triggered when the *Ribbon* is closed; the *LoadImage* event is triggered when the *Ribbon* is loaded if the *ImageName* property has been previously set.

The *Separator* control is displayed as a vertical separation bar if it is used within a group and as a horizontal separation bar when used within a menu. *RibbonSeparator* represents the object associated within the object model with a *Separator* control. The *RibbonSeparator* has the following properties: *Position* (that is of type *RibbonPosition*

and allows the developer to position the *Separator* before or after a built-in Office control) and *Title* (that is of type *string*).

Using a *Split Button* control, the developer can incorporate more controls in the list that is dropped when the *Split Button* is accessed. A *Split Button* control can include the following controls: *Button*, *Check Box*, *Gallery*, *Separator*, *Toggle Button* and *Menu Button*. A *Split Button* control can also include other *Split Buttons*. The *Split Button* control offers the possibility to be clicked directly, as a button control, or to access its dropdown list. The *Split Button* can be set to perform as a *Toggle Button* control having two states: the pressed state and the de-pressed state.

Unlike the *Menu* control, the *Split Button*'s items from the list cannot be altered dynamically at runtime. *RibbonSplitButton* represents the object associated within the object model with a *Split Button* control. The *RibbonSplitButton* exposes an *Items* collection of type *RibbonComponentCollection*, containing the controls within the list. The collection cannot be altered at runtime. The *ItemSize* property is of type *RibbonControlSize* and allows the developer to specify or retrieve the size of the *Items* contained in the list. The *ButtonEnabled* property is of type *bool* and allows the developer to specify or retrieve the state of the button within the *Split Button* control (enabled or not). The *ButtonType* property is of type *RibbonButtonType* and allows the developer to specify or retrieve the type of the button (*Button* or *ToggleButton*). The *RibbonSplitButton* also offers the following properties: *Title*, *Description*, *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *Position*, *ScreenTip*, *ShowLabel* and *SuperTip*. A *Click* event is fired when the button within the *Split Button* control is clicked.

The *Tab* control is an essential part of the *Ribbon*, having a caption and a key tip that indicates a shortcut for activating the tab. *RibbonTab* represents the object associated within the object model with a *Tab* control. The *ControlId* property is of type *RibbonControlIdType* that represents the unique name used by Office for the respective tab. A tab has associated a *Groups* collection that is of type *RibbonComponentCollection* and includes all the controls within the group associated with the tab that have been created by the Add-In. The *RibbonTab* also exposes the properties *KeyTip*, *Label* and *Position*.

There are many similarities between a *Toggle Button* control and a button control, the difference being the fact that it has two states: a pressed and a de-pressed state. *RibbonToggleButton* represents the object associated within the object model with a *Toggle Button* control. The *Checked* property is of type *bool* and allows the developer to specify or retrieve the state of the *Toggle Button* control (pressed or de-pressed). The *Toggle Button* control has the following properties: *ControlSize*, *Description*, *Image*, *ImageName*, *KeyTip*, *Label*, *OfficeImageId*, *Position*, *ScreenTip*, *ShowImage*, *ShowLabel* and *SuperTip*. In addition to the above-mentioned properties, the *Toggle Button* control also provides a *Click* event that is fired when the *Toggle Button* control is clicked.

In order to exemplify the VSTO controls and their properties, in the following section we develop a new customized tab and three customized groups containing within the most important of the above-mentioned controls.

3. Developing rich Graphical User Interfaces for Word 2013 through Component Object Model Add-Ins

In order to develop a rich GUI for Word 2013 and extend the existing Ribbon through COM Add-Ins, we have created a new customized tab and three customized groups within it (**Fig. 1**).



Fig. 1. The new customized tab and the three customized groups within it

The new tab is entitled *CONTROALE* and has the following structure:

1. The first group within the new tab, entitled *Grup1*, contains 4 menus: the first menu, entitled *Meniu1*, contains three *CheckBox* controls, entitled *CheckBox1*, *CheckBox2* and *CheckBox3*; the second menu, entitled *Meniu2*, contains three *CheckBox* controls, entitled *CheckBox4*, *CheckBox5* and *CheckBox6*; the third menu, entitled *Meniu3*, contains three *Button* controls, entitled *Buton1*, *Buton2* and *Buton3*; the fourth menu, entitled *Meniu4*, contains two *ToggleButton* controls, entitled *ToggleButton1* and *ToggleButton2* (**Fig. 2**).



Fig. 2. Creating the first group within the new tab, entitled *Grup1*

2. The second group within the new tab, entitled *Grup2*, contains: three *CheckBox* controls, entitled *CheckBox7*, *CheckBox8* and *CheckBox9*; a vertical separator; three *Button* controls of normal size, entitled *Buton4*, *Buton5* and *Buton6*; a vertical separator; three *ToggleButton* controls of normal size, entitled *ToggleButton3*, *ToggleButton4* and *ToggleButton5*; a vertical separator; a *Button* control of large size, entitled *Buton7*; a vertical separator; a *ToggleButton* control of large size, entitled *ToggleButton6* (**Fig. 3**).

```

<group id="g2" label="Grup2">
  <checkBox id="ck7" label="CheckBox7"/>
  <checkBox id="ck8" label="CheckBox8"/>
  <checkBox id="ck9" label="CheckBox9"/>
  <separator id="sep1"/>
  <button id="b4" label="Buton4" imageMso="ReplyAll"/>
  <button id="b5" label="Buton5" imageMso="FormattingRequiredField"/>
  <button id="b6" label="Buton6" imageMso="ServerRestoreSqlDatabase"/>
  <separator id="sep2"/>
  <toggleButton id="tg3" label="ToggleButton3" imageMso="TextReflectionGallery"/>
  <toggleButton id="tg4" label="ToggleButton4" imageMso="DistributionListRemoveMember"/>
  <toggleButton id="tg5" label="ToggleButton5" imageMso="SortRemoveAllSorts"/>
  <separator id="sep3"/>
  <button id="b7" label="Buton7" size="large" imageMso="Risks"/>
  <separator id="sep4"/>
  <toggleButton id="tg6" label="ToggleButton6" size="large" imageMso="SharingOpenWsDiscussionList"/>
</group>

```

Fig. 3. Creating the second group within the new tab, entitled Grup2

3. The third group within the new tab, entitled *Grup3*, contains: a *ComboBox* control, entitled *ComboBox1*, that contains four *Items*, entitled *Item1*, *Item2*, *Item3* and *Item4*; a *DropDown* control entitled *DropDownBox1*, that contains four *Button* controls, entitled *Buton8*, *Buton9*, *Buton10* and *Buton11*; a *DropDown* control entitled *DropDownBox2*, that contains four *Items*, entitled *Item5*, *Item6*, *Item7* and *Item8*; a *Button* control of large size, entitled *Buton12*, that contains two *ToggleButton* controls entitled *ToggleButton8* and *ToggleButton9*; three *EditBox* controls, entitled *EditBox1*, *EditBox2* and *EditBox3* (Fig. 4).

```

<group id="g3" label="Grup3">
  <comboBox id="com1" label="ComboBox1" sizeString="abcdefghijkl" imageMso="FormPublish">...</comboBox>
  <dropDown id="drop1" label="DropDownBox1" sizeString="abcdefghijkl" imageMso="MacroShowAllAct">...</dropDown>
  <dropDown id="drop2" label="DropDownBox2" sizeString="abcdefghijkl" imageMso="FileDocumentMan">...</dropDown>
  <splitButton id="spb1" size="large">...</splitButton>
  <separator id="sep5"/>
  <editBox id="ed1" label="EditBox1" sizeString="abcdefghijklmnoprst" imageMso="TableSharePointListsModifyColumnsAndSettings"/>
  <editBox id="ed2" label="EditBox2" sizeString="abcdefghijklmnoprst" imageMso="CreateReportFromWizard"/>
  <editBox id="ed3" label="EditBox3" sizeString="abcdefghijklmnoprst" imageMso="SlideMasterDiagramPlaceholderInsert"/>
</group>

```

Fig. 4. Creating the third group within the new tab, entitled Grup3

Finally, we obtain the customized extended rich GUI for Word 2013, developed in VSTO 2013 and containing within it the most important controls depicted in the previous section (Fig. 5).

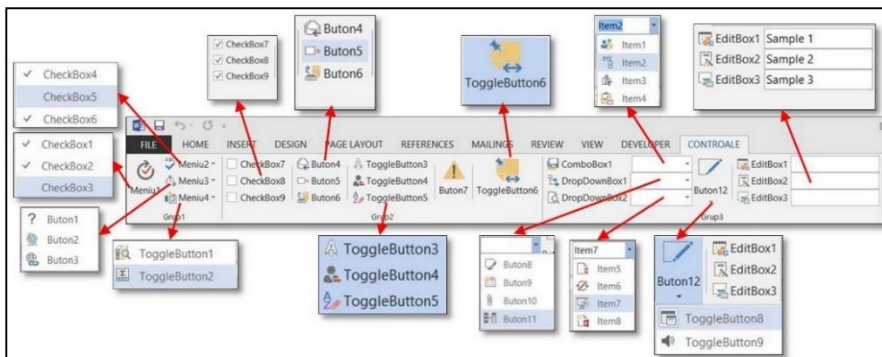


Fig. 5. An example for developing rich Graphical User Interfaces for Word 2013 and extending the Ribbon bar through Component Object Model Add-Ins

4. Conclusions

VSTO offers a versatile range of controls and an efficient possibility of programmatically controlling them, facilitating the development of complex Office solutions. It has indisputable advantages arising from the .NET integration, offering the possibility of developing robust projects, using Visual Studio in order to program the software solutions. The most important of the benefits that it brings consists in the fact that it offers the possibility of developing rich Graphical User Interfaces for Office applications and extending the existing features according to the user's specific needs and requirements.

5. Acknowledgement

This paper presents a series of results obtained within the project "Practical School: Innovation in Higher Education and Success on the Labour Market", POSDRU/156/1.2/G/132920, implemented by the Romanian-American University in partnership with the Center for Legal Resources. The project is co-financed from the European Social Fund through the Sectorial Operational Programme Human Resource Development 2007-2013.

REFERENCES

- [1] Carter Eric, Lippert Eric, Visual Studio Tools for Office 2007: VSTO for Excel, Word, and Outlook, Publisher: Addison-Wesley Professional, 2009, ISBN-10: 0321533216, ISBN-13: 978-0321533210
- [2] Sempf Bill, Jausovec Peter, VSTO For Dummies, Publisher: For Dummies, 2010, ISBN-10: 0470046473, ISBN-13: 978-0470046470