Official Guide

Add-in Express™

Add-in Express for Office and VSTO

Getting Started

Add-in Express™ 2010 for Microsoft Office and VSTO

Revised on **6-Dec-11**

Table of Contents

Introduction

Add-in Express for VSTO is designed to simplify and speed up the development of Office add-ins as well as document-level customizations in Visual Studio Tools for Office (VSTO 2005 SE, VSTO 2008 and VSTO 2010). It provides a number of specialized components that allow the developer to jump through the interface-programming phase to the functional programming phase with a minimal loss of time.

System Requirements

Supported IDE Versions

- Visual Studio .NET 2005 Tools for Microsoft Office Second Edition
- Visual Studio 2008
- Visual Studio 2010

Host Applications

- Microsoft Outlook 2003 and higher
- Microsoft Excel 2003 and higher
- Microsoft Word 2003 and higher
- Microsoft PowerPoint 2003 and higher
- Microsoft Visio 2003 and higher
- Microsoft Project 2003 and higher
- Microsoft InfoPath 2007 and higher

Technical Support

Add-in Express is developed and supported by the Add-in Express Team, a branch of Add-in Express Ltd. You can obtain technical support using any of the following methods.

Resources of the Add-in Express web site (www.add-in-express.com):

- The HOWTOs section with sample projects that answer most common "how to" questions.
- ADX Toys, entire and "open sourced" add-ins for popular Office applications.
- Forums. We are actively participating in these forums. Really.

Also, you can e-mail us at support@add-in-express.com.

If you are a subscriber of our Premium Support Service, you can request technical support via an instant messenger, e.g. Windows/MSN Messenger or Skype.

Installing and Activating

There are two main points in the Add-in Express installation. First off, you have to specify the development environments in which you are going to use Add-in Express (see Supported IDE Versions). Second, you need to activate the product. What follows below is a brief guide on activation.

Activation Basics

During the activation process, the activation wizard prompts you to enter your license key. The key is a 30-character alphanumeric code shown in six groups of five characters each (for example, AXN4M-GBFTK-3UN78-MKF8G-T8GTY-NQS8R). Keep the license key in a safe location and do not share it with others. This license key forms the basis for your ability to use the software.

For purposes of product activation only, a non-unique hardware identifier is created from general information that is included in the system components. At no time are files on the hard drive scanned, nor is personally identifiable information of any kind used to create the hardware identifier. Product activation is completely anonymous. To ensure your privacy, the hardware identifier is created by what is known as a "one-way hash". To produce a one-way hash, information is processed through an algorithm to create a new alphanumeric string. It is impossible to calculate the original information from the resulting string.

Your license key and a hardware identifier are the only pieces of information required to activate the product. No other information is collected from your PC or sent to the activation server.

If you choose the *Automatic Activation* option of the activation wizard, the wizard attempts to establish an online connection to the activation server, www.activatenow.com. If the connection is established, the wizard sends both the license key and the hardware identifier over the Internet. The activation service generates an activation code using this information and sends it back to the activation wizard. The wizard saves the activation code to the registry.

If an online connection cannot be established (or you choose the *Manual Activation* option), you can activate the software using your web-browser. In this case, you will be prompted to enter the license key and a hardware identifier on a web page, and you will get an activation code. This process finishes with saving the activation code to the registry.

Activation is completely anonymous; no personally identifiable information is required. The activation code can be used to activate the product on that computer an unlimited number of times. However, if you need to install the product on several computers, you will need to perform the activation process again on every PC. Please refer to your end-user license agreement for information about the number of computers you can install the software on.

Setup Package Contents

The Add-in Express for VSTO setup program installs the following folders on your PC:

- Bin – Add-in Express binary files
- Docs – Add-in Express documentation including class reference
- Images – Add-in Express icons
- Redistributables – Add-in Express redistributable files
- Sources – Add-in Express source code (see the note below).

Please note that the source code of Add-in Express is or is not delivered depending on the product package you purchased. See the Feature matrix and prices page on our web site for details.

Add-in Express setup program installs the following text files on your PC:

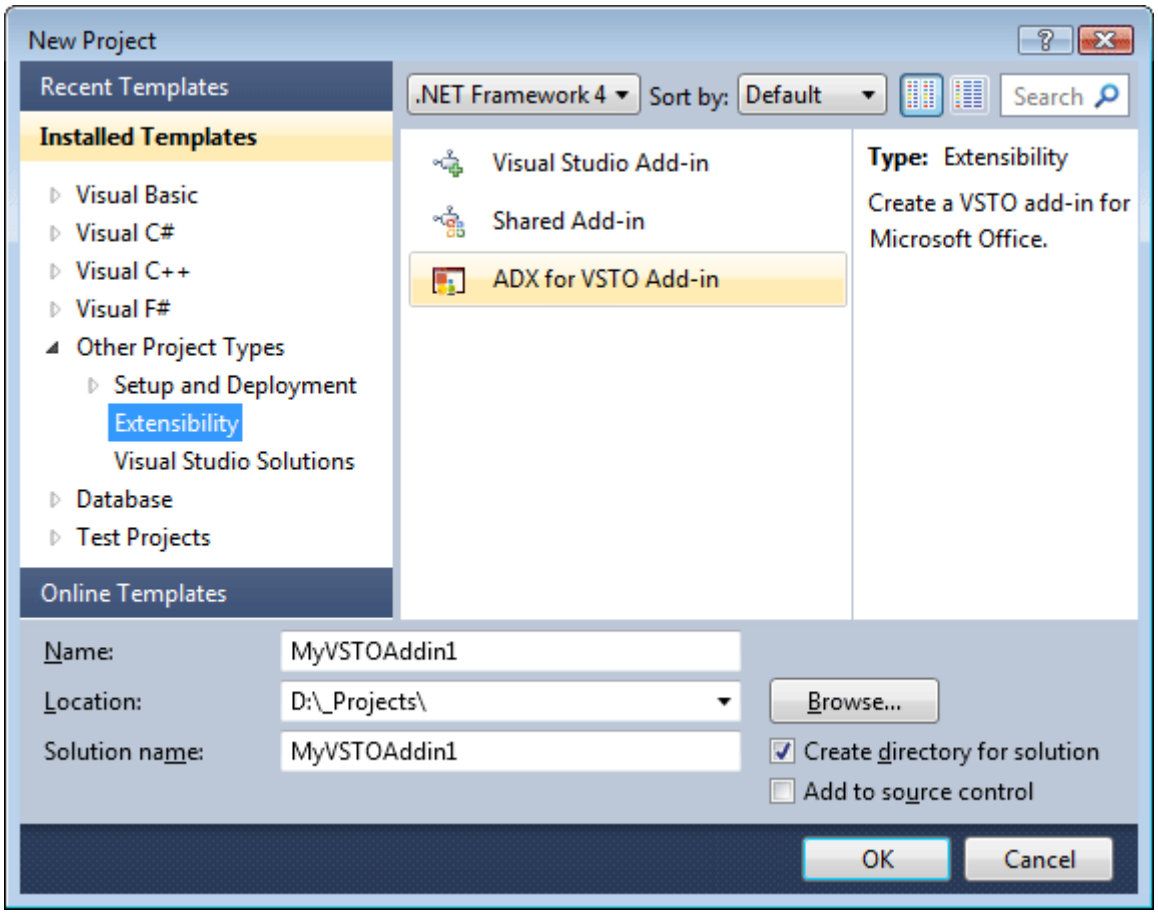
- licence.txt – EULA
- readme.txt – short description of the product, support addresses and such
- whatsnew.txt – this file describes the latest information on the product features added and bugs fixed.

Getting Started

Add-in Express is a development tool designed to simplify and speed up the development of add-ins for Microsoft Office in VSTO 2005-2010 through the consistent use of the RAD paradigm. It provides a number of specialized components allowing the developer to skip the interface-programming phase and get to functional programming in no time.

Creating Add-in Projects

If you use VS 2008 or VS 2010, choose the Add-in Express for VSTO item in the New Project dialog:



Click OK to start the project wizard that allows selecting the programming language and the Office application of your add-in. It also allows generating the setup project.

If you use VS 2005, then after creating any new VSTO add-in project, your first step is to add the Add-in Express Module (also referred to as the add-in module) to your project using the Add New Item Dialog in Visual Studio.

The solution created in either of the ways above includes the following main parts:

- ADXModule.vb (or ADXModule.cs), the Add-in Express Module, also add-in module, the core part of the add-in project
- Ribbon.vb (or Ribbon.cs); the Ribbon designer available for Office 2007-2010 add-in projects created in VS 2008 and 2010. For solutions targeting Office 2003 (and 2007), Add-in Express provides Ribbon components of its own. See Office Ribbon Components
- adxregaddin.exe – the add-in registrator that simplifies the deployment of VSTO add-ins. See VSTO Deployment Support in Add-in Express

Add-in Express Module

The add-in module represents an add-in in the targeted Office application and centralizes all programming logics in one place. Its designer allows adding other Add-in Express components and setting their properties at design-time. It provides all events of the host application. It also supplies the OnStartupComplete and OnBeginShutdown events, so you can handle add-in startup and shutdown. There are also events related to Office 2007 task panes (OnTaskPaneBeforeCreate, OnTaskPaneBeforeShow, OnTaskPaneBeforeDestroy, etc), and the Ribbon UI (OnRibbonBeforeCreate, OnRibbonBeforeLoad, OnRibbonLoaded). For Outlook add-ins, you can specify pages for the Tools | Options and Folder Properties dialogs (see Outlook Property Page).

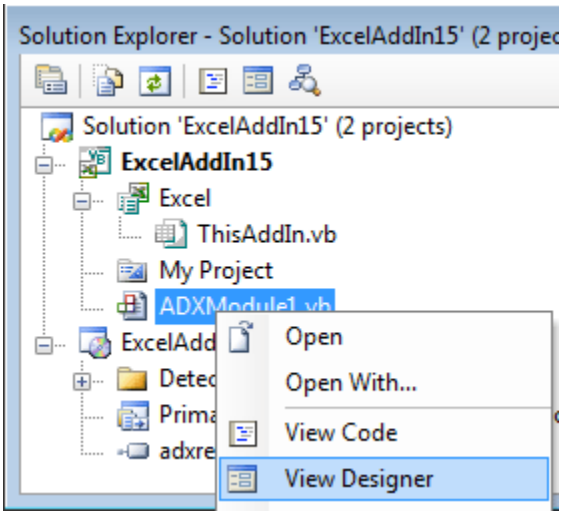
See the following chapters describing the Add-in Express components that you can add onto the add-in module: Office Ribbon Components, Command Bars: Toolbars, Menus, and Context Menus, Built-in Control Connector, Keyboard Shortcut, Outlook Bar Shortcut Manager.

You can also add custom task panes in Outlook, Excel, Word, and PowerPoint, versions 2003-2010. See Advanced Custom Task Panes in Office 2003-2010.

Add-in Express for VSTO components

Adding Components to the Add-in Module

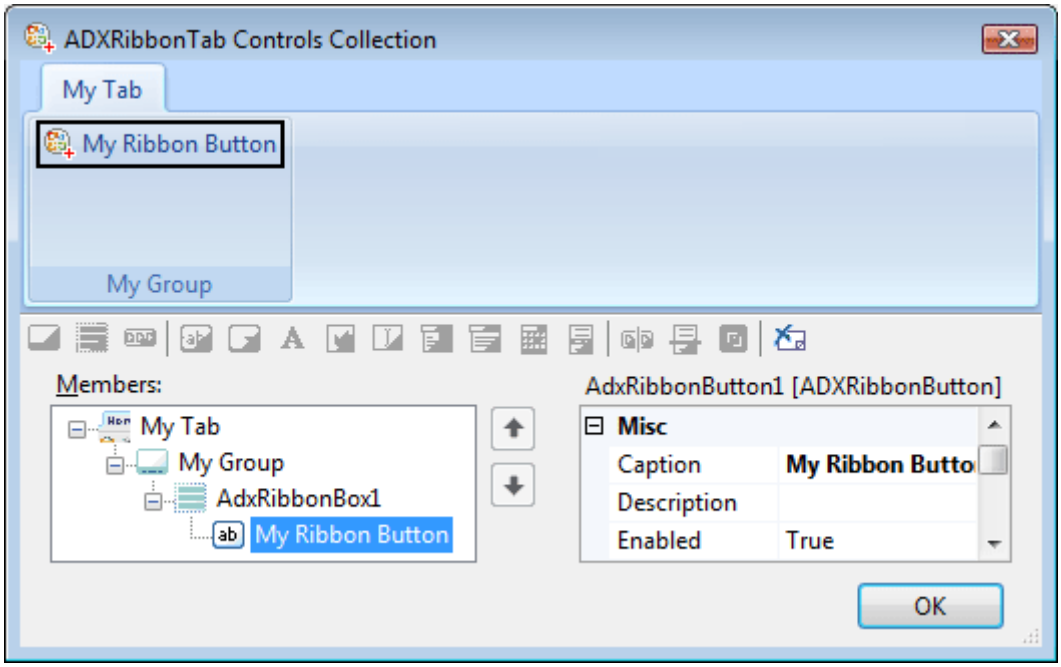
To add any Add-in Express component onto the add-in module, activate the module designer window and use commands available either in the Properties window or in the context menu (see also Add-in Module Commands). To activate the designer window, right-click the module in Solution Explorer and choose View Designer in the context menu.



Office Ribbon Components

Office 2007 presented a new Ribbon user interface. Microsoft states that the interface makes it easier and quicker for users to achieve the wanted results. The developers extend this interface by using the XML markup that the add-in should return to the host through an appropriate COM interface.

Add-in Express supports the Ribbon designer provided by VS 2008 and 2010 for Office 2007-2010 add-ins. If you have such a designer in your add-in project, then any of the Add-in Express Ribbon controls below will not be considered when generating the markup. The Add-in Express Ribbon controls are useful when creating the Ribbon interface for an add-in targeting Office 2003 and up; when such add-in is run in Office 2007-2010, Add-in Express generates the XML markup automatically.



Add-in Express provides some 50 Ribbon components that undertake the task of creating the markup. Also, there are 5 visual designers that allow creating the Ribbon UI of your add-in: Ribbon Tab (ADXRibbonTab), Ribbon Office Menu (ADXRibbonOfficeMenu), Quick Access Toolbar (ADXRibbonQuickAccessToolbar), Ribbon BackstageView (ADXBackStageView), and Ribbon Context Menu (ADXRibbonContextMenu).

In Office 2010, Microsoft abandoned the Office Button (introduced in Office 2007) in favor of the File Tab (Backstage View). To provide some sort of compatibility for you, ADXRibbonOfficeMenu will map your controls to the File tab **unless** you use ADXBackStageView components in your project; otherwise, all the controls you add to ADXRibbonOfficeMenu are ignored when Office 2010 loads your add-in.

Microsoft require developers to use the StartFromScratch parameter (see the StartFromScratch property of the add-in module) when customizing the Quick Access Toolbar.

Office Custom Task Panes

Add-in Express supports custom task panes by equipping the Add-in module with the TaskPanes property. Add a UserControl to your project, add an item to the TaskPanes collection, and set up the item by choosing the control in the ControlProgId property and filling in the Title property. Add your reaction to the TaskPaneXXX event series of the Add-in module and the DockPositionStateChange and VisibleStateChange events of the task pane. See also Office Custom Task Panes.

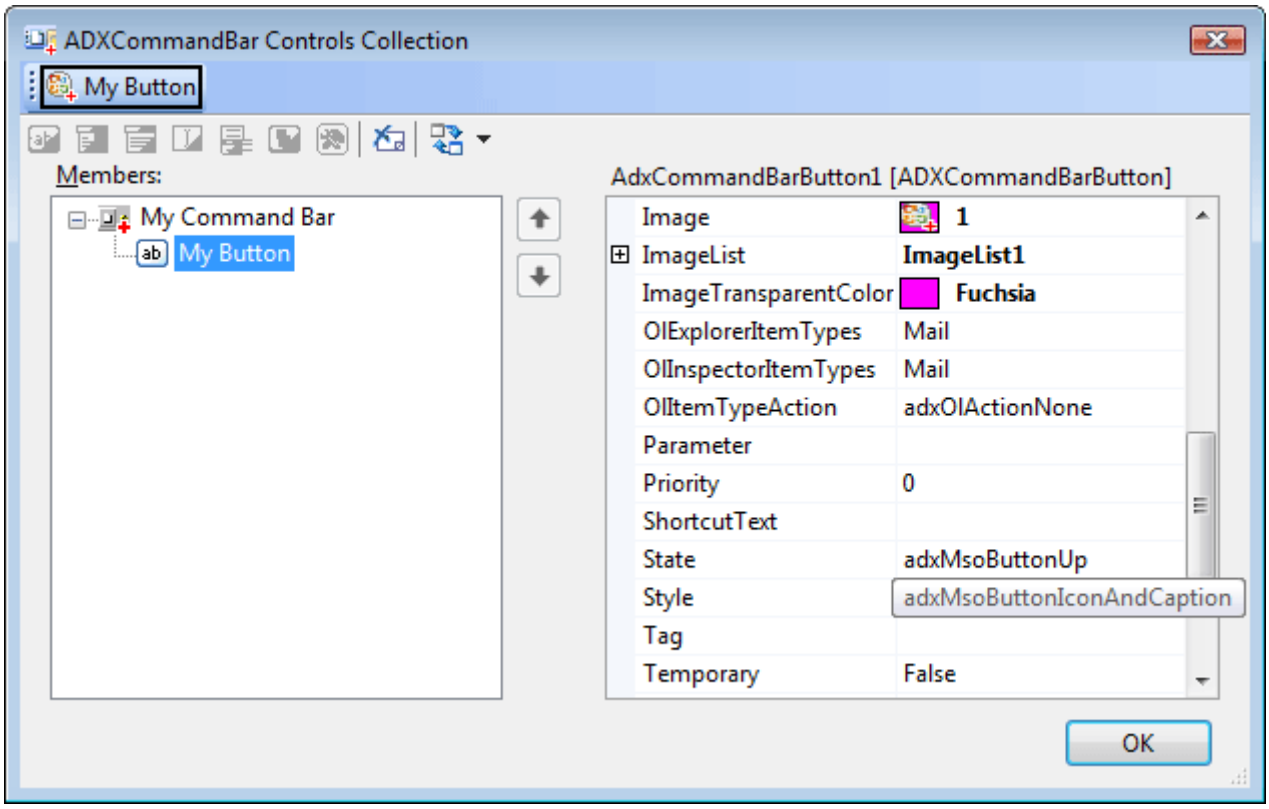
Advanced Custom Task Panes in Office 2003-2010

Add-in Express provides the technology to show custom task panes in Outlook, Excel, Word and PowerPoint, versions 2003-2010. See Advanced Custom Task Panes for details.

Command Bars: Toolbars, Menus, and Context Menus

Microsoft Office 2000-2003 supplied us with a common term for Office toolbars, menus, and context menus. This term is "command bar". Add-in Express provides toolbar, menu, and context menu components that allow tuning up targeted command bars at design-time. There are also Outlook-specific versions of these components. Every such component provides a visual designer available in the Controls property of the component. The screenshot below shows the visual designer for the toolbar component that creates a custom toolbar with a button. Note that this screenshot was captured when creating a sample project described in Your First Microsoft Office Add-in.

You can still use command bar controls in Office 2007-2010 add-ins. To do this, set the UseForRibbon property of the command bar component to True. In this case, the controls will be added to the Add-ins tab of the host application's Ribbon UI.



Toolbar

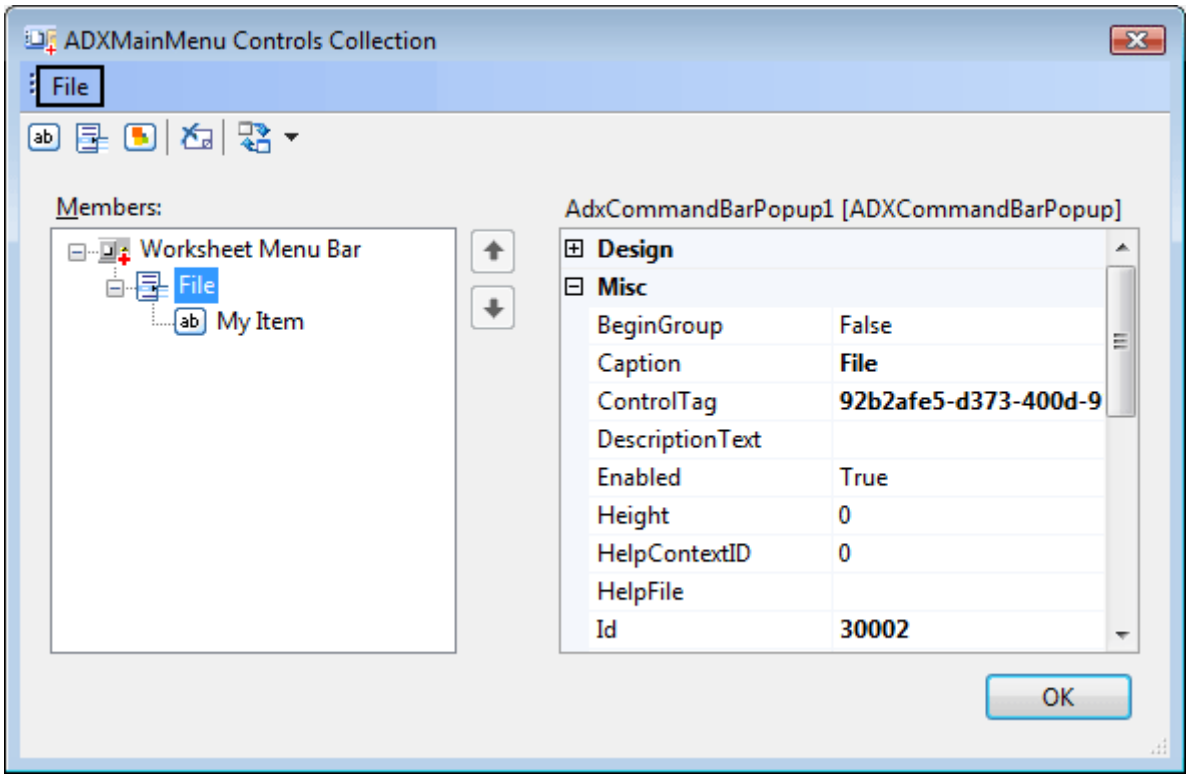
To add a toolbar to the host application, use the "Add CommandBar" command (see Adding Components to the Add-in Module and Add-in Module Commands). It adds an ADXCommandBar to the module. Its most important property is CommandBarName. If its value is equal to any built-in command bar of the host application, then you are connecting to a built-in command bar. Otherwise, you are creating a new command bar. To find out the built-in command bar names, use our free Built-in Controls Scanner utility.

To position your toolbar, use the Position property that allows docking your toolbar to the top, right, bottom, or left edges of the host application window. You can also leave your toolbar floating. For a fine positioning, you use the Left, Top, and RowIndex properties. To show a pre-2007 toolbar in the Add-ins tab in Office 2007-2010, set the UseForRibbon property of the corresponding command bar component to true.

To speed up add-in loading when connecting to an existing command bar, set the Temporary property to False. To make the host application remove the command bar when the host application quits, set the Temporary property to true. See also Temporary or Not?

Main Menu

By using the Add Main Menu command of the add-in module (see Adding Components to the Add-in Module and Add-in Module Commands), you add an ADXMainMenu.



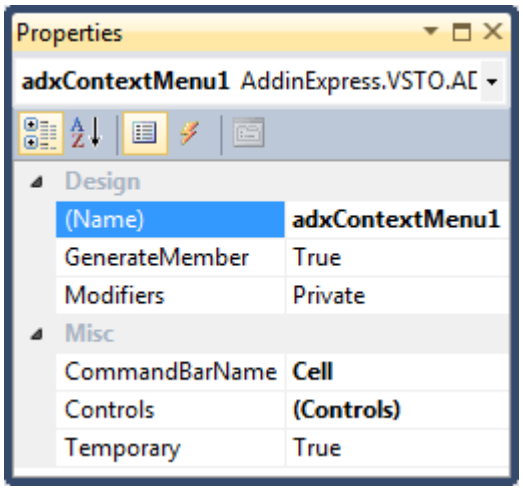
Like the toolbar component, it provides a visual designer for the Controls property. To add a custom top-level menu item, just add a popup control to the command bar. Then you can populate it with other controls. Note, however, that for all menu components, the controls can be buttons and pop-ups only. To add a custom button to a built-in top-level menu item, you specify the ID of the top-level menu item in the Id property of the button control. For instance, the ID of the File menu item in all Office applications is 30002. See more details about IDs of command bar controls in Using Existing Command Bar Controls. In main applications of Office 2007, they replaced the command system with the Ribbon UI. Therefore, instead of adding custom items to the main menu, you need to add them to a custom or built-in Ribbon tab. Also, you can add custom items to the menu of the Office Button in Office 2007. In Office 2010, they abandoned the Office button in favor of the File Tab, also known as Backstage View. Add-in Express provides components allowing customizing both the File Tab and the Ribbon Office Menu, see Step #10 – Customizing the Ribbon User Interface in Your First Microsoft Office Add-in. Note, if you customize the Office Button menu only, Add-in Express will map your controls to the Backstage View when the add-in is run in Office 2010. If, however, both Office Button menu and File tab are customized at the same time, Add-in Express ignores custom controls you add to the Office Button menu.

Context Menu

In Office 2003-2007, context menus are command bars and they can be customized in the same way as any other command bar. In Office 2010, they allow us to customize context menus via the Ribbon XML. Accordingly, Add-in Express provides two components: a commandbar-based (ADXContextMenu) and Ribbon-based (ADXRibbonContextMenu).

The PowerPoint development team explicitly states that PowerPoint 2007 doesn't support customizing context menus with command bar controls. However, some context menus in PowerPoint 2007 are still customizable in this way.

The Add ADXContextMenu command of the add-in module (see Adding Components to the Add-in Module and Add-in Module Commands) adds an ADXContextMenu, which allows adding a custom command bar control to any context menu available in all Office 2003-2007 applications. To specify the context menu you want to connect to, just choose the name of the context menu in the CommandBarName combo.



Please note that the context menu names for this property were taken from Office 2007, the last Office version that introduced **new** commandbar-based context menus. That is, it is possible that the targeted context menu is not available in Office 2003.

In Office 2010 and higher, you can customize both commandbar-based and Ribbon-based context menus. See Step #8 – Customizing Context Menus and Step #9 – Customizing Context Menus in Outlook.

Outlook Toolbars and Main Menus

While the look-n-feel of all Office toolbars is the same, Outlook toolbars differ from toolbars of other Office applications. They are different for the two main Outlook window types – for Outlook Explorer and Outlook Inspector windows. Accordingly, Add-in Express provides you with Outlook-specific command bar components that work correctly in multiple Explorer and Inspector windows scenarios: ADXOlExplorerCommandBar and ADXOlInspectorCommandBar. In the same way, Add-in Express provides Outlook-specific versions of the Main Menu component: ADXOlExplorerMainMenu and ADXOlInspectorMainMenu. See also Adding Components to the Add-in Module and Add-in Module Commands.

All of the components above provide the FolderName, FolderNames, and ItemTypes properties that add context-sensitive features to the command bar. For instance, you can choose your toolbar to show up for e-mails only. To achieve this, just choose a correct checkbox in the ItemTypes property editor.

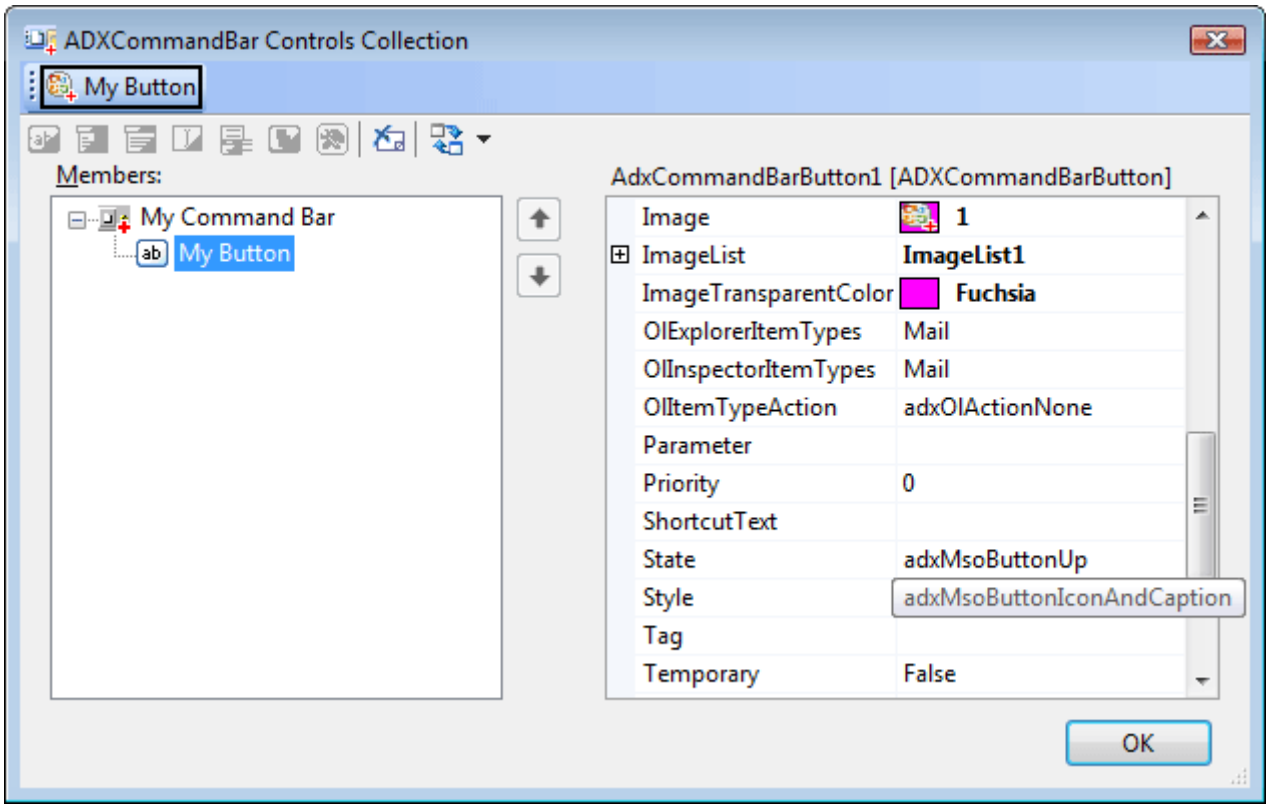
Connecting to Existing Command Bars

In Office, all command bars are identified by their names. Keeping it in mind, you can add a custom or built-in control to any existing command bar. The only thing you need to know is the command bar name. Use our free Built-in Controls Scanner to get the names of all command bars and controls existing in any Office application. Then you can specify any of the command bar names in the CommandBarName property of the appropriate command bar component.

Command Bar Controls

Command bar components provide the Controls property. Clicking it in the Properties window in Visual Studio, invokes the appropriate visual designer. On the screenshot below, you see the visual designer of the ADXCommandBar component.

Using the designer, you can populate your command bars with controls and set up their properties at design-time. At run-time, you use the Controls collection of your command bar. Every control (built-in and custom) added to this collection will be added to the corresponding toolbar at your add-in startup.



Command Bar Control Properties and Events

The main property of any command bar control (they descend from ADXCommandBarControl) is the Id property. To add a built-in control to your toolbar, specify its ID in the corresponding property of the command bar control component. To find out the ID of every built-in control in any Office application, use our free Built-in Controls Scanner utility. To add a custom control to the toolbar, leave the Id unchanged.

To add a separator before any given control, set its BeginGroup property to true.

Set up a control's appearance using a large number of its properties, such as Enabled and Visible, Style and State, Caption and ToolTipText, DropDownLines and DropDownWidth, etc. You also control the size (Height, Width) and location (Before, AfterId, and BeforeId) properties. To provide your command bar buttons with a default list of icons, drop an ImageList component onto the add-in module and specify the ImageList in the Images property of the module. Do not forget to set the button's Style property to either adxMsoButtonIconAndCaption or adxMsoButtonIcon.

Use the OlExplorerItemTypes, OlInspectorItemTypes, and OlItemTypesAction properties to add context-sensitivity to controls on Outlook-specific command bars. The OlItemTypesAction property specifies an action that Add-in Express will perform with the control when the current item's type coincides with that specified by you.

To handle user actions, use the Click event for buttons and the Change event for edit, combo box, and drop down list controls. Use also the DisableStandardAction property available for built-in buttons added to your command bar. To intercept events of any built-in command bar control, see Built-in Control Connector.

Command Bar Control Types

The Office Object Model contains the following control types available for **toolbars**: button, combo box, and pop-up. Using the correct property settings of the combo box component, you can extend the list with edits and dropdowns.

Nevertheless, this list is extremely short. Add-in Express allows extending this list with any .NET control (see Toolbar Controls for Microsoft Office).

Please note that due to the nature of command bars, **menu and context menu items** can only be buttons and pop-ups (item File in any main menu is a sample of a popup).

Using Existing Command Bar Controls

Add-in Express connects to built-in controls using the Id property. That is, if you specify the ID of a control not equal to 1 and this control exists on the specified command bar, Add-in Express connects to this control and

ignores all other properties. If the control is not found, Add-in Express creates it. Using this approach, you can override the standard behavior of a built-in button on a given toolbar:

- Add a new toolbar component to the module
- Specify the toolbar name in the CommandBarName property
- Add an ADXCommandBarButton to the command bar
- Specify the ID of the built-in button in the ADXCommandBarButton.Id property
- Set DisableStandardAction to true
- Now you should handle the Click event of the button

You can find the IDs using the free Built-in Controls Scanner utility. Download it at http://www.add-in-express.com/downloads/controls-scanner.php.

Built-in Control Connector

Built-in controls of an Office application have predefined IDs. You find the IDs using the free Built-in Controls Scanner utility.

The Built-in Control Connector component allows overriding the standard action for any built-in control without adding it onto any command bar.

Add a built-in control connector onto the module. Set its Id property to the ID of your command bar control. To connect the component to all instances of the command bar control having this ID, leave its CommandBar property empty. To connect the component to the control on a given toolbar, specify the toolbar in the CommandBar property. To override the default action of the control, use the Action event. The component traces the context and when any change happens, it reconnects to the currently active instance of the command bar control with the given Id, taking this task away from you.

To add a built-in control onto your command bar, see Command Bar Control Properties and Events.

Keyboard Shortcut

Every Office application provides built-in keyboard combinations that allow shortening the access path for commands, features, and options of the application. Add-in Express allows adding custom keyboard combinations and processing both custom and built-in ones.

Add a Keyboard Shortcut component onto the add-in module, choose or specify the keyboard shortcut you need in the ShortcutText property, set the HandleShortCuts property of the module to true and process the Action event of the component.

Outlook Bar Shortcut Manager

Outlook provides us with the Outlook Bar (Navigation Pane in Outlook 2003). The Outlook Bar displays Shortcut groups consisting of Shortcuts that you can target a Microsoft Outlook folder, a file-system folder, or a file-system path or URL. You use the Outlook Bar Shortcut Manager to customize the Outlook Bar with your shortcuts and groups.

This component is available for ADXAddinModule. Use the Groups collection of the component to create a new shortcut group. Use the Shortcuts collection of a short group to create a new shortcut. To connect to an existing shortcut or shortcut group, set the Caption properties of the corresponding ADXOlBarShortcut and/or ADXOlBarGroup components equal to the caption of the existing shortcut or shortcut group. Please note that there is no other way to identify the group or shortcut.

That is why your shortcuts and shortcut groups must be named uniquely for Add-in Express to remove only the specified ones (and not those having the same names) when the add-in is uninstalled. If you have several groups (or shortcuts) with the same name, you will have to remove them yourself. Depending on the type of its value, the Target property of the ADXOlBarShortcut component allows you to specify different shortcut types. If the type is Outlook.MAPIFolder, the shortcut represents a Microsoft Outlook folder. If the type is String, the shortcut represents a file-system path or a URL. No events are available for these components.

Outlook Property Page

Outlook allows extending its Options dialog with custom pages. You see this dialog when you choose Tools | Options menu. In addition, Outlook allows adding such page to the Folder Properties dialog. You see this dialog when you choose the Properties item in the folder context menu. You create such pages using the Outlook Property Page component.

In the Add New Item Dialog, choose the Outlook Options Page item to add a class to your project. This class is a descendant of the System.Windows.Forms.UserControl class. It allows creating Outlook property pages using its visual designer. Just set up the property page properties, place your controls onto the page, and add your code. To add this page to the Outlook Options dialog, select the name of your control class in the PageType combo of ADXAddinModule and enter some characters into the PageTitle property.

To add a page to the Folder Properties dialog for a given folder(s), you use the FolderPages collection of the add-in module. Run its property editor and add an item (of the ADXOlFolderPage type). You connect the item to a given property page through the PageType property. Note, the FolderName, FolderNames, and ItemTypes properties of the ADXOlFolderPage component work in the same way as those of Outlook-specific command-bars.

Specify reactions required by your business logics in the Apply and Dirty event handlers. Use the OnStatusChange method to raise the Dirty event, the parameters of which allow marking the page as Dirty.

Event Classes

Outlook and Excel differ from other Office applications because they have event-raising objects not only at the topmost level of their object models. These exceptions are the Folders and Items classes as well as all item types in Outlook, and the Worksheet class in Excel. Naturally, you need to handle events from these sources. Add-in Express event classes provide you with version independent components that ease the pain of handling such events. The event classes also handle releasing of COM objects required for their functioning.

At design-time, you add an event class to the project (see Add New Item Dialog) and use its event procedures to write the code for just one set of event handling rules for a given event source type, say, for the Items collection of the MAPIFolder class in Outlook 2003; in Outlook 2007, you can also use the Folder class. To implement another set of event handling rules for the same event source type, you add another event class to your project.

At run-time, you connect an event class instance to an event source using its ConnectTo method. To disconnect the event class from the event source you use the RemoveConnection method. To apply the same business rules to another event source of the same type (say, to items of another folder), you create a new instance of the same event class.

What follows below is the source code of a newly added event class that processes the events of the Items collection of the MAPIFolder class in Outlook (also the Folder class in Outlook 2007).

```
Imports System
'Add-in Express for VSTO
'Outlook Items Events Class
Public Class OutlookItemsEventsClass1
    Inherits AddinExpress.VSTO.ADXOutlookItemsEvents

    Public Sub New(ByVal ADXModule As AddinExpress.VSTO.ADXOutlookModule)
        MyBase.New(ADXModule)
    End Sub

    Public Overrides Sub ProcessItemAdd(ByVal Item As Object)
      MsgBox("The item with subject '" + Item.Subject + _
         "' has been added to the Inbox folder")
   End Sub

    Public Overrides Sub ProcessItemChange(ByVal Item As Object)
        'TODO: Add some code
    End Sub

    Public Overrides Sub ProcessItemRemove()
        'TODO: Add some code
    End Sub
End Class
```

Advanced Custom Task Panes

Add-in Express allows COM add-ins to show custom panes in Outlook, Excel, Word, and PowerPoint, versions 2003-2007.

An Absolute Must-Know

Here are the three main points you should know about:

- there are application-specific <Manager> components; every <Manager> component provides a collection; each <Item> from the collection binds a <Form> (an application-specific descendant of System.Windows.Forms.Form) to the visualization and context (Outlook-only) settings;
- you **never** create an instance of a <Form> in the way you create an instance of System.Windows.Forms.Form; instead, the <Manager> creates instances of the <Form> for you; the instances are created either automatically or at your request;
- the Visible property of a <Form> instance is true, when the instance is embedded into a window region (as specified by the visualization settings) regardless of the actual visibility of the instance; the Active property of the <Form> instance is true, when the instance is shown on top of all other instances in the same region.

A required comment

Anywhere in this section, a term in angle brackets, such as <Manager> or <Form> above, specifies a component, class, or class member, the actual name of which is application-dependent. Every such term is covered in the corresponding chapter of this manual.

Hello, World!

Adding custom panes in a particular application is described in appropriate parts of the following samples:

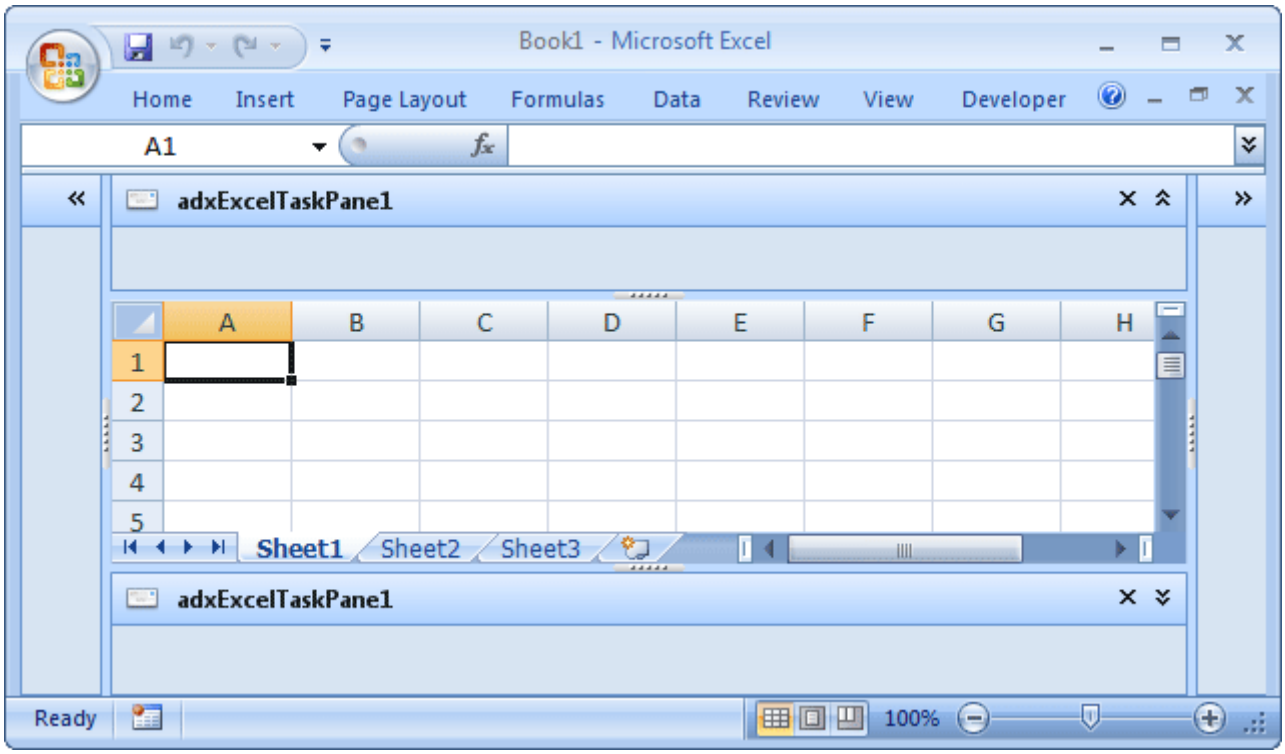
- Outlook – in Your First Microsoft Outlook Add-in see Step #10 – Adding a Custom Task Pane in Outlook 2003-2010
- Excel – in Your First Microsoft Office Add-in, see Step #11 – Adding Custom Task Panes in Office 2003-2010

The Regions

Obviously, all Office applications have different window structures. Accordingly, Add-in Express provides a number of application-specific options for embedding your forms.

Word, Excel and PowerPoint Regions

These Office applications allow showing your forms in four regions; the regions are docked to the four edges of the application's main window. The names of the regions are Left, Top, Right, and Bottom (see the Position property of the <Item>).

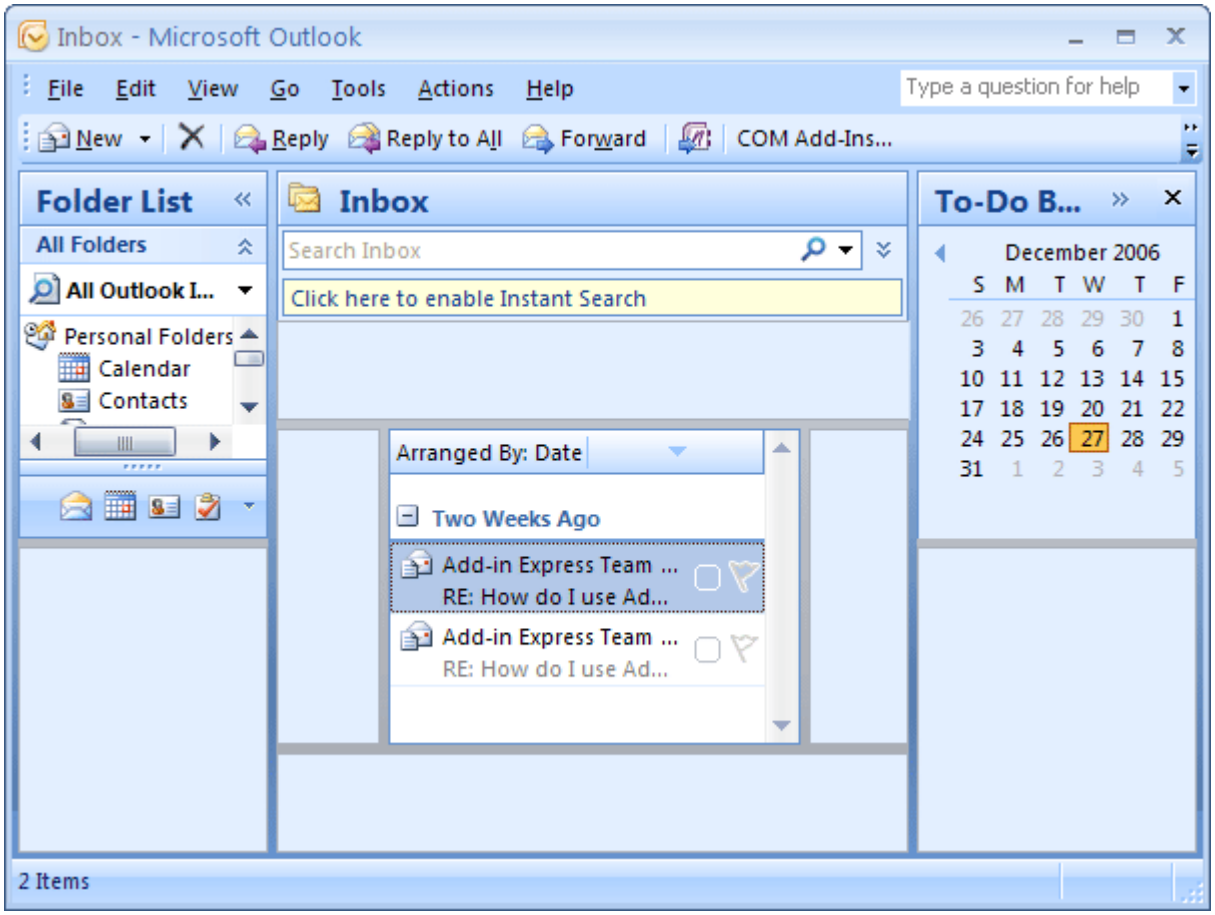


Outlook Regions

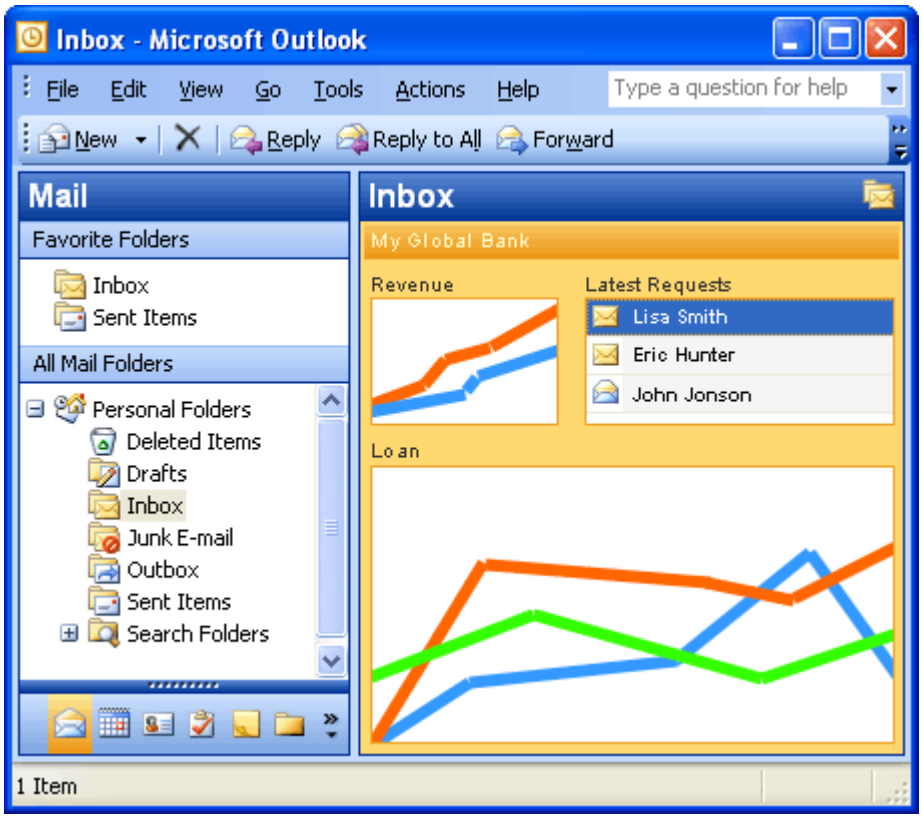
Outlook regions are specified in the ExplorerLayout and InspectorLayout properties of the item (= ADXOlFormsCollectionItem). Note that you must also specify the item's ExplorerItemTypes and/or InspectorItemTypes properties; otherwise, the form (an instance of ADXOlForm) will never be shown. Here is the list of Outlook regions:

- Four regions around the list of mails, tasks, contacts etc. The region names are LeftSubpane, TopSubpane, RightSubpane, BottomSubpane (see the screenshot below)
- One region below the Navigation Pane – BottomNavigationPane (see the screenshot below)

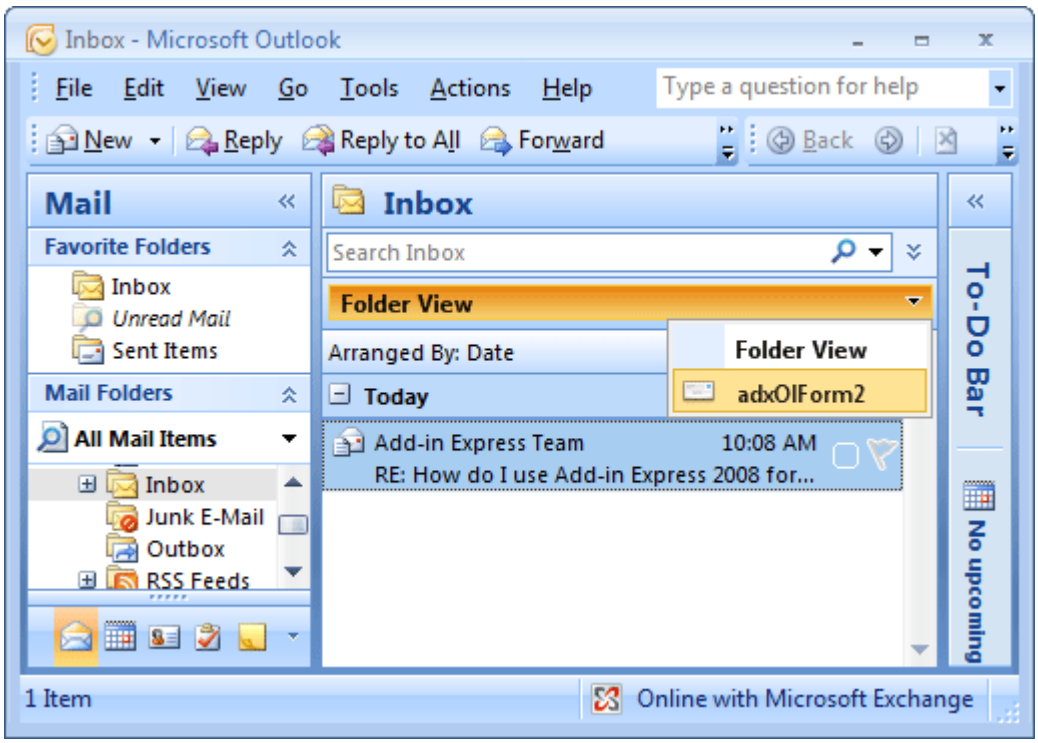
- One region below the To-Do Bar – BottomTodoBar (see the screenshot below)



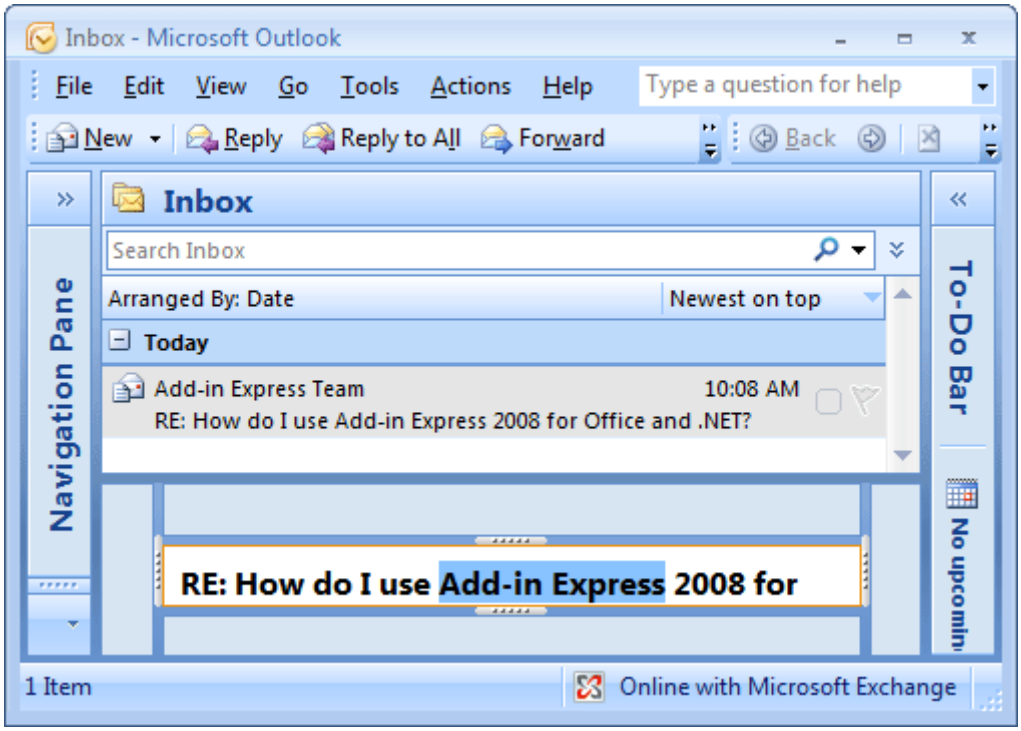
- The WebViewPane region (see the screenshot below). Note that it uses Outlook properties in order to replace the items grid with your form (see also WebViewPane).



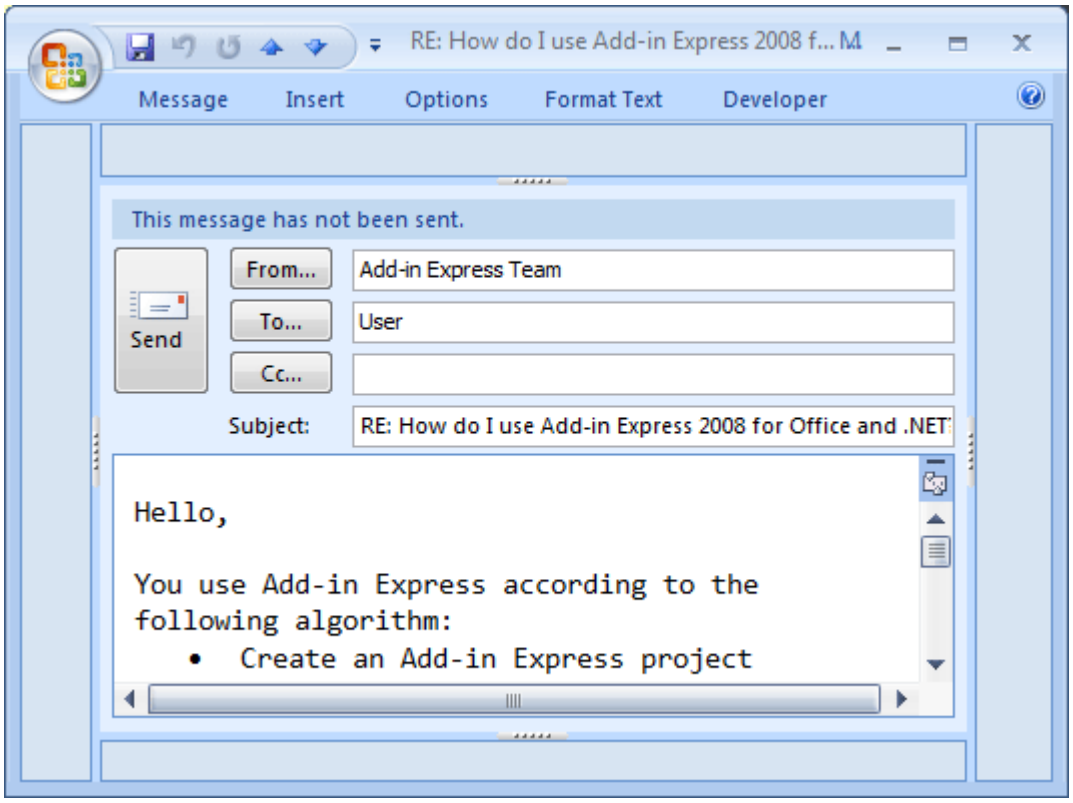
- The FolderView region. Unlike WebViewPane, it allows the user to switch between the original Outlook view and your form (see the screenshot below).



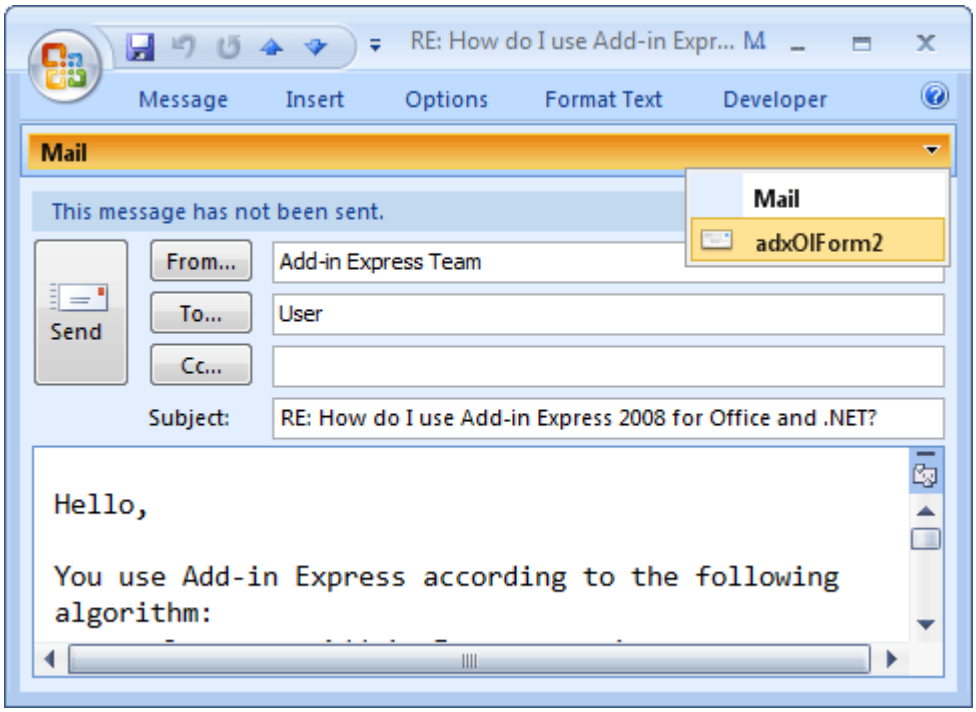
- Four regions around the Reading Pane – LeftReadingPane, TopReadingPane, RightReadingPane, BottomReadingPane (see the screenshot below)



- Four regions around the body of an e-mail, task, contact, etc. The region names are LeftSubpane, TopSubpane, RightSubpane, BottomSubpane (see the screenshot below)



- The InspectorRegion region (see the screenshot below)

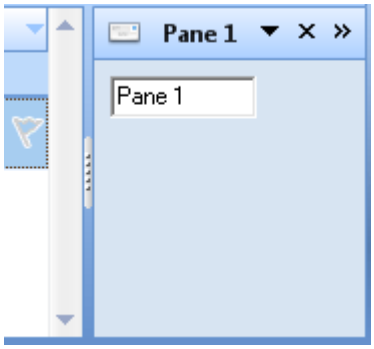


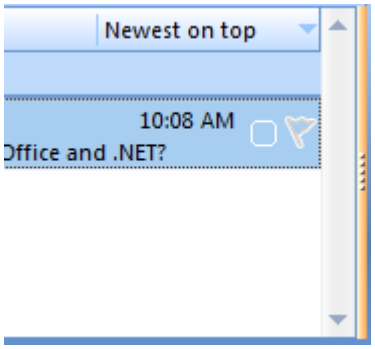
The UI Mechanics

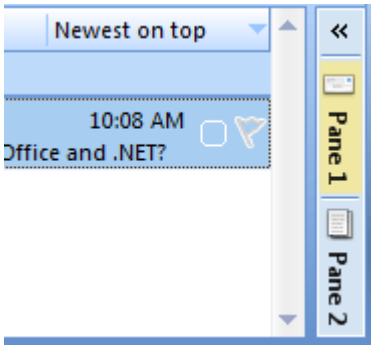
The UI, Related Properties and Events

As mentioned in An Absolute Must-Know, the <Manager> creates instances of the <Form>. An instance of the <Form> (further on it is referenced as form) is considered visible if it is embedded into a region. The form may be actually invisible either due to the region state (see below) or because other forms in the same region hide it; anyway, in this case, <Form>.Visible returns true. To prevent embedding the form into a region, you can cancel an event named ADXBeforeFormShow in Outlook, ADXBeforeTaskPaneShow in Excel, Word, and PowerPoint. When the form is shown in a region, the Activated event occurs and <Form>.Active becomes true. When the user moves the focus onto the form, the <Form> generates the ADXEnter event. When the form loses focus, the ADXLeave event occurs. When the form becomes invisible (actually), it generates the Deactivate event. When the corresponding <Manager> removes the form from its region, <Form>.Visible becomes false and the form generates the ADXAfterFormHide event in Outlook, ADXAfterTaskPaneHide event in Excel, Word, and PowerPoint.

The form may be initially shown in any of the following region states: normal, hidden (collapsed to a 5px wide strip), minimized (reduced to the size of the form caption).







These states are reflected in the corresponding values of the DefaultRegionState property of the <Item> - Hidden, Normal and Minimized.

When the region is in the hidden state, the user can click on the splitter and the region will go to the normal state.

When the region is in the normal state, the user can choose any of the options below:

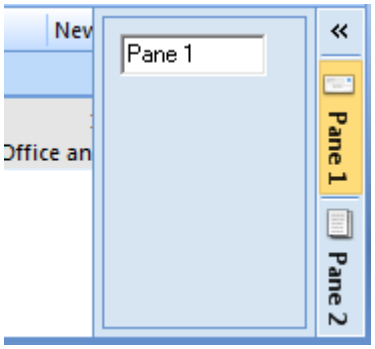
- change the region size by moving the splitter; this raises size-related events of the form
- hide the form by clicking on the "dotted" mini-button or by double-clicking anywhere on the splitter; this fires the Deactivate event of the <Form>
- close the form by clicking on the Close button in the form header; this fires the ADXCloseButtonClick event of the <Form>. The event is cancellable; if the event isn't cancelled, the Deactivate event occurs,

then the pane is being deleted from the region (<Form>.Visible = false) and finally, the <ADXAfterFormHide> event of the <Form> occurs

- show another form by clicking the header and choosing an appropriate item in the popup menu; this fires the Deactivate event on the first form and the Activated event on the second form
- transfer the region to the minimized state by clicking the arrow in the right corner of the form header; this fires the Deactivate event of the form.

When the region is in the minimized state, the user can choose either of the two options below:

- return the region to the normal state by clicking the arrow at the top of the slim profile of the form region; this raises the Activated event of the form and changes the Active property of the form to true
- expand the form itself by clicking on the form's button; this opens the form so that it overlaps a part of the Outlook window near the form region; this also raises the Activated event of the form and sets the Active property of the form to true.
- drag an Outlook item, Excel chart, file, selected text, etc onto the form button; this fires the ADXDragOverMinimized event of the form; the event allows you to check the object being dragged and to decide if the form should be restored.



The Close Button and the Header

The Close button is shown if the CloseButton property of the <Item> is true. The header is always shown when there are two or more forms in the same region. When there is just one form in a region, the header is shown only if the AlwaysShowHeader property of the <Item> is true.

Clicking on the Close button in the form header fires the ADXCloseButtonClick event of the <Form>, the event is cancellable:

```
Private Sub ADXOlForm1_ADXCloseButtonClick(ByVal sender As System.Object, _
    ByVal e As AddinExpress.OL.ADXOlForm.ADXCloseButtonClickEventArgs) _
    Handles MyBase.ADXCloseButtonClick
    e.CloseForm = False
End Sub
```

You can create a Ribbon or command bar button that allows the user to show the form that was previously hidden.

Showing/Hiding Form Instances Programmatically

In Excel and PowerPoint, a single instance of the <Form> is always created for a given <Item> because these applications show documents in a single main window. On the contrary, Word is an application that **normally** shows multiple windows, and in this situation, the Word Task Panes Manager creates one instance of the pane for every document opened in Word.

Outlook is a specific host application. It shows several instances of two window types simultaneously. In addition, the user can navigate through the folder tree and select, create and read several Outlook item types. Accordingly, an ADXOlFormsCollectionItem can generate and show several instances of ADXOlForm at the same time. Find more details on managing custom panes in Outlook in Advanced Outlook Regions.

To access the form, which is currently active in Excel or PowerPoint, you use the TaskPaneInstance property of the <Item>; in Word, the property name is CurrentTaskPaneInstance; in Outlook it is the GetCurrentForm method. To access all instances of the <Form> in Word, you use the TaskPaneInstances property of ADXWordTaskPanesCollectionItem; in Outlook, you use the FormInstances method of ADXOlFormsCollectionItem. Note that in Excel and PowerPoint an only instance of the <Form> is always created for a given <Item>.

By setting the Enabled property of an <Item> to false, you delete all form instances created for that <Item>. To hide any given form (i.e. to remove it from the region), call its Hide method.

You can check that a form isn't available in the UI (say, you cancelled the <BeforInstanceCreate> or <BeforeFormShow> events or the user closed it) by checking the Visible property of the form:

```
Dim Pane As ADXWordTaskPane1 = _
    TryCast(Me.AdxWordTaskPanesCollectionItem1.CurrentTaskPaneInstance, _
        ADXWordTaskPane1)
Dim DoesPaneExist As Boolean
If Pane IsNot Nothing Then
    DoesPaneExist = Pane.Visible
Else
    DoesPaneExist = False
End If
```

If the form isn't available in the UI, you can show such a form in one step:

- for Outlook, you call the ApplyTo method of the <Item>; the method accepts the parameter, which is either Outlook.Explorer or Outlook.Inspector;
- for Excel, Word, and PowerPoint, you call the ShowTaskPane method of the <Item>

The methods above also transfer the region that shows the form to the normal state.

If the Active property of your form is false, that is if your form is hidden by other forms in the region, then you can call the Activate method of the <Form> to show the form on top of all other forms in that region. Note that if the region was in either minimized or hidden state, calling Activate will also transfer it to the normal state.

Note that your form doesn't restore its Active state in subsequent sessions of the host application in regions showing several forms. In other words, if several add-ins show several forms in the same region and the current session finishes with a given form on top of all other forms in that region, the subsequent start of the host application may show some other form as active. This is because events are given to add-ins in an unpredictable order. When dealing with several forms of a given add-in, they are created in the order determined by their locations in the <Items> collection of the <Manager>.

In Outlook, due to context-sensitivity features provided by the <Item>, an instance of your form will be created whenever the current context matches that specified by the corresponding <Item>.

Resizing the Forms

There are two values of the Splitter property of the <Item>. The default one is Standard. This value shows the splitter allowing the user to change the form size as required. The form size is stored in the registry so that the size is restored whenever the user starts the host application.

You can only resize your form programmatically, if you set the Splitter property to None. Of course, no splitter will be shown in this case. Changing the Splitter property programmatically doesn't affect a form currently loaded into its region (that is, having Visible = true). Instead, it will be applied to any newly shown form.

If the form is shown in a given region for the first time and no forms were ever shown in this region, the form will be shown using the appropriate dimensions that you set at design-time. On subsequent host application sessions, the form will be shown using the dimensions set by the user.

Tuning the Settings at Run-Time

To add/remove an <Item> to/from the collection and to customize the properties of an <Item> at add-in start-up, you use the <Initialize> event of the <Manager>; the event's name is OnInitialize for Outlook and ADXInitalize for Excel, Word and PowerPoint.

Changing the Enable, Cached (Outlook only), <FormClassName> properties at run-time deletes all form instances created by the <Item>.

Changing the InspectorItemTypes, ExplorerItemTypes, ExplorerMessageClasses, ExplorerMessageClass, InspectorMessageClasses, InspectorMessageClass, FolderNames, FolderName properties of the ADXOlFormsCollectionItem deletes all non-visible form instances.

Changing the <Position> property of the <Item> changes the position for all visible form instances.

Changing the Splitter and Tag properties of the <Item> doesn't do anything for the currently visible form instances. You will see the splitter changed when the <Manager> shows a new instance of the <Form>.

Excel Task Panes

Please see The UI Mechanics above for the detailed description of how Add-in Express panes work. Below you see the list containing some generic terms mentioned in An Absolute Must-Know and their Excel-specific equivalents:

- <Manager> - AddinExpress.XL.ADXExcelTaskPanesManager, the Excel Task Panes Manager
- <Item> - AddinExpress.XL.ADXExcelTaskPanesCollectionItem
- <Form> - AddinExpress.XL.ADXExcelTaskPane

Application-specific features

ADXExcelTaskPane provides useful events unavailable in the Excel object model: ADXBeforeCellEdit and ADXAfterCellEdit.

Keyboard and Focus

ADXExcelTaskPane provides the ADXKeyFilter event. It deals with the feature of Excel that captures the focus if a key combination handled by Excel is pressed. By default, Add-in Express panes do not pass key combinations to Excel. In this way, you can be sure that the focus will never leave the pane unexpectedly.

Just to understand that Excel feature, imagine that you need to let the user press Ctrl+S and get the workbook saved while your pane is focused. In such a scenario, you have two ways:

- You process the key combination in the code of the pane and use the Excel object model to save the workbook.
- Or, you send this key combination to Excel using the ADXKeyFilter event.

Besides the obvious difference between the two ways above, the former leaves the focus on your pane while the latter effectively moves it to Excel because of the focus-capturing feature just mentioned.

The algorithm of key processing is as follows. Whenever a single key is pressed, it is sent to the pane. When a key combination is pressed, ADXExcelTaskPane determines if the combination is a shortcut on the pane. If it is, the keystroke is sent to the pane. If it isn't, ADXKeyFilter is fired and the key combination is passed to the event handler. Then the event handler specifies whether to send the keystroke to Excel or to the pane. The latter is the default behavior. Note that sending the key combination to Excel will result in moving the focus off the pane. The above-said implies that the ADXKeyFilter event never fires for shortcuts on the pane's controls.

Also, ADXKeyFilter is never fired for hot keys (Alt + an alphanumeric symbol). If ADXExcelTaskPane determines that the pane cannot process the hot key, it sends the hot key to Excel, which activates its main menu. After the user has navigated through the menu by pressing arrow buttons, Esc, and other hot keys, opened and closed Excel dialogs, ADXExcelTaskPane will get focus again.

Wait a Little and Focus Again

The pane provides a simple infrastructure that allows implementing the Wait a Little schema - see the ADXPostMessage method and the ADXPostMessageReceived event.

Currently we know at least one situation when this trick is required. Imagine that you show a pane and you need to set the focus on a control on the pane. It isn't a problem in, say the Activated event. Nevertheless, it is useless because Excel, continuing its initialization, moves the focus off the pane. With the above-said method and event, you can make your pane look like it never loses focus: in the Activated event handler, you call the ADXPostMessage method specifying a unique message ID and, in the ADXPostMessageReceived event, you filter incoming messages. When you get the appropriate message, you set the focus on the control. Beware, there will be a huge lot of inappropriate messages.

Advanced Outlook Regions

Please see The UI Mechanics above for the detailed description of how Add-in Express panes work. Below you see the list containing some generic terms mentioned in An Absolute Must-Know and their Excel-specific equivalents:

- <Manager> - AddinExpress.OL.ADXOlFormsManager, the Outlook Forms Manager
- <Item> - AddinExpress.OL.ADXOlFormsCollectionItem
- <Form> - AddinExpress.OL.ADXOlForm

Context-Sensitivity of Your Outlook Form

Whenever the Outlook Forms Manager detects a context change in Outlook, it searches the ADXOlFormsCollection collection for enabled items that match the current context and, if any match is found, it shows or creates the corresponding instances.

ADXOlFormsCollectionItem provides a number of properties that allow specifying the context settings for your form. Say, you can specify **item types** for which your form will be shown. Note that in case of explorer, the item types that you specify are compared with the default item type of the current folder. In addition, you can specify **the names of the folders** for which your form will be shown in the FolderName and FolderNames properties; these properties also work for Inspector windows – in this case the parent folder of the Outlook item is checked. A special value in FolderName is an asterisk ('*'), which means "all folders". See also Outlook Add-ins – Template Characters in FolderName. Also, you can specify **message class(es)** for which your form will be

shown. Note that all context-sensitivity properties of an ADXOlFormsCollectionItem are treated using the **OR** boolean operation.

In advanced scenarios, you can also use the ADXOlFormsCollectionItem.ADXBeforeFormInstanceCreate and ADXOlForm.ADXBeforeFormShow events in order to prevent your form from being shown (see Showing/Hiding Form Instances Programmatically). In addition, you can use events provided by ADXOlForm in order to check the current context. Say, you can use the ADXBeforeFolderSwitch or ADXSelectionChange events of ADXOlForm.

Caching Forms

By default, whenever Add-in Express needs to show a form, it creates a new instance of that form. You can change this behavior by choosing an appropriate value of the ADXOlFormsCollectionItem.Cached property. The values of this property are:

- NewInstanceForEachFolder – it shows the same form instance whenever the user navigates to the same Outlook folder.
- OneInstanceForAllFolders – it shows the same form instance for all Outlook folders.
- None – no form caching is used.

Caching works within the same Explorer window; when the user opens another Explorer window, Add-in Express creates another set of cached forms. Forms shown in Inspector windows cannot be cached.

Is It Inspector or Explorer?

Check the InspectorObj and ExplorerObj properties of ADXOlForm. These properties return COM objects that will be released when your form is removed from its region. And this may occur several times during the life-time of a given form instance because Add-in Express may remove your form from a given region and then embed the form to the same region in order to comply with Outlook windowing.

WebViewPane

When this value (see Outlook Regions) is chosen in the ExplorerLayout property of ADXOlFormsCollectionItem, Add-in Express uses the WebViewUrl and WebViewOn properties of Outlook.MAPIFolder (also Outlook.Folder in Outlook 2007) in order to show your form as a home page for a given folder(s).

Unfortunately, due to a bug in Outlook 2002, Add-in Express has to scan all folders in Outlook in order to set and restore the WebViewUrl and WebViewOn properties. The first consequence is a delay at startup if the current profile contains thousands of folders. A simple way to prevent the delay is to disable the corresponding item(s) of the Items collection of the Forms Manager at design-time and enable it in the AddinStartupComplete

event of the add-in module. Because PublicFolders usually contains many folders, Add-in Express doesn't allow using WebViewPane for PublicFolders and all folders below it.

Also, Outbox and Sync Issues and all folders below them aren't supported when using WebViewPane.

Because of the need to scan Outlook folders, WebViewPane produces another delay when the user works in the Cached Exchange Mode (see the properties of the Exchange account in Outlook) and the Internet connection is slow or broken. To bypass this problem Add-in Express allows reading EntryIDs of those folders from the registry. Naturally, you are supposed to write appropriate values to the registry at add-in start-up. Here is the code to be used in the add-in module:

```
internal void SaveDefaultFoldersEntryIDToRegistry(string PublicFoldersEntryID,
    string PublicFoldersAllPublicFoldersEntryID,
    string FolderSyncIssuesEntryID)
{
    RegistryKey ModuleKey = null;
    RegistryKey ADXXOLKey = null;
    RegistryKey WebViewPaneSpecialFoldersKey = null;
    try
    {
        ModuleKey = Registry.CurrentUser.OpenSubKey(this.RegistryKey, true);
        if (ModuleKey != null)
        {
            ADXXOLKey = ModuleKey.CreateSubKey("ADXXOL");
            if (ADXXOLKey != null)
            {
                WebViewPaneSpecialFoldersKey =
                    ADXXOLKey.CreateSubKey
                    ("FoldersForExcludingFromUseWebViewPaneLayout");
                if (WebViewPaneSpecialFoldersKey != null)
                {
                    if (PublicFoldersEntryID.Length >= 0)
                    {
                        WebViewPaneSpecialFoldersKey.
                            SetValue("PublicFolders",
                            PublicFoldersEntryID);
                    }
                    if (PublicFoldersAllPublicFoldersEntryID.Length >= 0)
                    {
                        WebViewPaneSpecialFoldersKey.
                            SetValue("PublicFoldersAllPublicFolders",
                            PublicFoldersAllPublicFoldersEntryID);
                    }
                    if (FolderSyncIssuesEntryID.Length >= 0)
                    {
                        WebViewPaneSpecialFoldersKey.
```

```
                                SetValue("FolderSyncIssues",
                                FolderSyncIssuesEntryID);
                        }
                    }
                }
            }
        }
        finally
        {
            if (ModuleKey != null)
            {
                ModuleKey.Close();
            }
            if (WebViewPaneSpecialFoldersKey != null)
            {
                WebViewPaneSpecialFoldersKey.Close();
            }
            if (ADXXOLKey != null)
            {
                ADXXOLKey.Close();
            }
        }
}
```

Toolbar Controls for Microsoft Office

The Add-in Express 2009 Extensions for Microsoft Office Toolbars (or the Toolbar Controls) is a plug-in for Add-in Express designed to overstep the limits of existing CommandBar controls. With the Toolbar Controls you can use any .NET controls, not only Office controls, on your command bars. Now you can add tree-views, grids, diagrams, edit boxes, reports, etc. to your command bars.

To make the text below easy to read, let's define three terms, namely:

- Command bar controls are controls such as command bar buttons and command bar combo boxes provided by the Office object model. These controls are Office controls and they are supported by Add-in Express.
- Non-Office controls are any controls, both .NET built-in and third party controls, such as tree-views, grids, user controls, etc. Usually, you use these controls on your Windows application forms.
- Advanced command bar control is an instance of ADXCommandBarAdvancedControl or the ADXCommandBarAdvancedControl class itself (depending on the context).

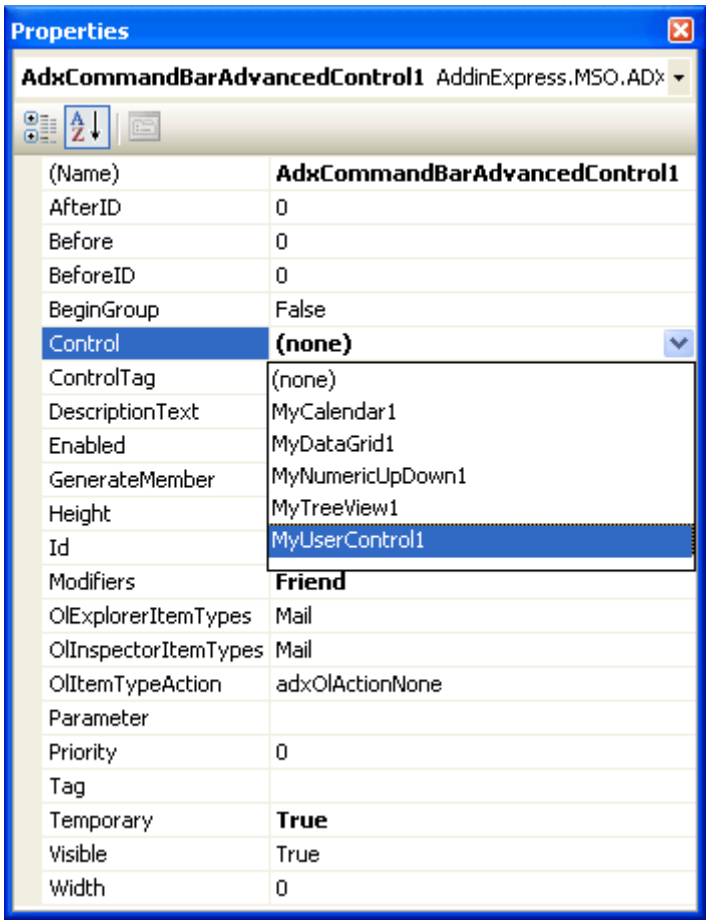
What is ADXCommandBarAdvancedControl?

If you have developed at least one add-in based on Add-in Express, you probably ran into ADXCommandBarAdvancedControl when adding command bar controls to your command bars. Yes, it is that strange item of the Add button on the ADXCommandBarControl collection editor.

This plug-in gives you a chance to use any non-Office controls such as tree-views, grinds, labels, edit and combo boxes, diagrams on any Office command bars. Now you can add ADXCommandBarAdvancedControl, an advanced command bar control, to your command bar and bind it to any non-Office control you placed on the add-in module. As a result, you will have your grid, tree-view or image placed on your command bar.

Hosting any .NET Controls

In addition to properties common for Office command bar controls, ADXCommandBarAdvancedControl has one more property. It is the Control property, the most important one. With this property, you can select a non-Office control to place it on your command bar. Have a look at the picture below. The add-in module contains five controls – MyCalendar, MyDataGrid, MyNumericUpDown, MyTreeView and MyUserControl. The Control property asks you to select one of these controls. If you select MyUserControl, your add-in adds MyUserControl to your command bar. With the Control property, ADXCommandBarAdvancedControl becomes a host for your non-Office controls.

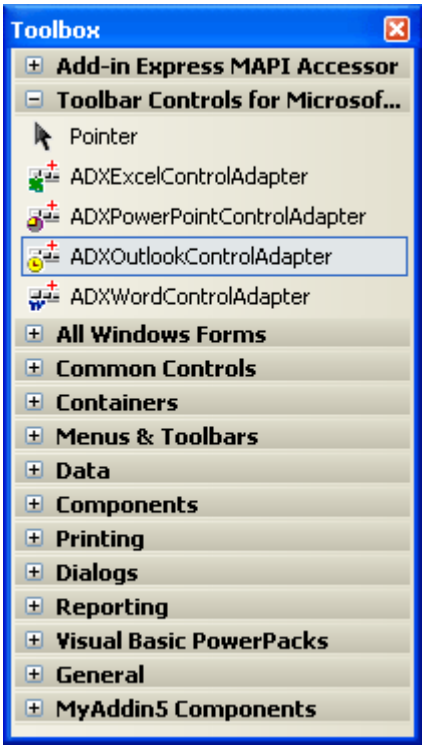


On .NET, ADXCommandBarAdvancedControl supports all controls based on System.Windows.Forms.Control. So, on your command bars, you can use both built-in controls and third-party controls based on System.Windows.Forms.Control. Just add them to the add-in module, add an advanced command bar control to your command bar, and select your non-Office control in the Control property of ADXCommandBarAdvancedControl.

Control Adapters

You may ask us what the Toolbar Controls described above does and what it is for, if ADXCommandBarAdvancedControl is already included in Add-in Express. In general, ADXCommandBarAdvancedControl is still abstract in Add-in Express but it is implemented by the Toolbar Controls if it is plugged in Add-in Express. So, our answer is: the Toolbar Controls for Microsoft Office implements ADXCommandBarAdvancedControl for each Office application.

The Toolbar Controls adds a new tab, "Toolbar Controls for Microsoft Office", to the Toolbox and places several components on the tab (see the screenshot below). The Toolbar Controls supports each Office application by special components called control adapters. Only control adapters know how to add your controls to applications specific command bars. So, the control adapters are the Toolbar Controls itself.



In Express editions of Visual Studio, you need to add the control adapters manually.

The add-in module can contain control adapters only. For example, you should add an ADXExcelControlAdapter to the add-in module if you want to use non-Office controls in your Excel add-in. To use non-Office controls on several Office applications you should add several control adapters. For example, if you need to use your controls in your add-in that supports Outlook, Excel, and Word, you should add three control adapters: ADXExcelControlAdapter, ADXWordControlAdapter, and ADXOutlookControlAdapter to the add-in module.

ADXCommandBarAdvancedControl

As described above, the Toolbar Controls implements the ADXCommandBarAdvancedControl class that is still abstract in Add-in Express without the Toolbar Controls installed. In addition to properties common for all command bar controls, ADXCommandBarAdvancedControl provides two special properties related to the Toolbar Controls.

The Control Property

The Control property binds its ADXCommandBarAdvancedControl to a non-Office control and can be used at design-time as well as at run-time. To place your non-Office control on your command bar you just select your control in the Control property at design-time, or set the Control property to an instance of your control at run-time:

```
Private Sub AddinModule_AddinInitialize( _
   ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.AddinInitialize
   BossCheckbox = New System.Windows.Forms.CheckBox
   Me.AdxCommandBarAdvancedControl1.Control = BossCheckbox
End Sub
```

The ActiveInstance Property

The ActiveInstance property is read-only; it returns the instance of the control that was created for the current context. For example, you can initialize your control for the active Inspector window by handling the InspectorActivate event:

```
Private Sub adxOutlookEvents_InspectorActivate( _
   ByVal sender As System.Object, ByVal inspector As System.Object,
   ByVal folderName As System.String) _
      Handles adxOutlookEvents.InspectorActivate

   Dim ChkBox As System.Windows.Forms.CheckBox = _
      Me.AdxCommandBarAdvancedControl1.ActiveInstance
   If ChkBox IsNot Nothing Then ChkBox.Enabled = False
End Sub
```

Please note that the ActiveInstance property is not actual in most cases when you would like to use it. However, you can always use any window activate events such as the InspectorActivate event of Outlook and WindowActivate event of Word. The table bellow shows you the order of event processing by the example of the Outlook Inspector window opened by the user.

1. Outlook fires the built-in NewInspector event. Add-in Express traps it and fires the NewInspector event of ADXOutlookEvents.		ActiveInstance returns NULL.
	2. ADXOutlookEvents runs your NewInspector event handlers.	ActiveInstance returns NULL.
	3. The Toolbar Controls creates an instance of your control.	ActiveInstance returns NULL.
	4. Outlook fires the built-in InspectorActivate event. Add-in Express handles it and fires the InspectorActivate event of ADXOutlookEvents.	ActiveInstance returns NULL.
	5. The Toolbar Controls creates an instance of your control for the opened Inspector. ADXOutlookEvents runs your InspectorActivate event handlers.	ActiveInstance returns the instance of your control that was cloned from your original control.

Application-specific Control Adapters

All Office applications have different window architectures. All Office windows themselves are different. All our control adapters have a unified programming interface but different internal architectures that take into account the windows architecture of the corresponding applications. All features of all control adapters are described below.

Outlook

Outlook has two main windows – Explorer and Inspector windows. The user can open several Explorer and Inspector windows. Our Outlook control adapter supports non-Office controls on both Explorer and Inspector windows, and creates an instance of your control whenever the user opens a new window.

Please note, if Word is used as an e-mail editor, Outlook uses MS Word as an Inspector window. In this case, Word is running in a separate process. In this scenario, because of obvious and unsolvable problems the Outlook control adapter hides all instances of your control on all inactive Word Inspector windows, but shows them once the Inspector is activated.

Excel

In spite of the fact that Excel allows placing its windows on the Task Bar, all its command bars work like in MDI applications. So, your controls are created only once, at Excel start-up. However, you can still use the WorkbookActivate, WindowActivate, and SheetActivate events to initialize your non-Office controls according to the context.

Word

Word creates its command bars for all document windows, so your non-Office controls are instanced whenever the user opens a new window or a document. We recommend using the WindowActivate event to initialize your control for the current window.

PowerPoint

Notwithstanding the fact that PowerPoint makes possible placing its windows on the Task Bar, PowerPoint is an MDI application. So, your controls are created only once, at PowerPoint startup. However, you can still use the WindowActivate event to initialize your non-Office controls according to the context.

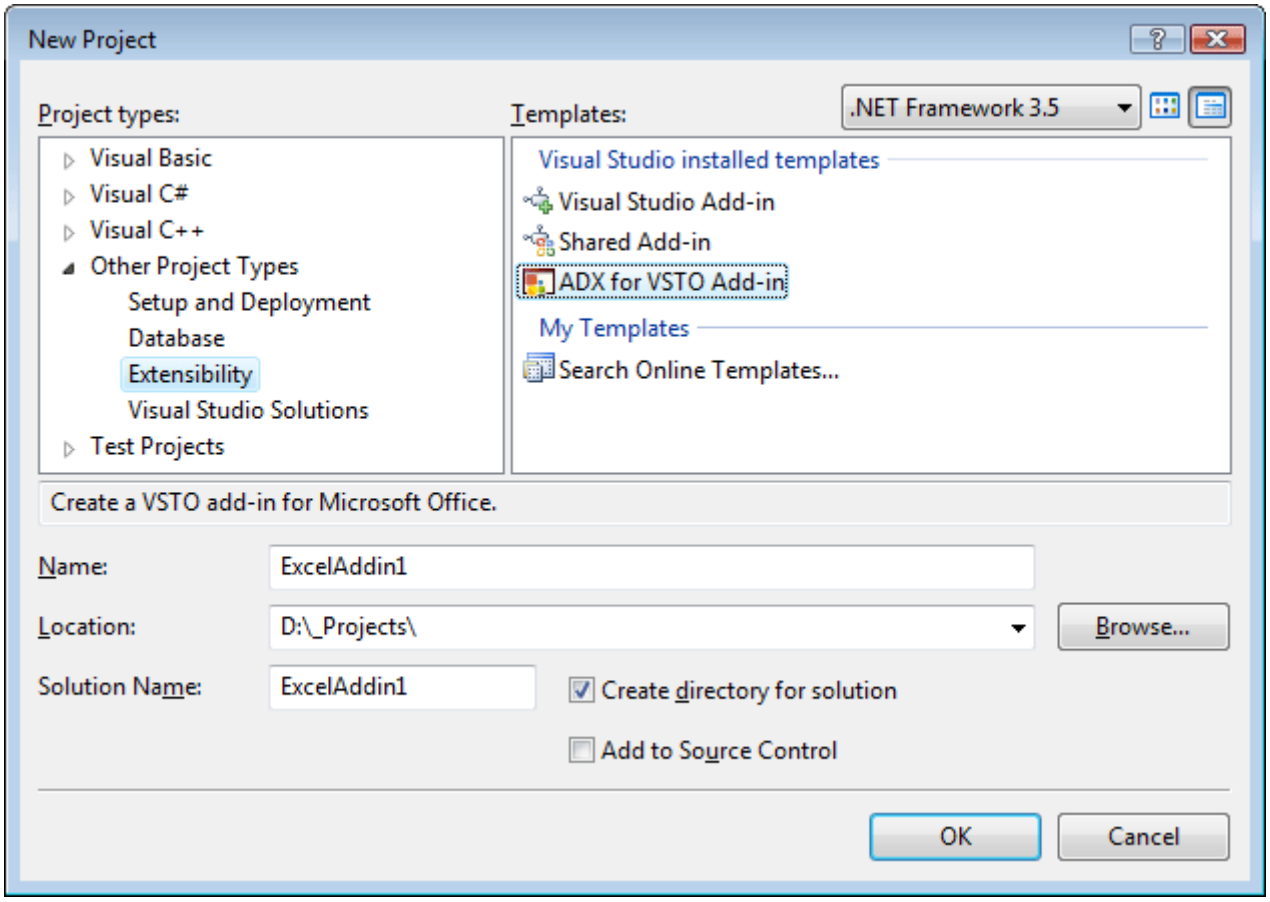
Sample Projects

The projects below are developed in VS 2008 for Office 2003. They demonstrate how you can develop a single codebase for an add-in that shows command bars in Office 2003, Ribbon controls in Office 2007-2010, and task panes in all these Office versions. See also Downloading Sample Projects

Your First Microsoft Office Add-in

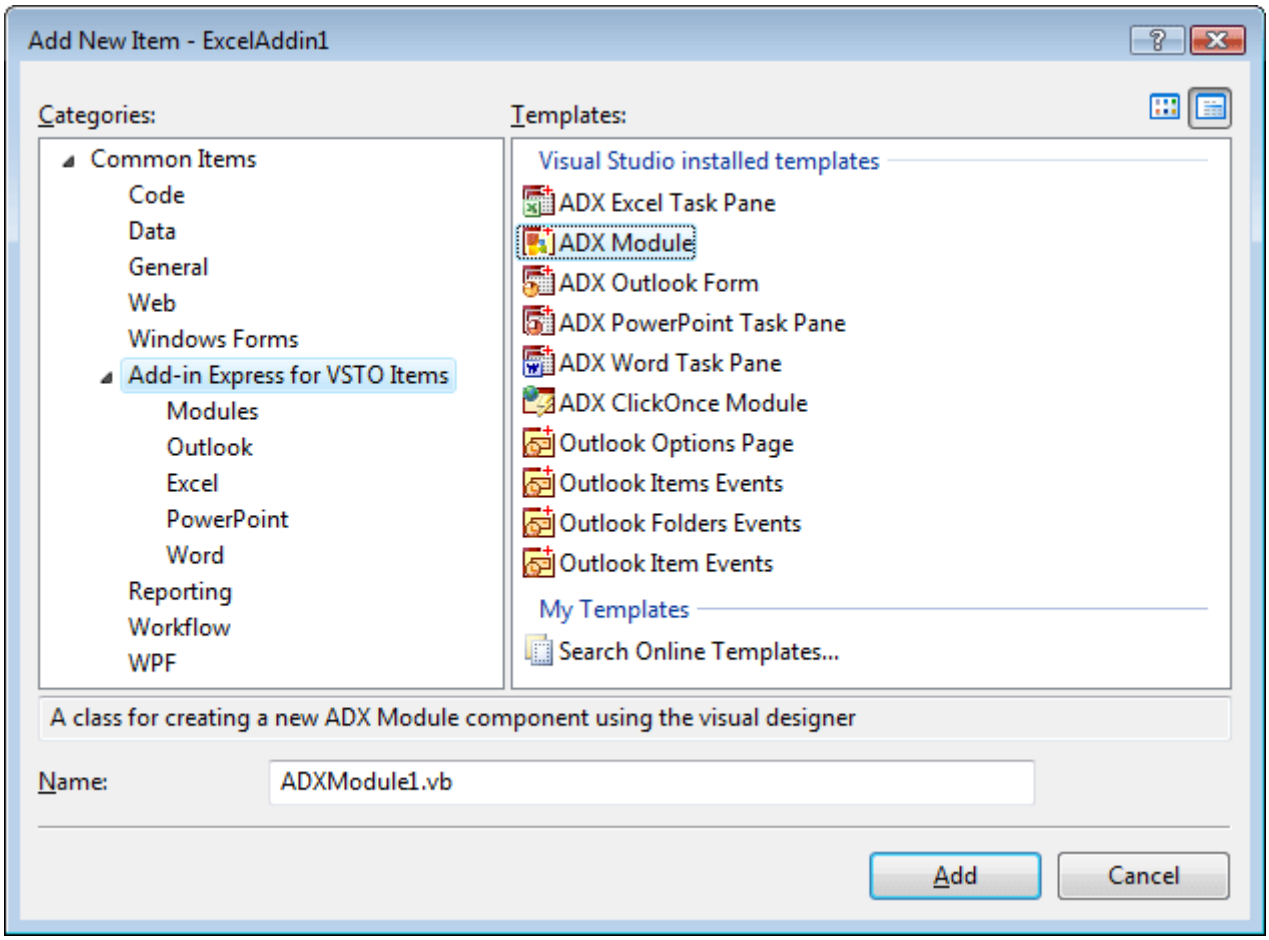
Step #1 – Creating an Excel Add-in Project

If you use VS 2008-2010, then choose File | New | Project… in the menu and find the Add-in Express for VSTO Add-in item in the Extensibility node of the New Project dialog:



Clicking on OK starts the Add-in Express project wizard. In the project wizard window choose the programming language, the host application of your add-in and its version, make sure that the Generate the Setup Project option is on and click Finish.

If you use VS 2005 with VSTO 2005 SE installed, then create a new Excel 2003 add-in solution and, in the Add New Item Dialog, choose the Add-in Express Module item as shown on the screenshot below:



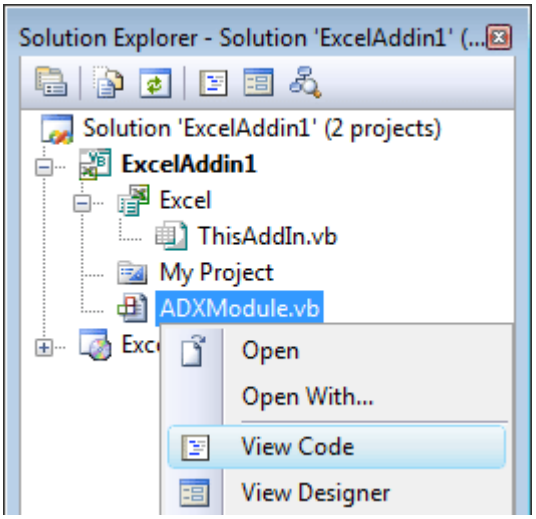
This produces the following code in ThisAddin.vb:

```
Public Class ThisAddIn
    Private Sub ThisAddIn_Startup(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Startup

        'Add-in Express for VSTO generated code
        ADXModule.Initialize(Me, System.Type.GetType("ExcelAddin1.ADXModule"))

        Me.Application = _
            CType( _
                Microsoft.Office.Tools.Excel.ExcelLocale1033Proxy.Wrap _
                    (GetType(Excel.Application), Me.Application), _
                Excel.Application)
    End Sub

    Private Sub ThisAddIn_Shutdown(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Shutdown
        'Add-in Express for VSTO generated code
        ADXModule.Finalize(Me)
```

```
    End Sub
End Class
```

Also, this adds the ADXModule.vb (or ADXModule.cs) file to your add-in project. We are going to look at this file in the next step.

Step #2 - Add-in Module



ADXModule.vb (or ADXModule.cs) is an add-in module that is the core part of the add-in project (see Add-in Express Module), a container for Add-in Express components. You specify the add-in properties in the module's properties, add Add-in Express components to the module's designer, and write the functional code of your add-in in this module. The code for ADXModule.vb is as follows:

```
Imports System.Runtime.InteropServices
Imports System.ComponentModel
Imports System.Windows.Forms
Imports Excel = Microsoft.Office.Interop.Excel
Imports Office = Microsoft.Office.Core

'Add-in Express for VSTO Module
Public Class ADXModule
    Inherits AddinExpress.VSTO.ADXExcelAddin

#Region " Component Designer generated code. "
    'Required by designer
    Private components As System.ComponentModel.IContainer

    'Required by designer - do not modify
    'the following method
    Private Sub InitializeComponent()

    End Sub

#End Region

#Region " Add-in Express automatic code "

    'Required by Add-in Express - do not modify
    'the methods within this region

    Public Overrides Function GetContainer() As System.ComponentModel.IContainer
        If components Is Nothing Then
```

```
            components = New System.ComponentModel.Container
        End If
        GetContainer = components
    End Function

#End Region

    Public Sub New(ByVal Application As Object)
        MyBase.New(Application)

        'This call is required by the Component Designer
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    Public Sub New()
        MyBase.New()

        'This call is required by the Component Designer
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    Public ReadOnly Property ExcelApp() _
        As Microsoft.Office.Interop.Excel.Application
        Get
            Return HostApplication
        End Get
    End Property

End Class

Partial Public Class ThisAddIn

    Protected Overrides Function RequestService(ByVal serviceGuid As Guid) As
Object
        If serviceGuid = _
            GetType(AddinExpress.VSTO.IRibbonExtensibility).GUID Then
            Dim useMSORibbon As Boolean = False
            Dim types As Type() = Me.GetType().Assembly.GetTypes()

            For i As Integer = 0 To types.Length - 1
```

```
                If (types(i).IsClass And _
                    types(i).BaseType.FullName.Equals _
                        ("Microsoft.Office.Tools.Ribbon.OfficeRibbon")) _
                Then
                    useMSORibbon = True
                    Exit For
                End If
            Next

            If (Not useMSORibbon) Then
                ADXModule.Initialize(Me, _
                    System.Type.GetType("ExcelAddin1.ADXModule"))
                Return ADXModule.CurrentInstance
            End If
        End If

        Return MyBase.RequestService(serviceGuid)
    End Function

End Class
```

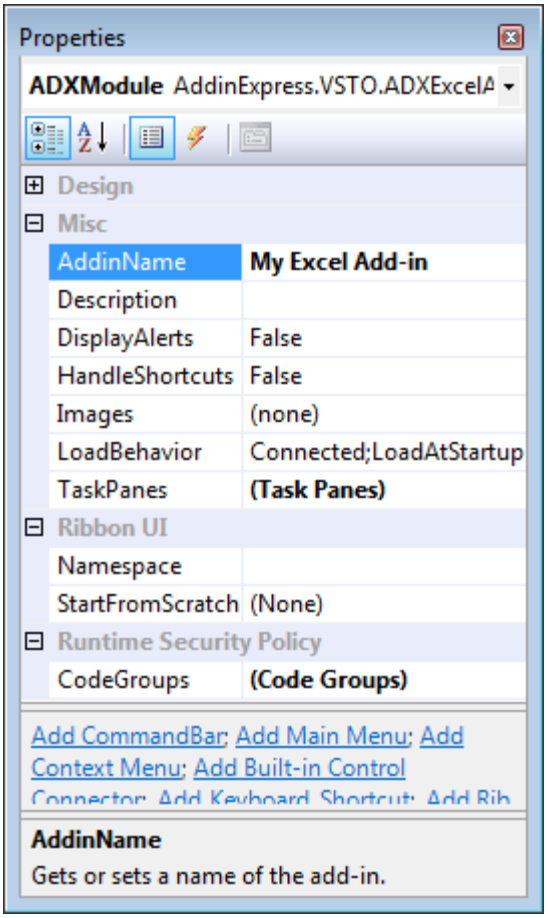
Pay attention to the ExcelApp property of the ADXModule class. You can use it in your code to access Excel objects.

Step #3 - Add-in Module Designer

The designer of the add-in module allows setting add-in properties and adding components to the module. To open the designer, right-click the add-in module in Solution Explorer and choose the View Designer popup menu item (see Adding Components to the Add-in Module).

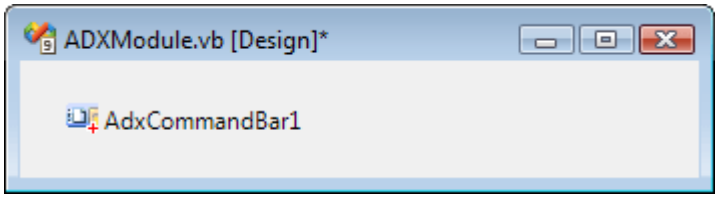
The designer provides add-in properties and a number of events including add-in events, Excel events, Ribbon control events, and custom task pane related events. It is also a container for Add-in Express components that can be added to the module via the context menu of the designer (see also Add-in Module Commands).

In the Properties window, you set the name and description of your add-in (see also Add-in Express Module).

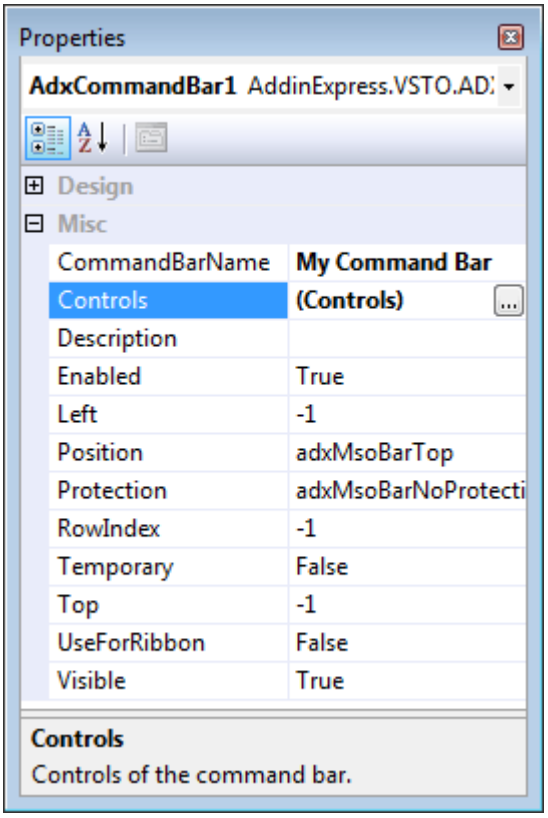


Step #4 - Adding a New Toolbar

To add a command bar to your add-in, use the Add CommandBar command that adds an ADXCommandBar component to the Add-in Module (see Adding Components to the Add-in Module).

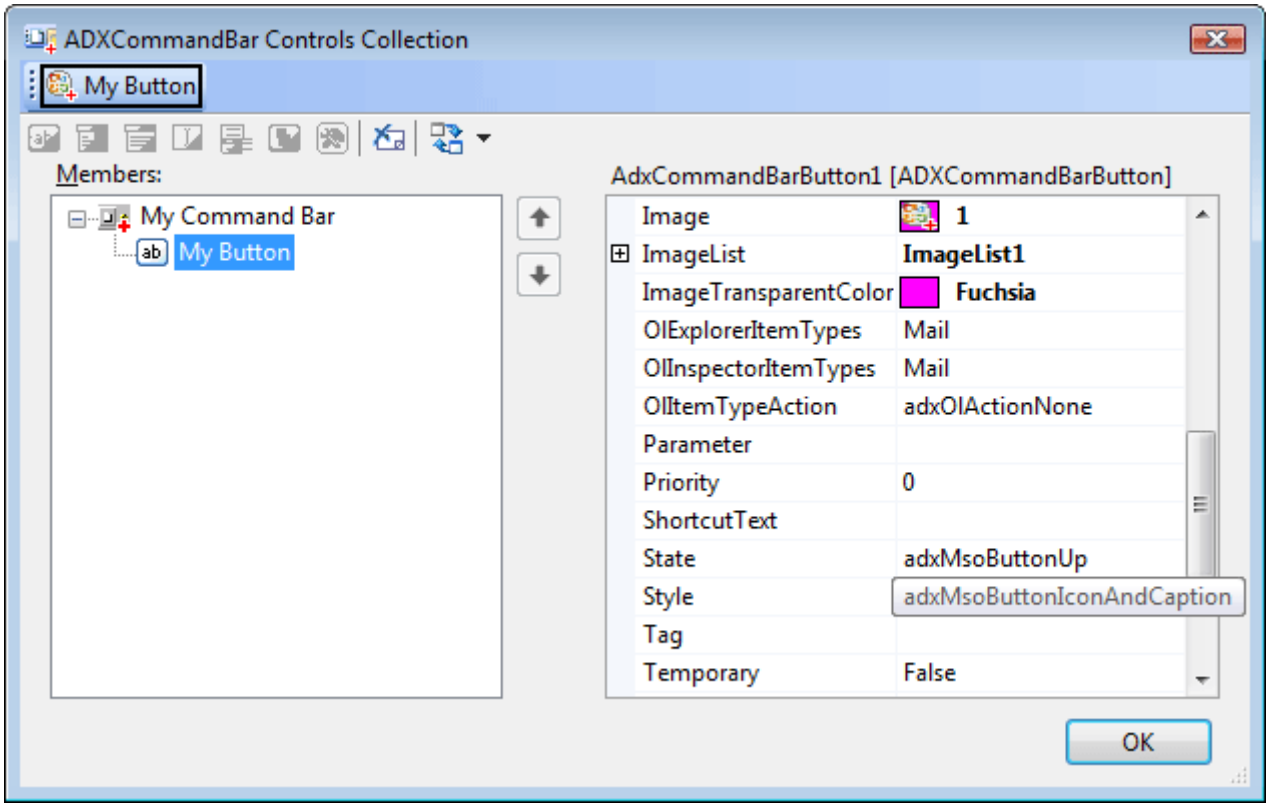


Select the command bar component and, in the Properties window, specify the command bar name using the CommandBarName property. In addition, you choose its position in the Position property.



Step #5 - Adding a New Toolbar Button

To add a new button to the command bar, choose the Controls property, and click the property editor button.



Specify the button's Caption, set its Style (the default value doesn't show a button image), and close the collection editor. Select the newly added button in the topmost combo of the Properties window and add the Click event handler.

Step #6 - Accessing Host Application Objects

The Add-in Module provides the HostApplication property that returns the Application object (of the Object type) of the host application the add-in is currently running in. For your convenience, the Add-in Express Project Wizard adds host-related properties to the Add-in module. You use these properties to access host application objects. For instance, this sample add-in has the ExcelApp property in the Add-in Module:

```
Public ReadOnly Property ExcelApp() As Excel._Application
    Get
        Return HostApplication
    End Get
End Property
```

This allows us to write the following code to the Click event of the button just added.

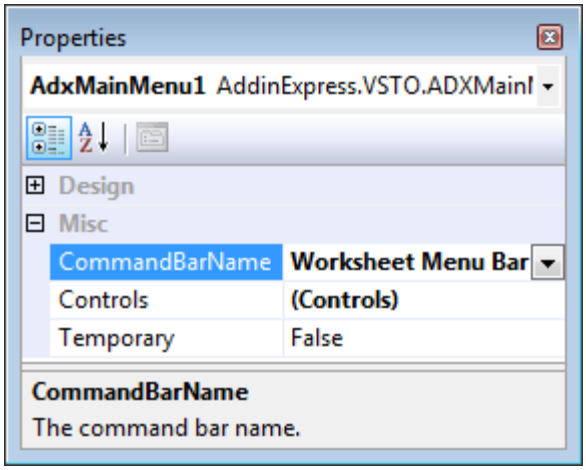
```
Private Sub DefaultAction(ByVal sender As System.Object) _
    Handles AdxCommandBarButton1.Click
    MsgBox("The current cell is " + GetAddress())
End Sub

Friend Function GetAddress() As String
    Dim Address As String = "Unknown"
    Dim ActiveWindow As Excel.Window = Me.ExcelApp.ActiveWindow
    If Not ActiveWindow Is Nothing Then
        Dim ActiveCell As Excel.Range = ActiveWindow.ActiveCell
        'relative address
        Address = ActiveCell.AddressLocal(False, False)
        Marshal.ReleaseComObject(ActiveCell)
        Marshal.ReleaseComObject(ActiveWindow)
    End If
    Return Address
End Function
```

The use of Marshal.ReleasComObject is described in Releasing COM objects.

Step #7 - Customizing the Main Menu

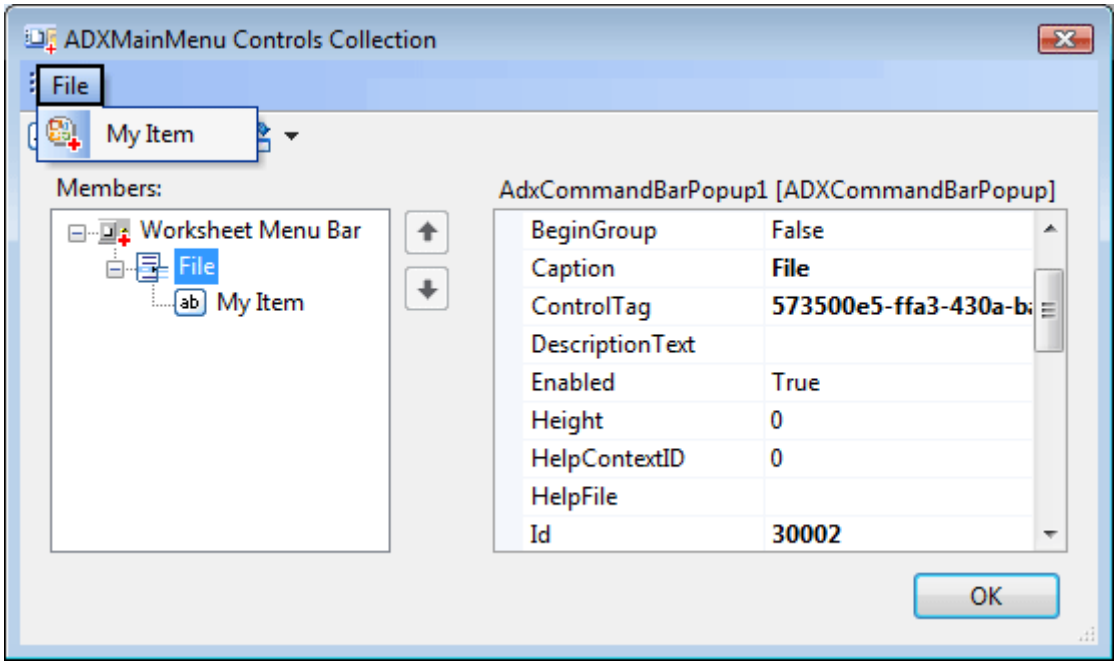
Add-in Express provides a component to customize the main menu of any Office application. Some Office applications have several main menus and Excel is a good example: Excel 2003 provides two main menus called Worksheet Menu Bar and Chart Menu Bar. Naturally, Excel 2007 and 2010 don't show these menus;

they are replaced with the Ribbon UI. Nevertheless, these menus still exist and you may want to use this fact in your code.



To customize the File menu in Excel version 2003, add a main menu component and, in the CommandBarName property, specify the main menu. The screenshot shows how you set up the main menu component in order to customize the Worksheet Menu Bar main menu in Excel 2003.

Now you can use the Controls property to add custom and built-in controls to the main menu. Please note, however, that Office imposes restrictions on the control types available to the developer when customizing the main menu; you can use command bar buttons and command bar popups only.



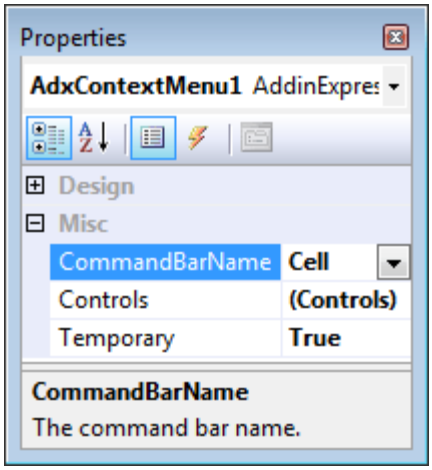
The screenshot above shows how to add a custom button to the File menu of Excel. First off, you add a popup control and specify its Id property: in our example it is 30002, which is the ID of the File menu in Office applications. To find this and similar IDs, use our free Built-in Control Scanner. See also Using Existing Command Bar Controls.

Then you add a button and set its properties in the way described in Step #5 – Adding a New Toolbar Button. Pay attention to the BeforeID property of the button. To show the button before the New button, you set this property to 3, which is the ID of the New button. Also, remember that showing an image for the button as well as for any command bar control requires choosing a correct value for the Style property.

Step #8 – Customizing Context Menus

A context menu is a specific command bar that can be customized too. Add-in Express allows customizing context menus via the Context Menu component. Its use is similar to that of the Main Menu component:

- Add a context menu component to the add-in module
- Specify the host application, the context menu of which you need to customize
- Specify the context menu to customize
- Add custom controls to the Controls collection



See how to set up such a component to add a custom button to the Cell menu of Excel.

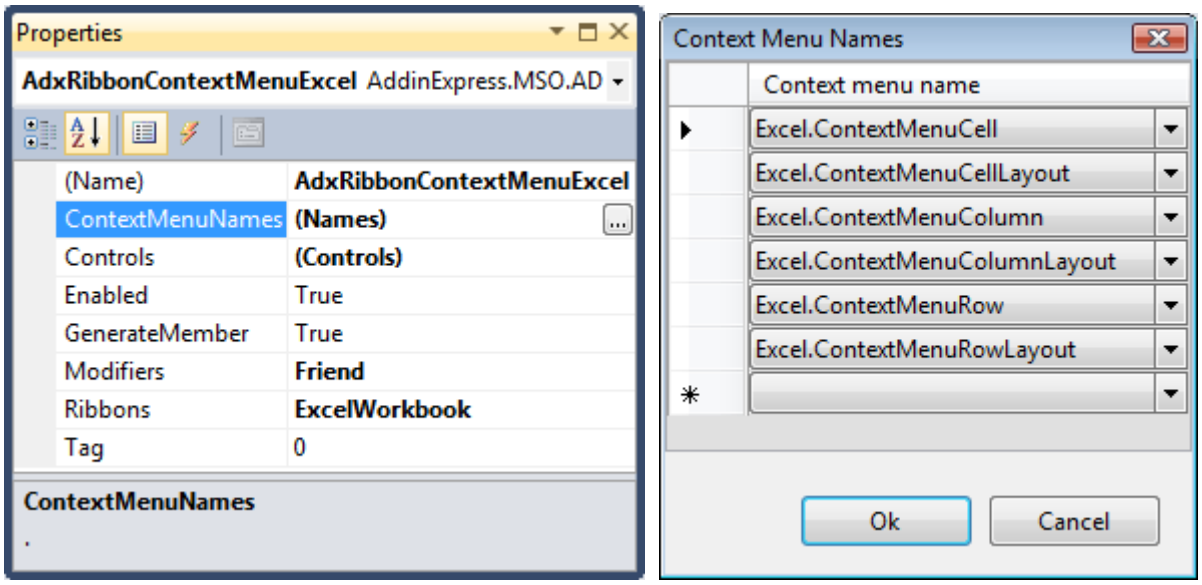
Now, using the visual designer of the Controls collection, you can add buttons and pop-ups to the context menu. Note that, in context menus, a popup is shown as a drop-down item.

You may want to use the BeforeAddControls event provided by the component to modify the context menu depending on the current context. Say the context menu may reflect the Excel cell content, the current chart, etc.

There are several issues related to using command bar based context menus:

- Excel contains two different context menus named Cell. This fact breaks down the command bar development model because the only way to recognize two command bars is to compare their names. This isn't the only exception: see the Built-in Control Scanner to find a number of examples. In this case, the context menu component cannot distinguish context menus. Accordingly, it connects to the first context menu of the name specified by you.
- Command bar based context menu items cannot be positioned in the Ribbon-based context menus: a custom context menu item created with the ADXContextMenu component will always be shown below the built-in and custom context menu items in a Ribbon-based context menu of Office 2010.

To add a custom item to a context menu in Office 2010, you use the ADXRibbonContextMenu component. Unlike its commandbar-based counterpart (ADXContextMenu), this component allows adding custom Ribbon controls to several context menus in the specified Ribbons. The screenshots below demonstrate component settings required for adding a control to the ExcelWorkbook Ribbon. To specify the context menus, to which the control will be added, you use the editor of the ContexMenuNames property of the component.



See also Step #10 – Customizing the Ribbon User Interface.

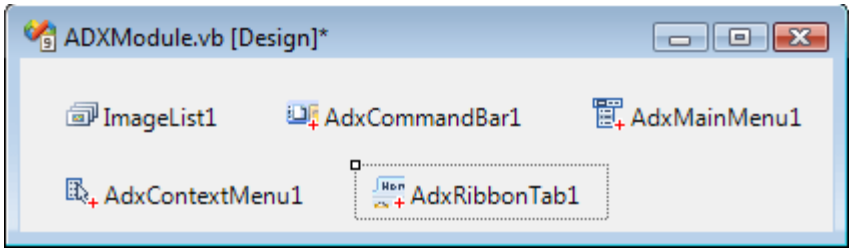
Step #9 - Handling Excel Events

You might see that the Click event handler in the previous step returns not very nice results when no workbook is open. You can disable the button when an Excel window deactivates and enable it when a window activates; so when the last window deactivates, it will not be possible to press the button. The add-in module provides all events of the host application so you can write the following code:

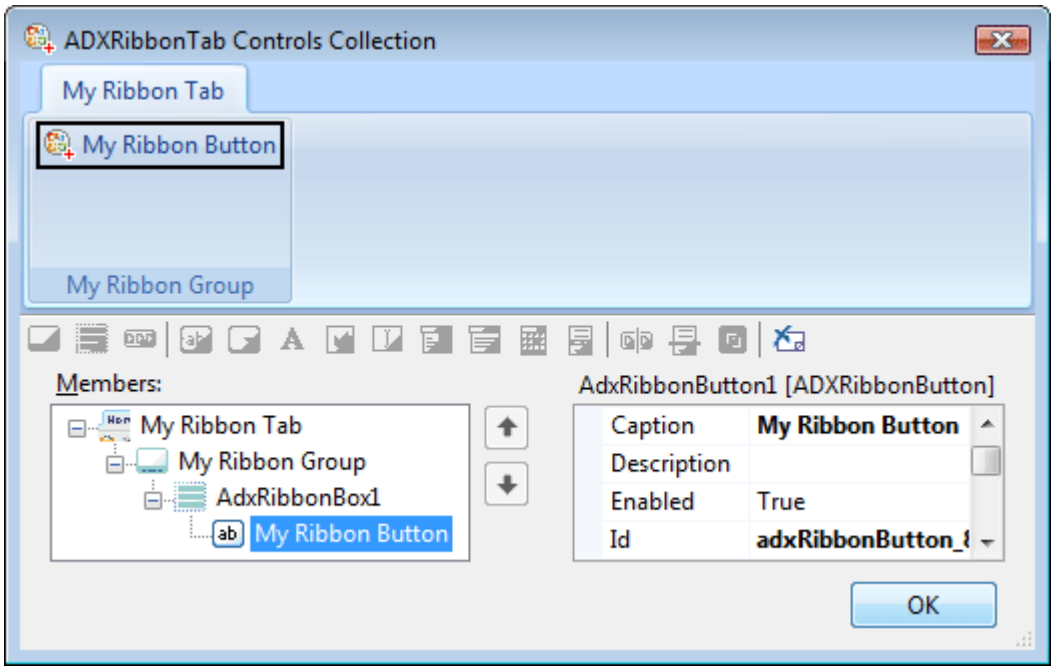
```
Private Sub ADXModule1_WindowActivate(ByVal sender As Object, _
   ByVal hostObj As Object, ByVal window As Object) Handles Me.WindowActivate
   Me.AdxCommandBarButton1.Enabled = True
End Sub

Private Sub ADXModule1_WindowDeactivate(ByVal sender As Object, _
   ByVal hostObj As Object, ByVal window As Object) Handles Me.WindowDeactivate
   Me.AdxCommandBarButton1.Enabled = False
End Sub
```

Step #10 – Customizing the Ribbon User Interface

To add a new tab to the Ribbon UI, you use the Add Ribbon Tab command that adds an ADXRibbonTab component to the module.



In the Properties window, run the visual designer for the Controls collection of the tab. In the designer, use the toolbar buttons or context menu to add or delete components that form the Ribbon interface of your add-in. First, change the caption of your tab to My Ribbon Tab. Then, select the tab, add a Ribbon group, and change its caption to My Ribbon Group. Next, select the group, and add a button group. Finally, select the button group and add a button. Set the button caption to My Ribbon Button. Use the ImageList and Image properties to set the icon for the button.



Click OK, and, in the Properties window, find the newly added Ribbon button. Now add the event handler to the Click event of the button. Write the following code:

```
Private Sub AdxRibbonButton1_OnClick(ByVal sender As System.Object, _
    ByVal control As AddinExpress.VSTO.IRibbonControl, _
    ByVal pressed As System.Boolean) Handles AdxRibbonButton1.OnClick
    DefaultAction(Nothing)
End Sub
```

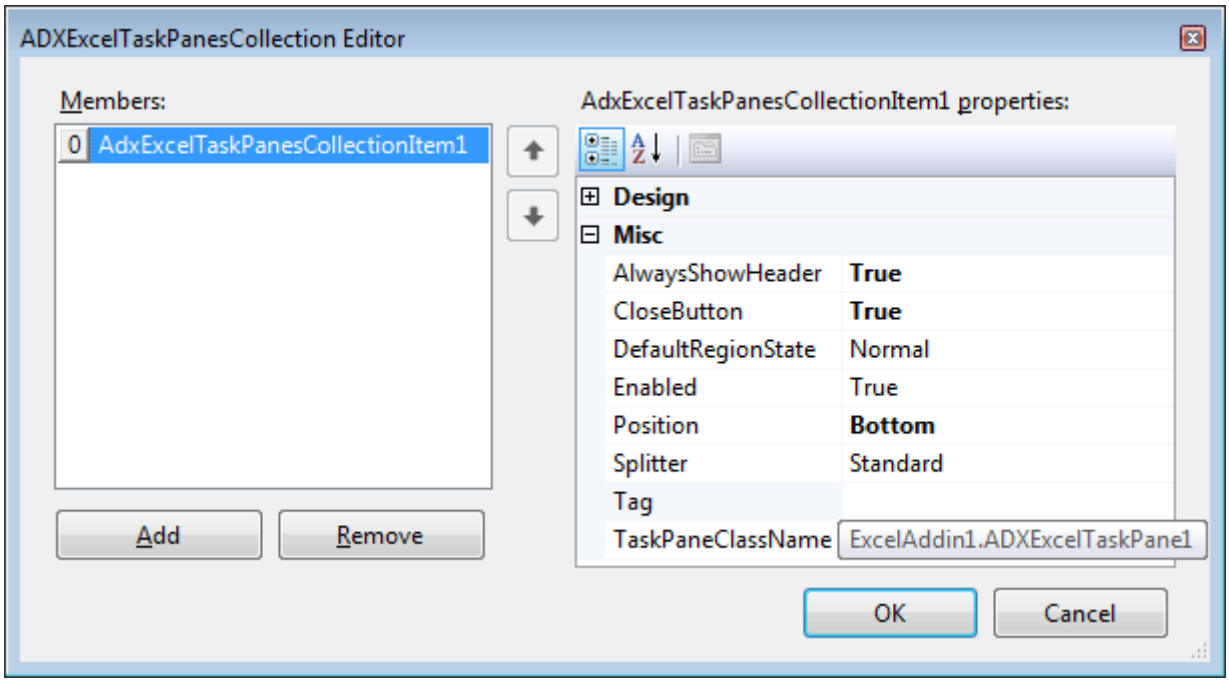
Remember, the designer validates the Ribbon XML markup automatically, so you may run into the situation when you cannot add a control to some Ribbon level. It is a restriction of the Ribbon XML schema.

In the code of this sample add-in, you can find how you can customize the Office Button menu in Office 2007, see the component named AdxRibbonOfficeMenu1. As to the Backstage View, also known as **File Tab** in Office 2010, the sample project provides the AdxBackstageView1 component that implements the customization shown in Figure 3 at Introduction to the Office 2010 Backstage View for Developers. Note, if you customize the Office Button menu only, Add-in Express maps your controls to the Backstage View when the add-in is loaded by Office 2010. If, however, both Office Button menu and File tab are customized at the same time, Add-in Express ignores custom controls you add to the Office Button menu. See also Office Ribbon Components.

Step #11 – Adding Custom Task Panes in Office 2003-2010

Creating a new Excel task pane includes the following steps:

- add an Excel Task Panes Manager (ADXExcelTaskPanesManager) to your add-in module (see Adding Components to the Add-in Module and Add-in Module Commands)
- add an Add-in Express Excel Task Pane (ADXExcelTaskPane) to your project (see Add New Item Dialog)
- add an item to the Items collection of the manager, select the newly added pane in the TaskPaneClassName property of the item and set other properties, such as Position (see the screenshot below).



The properties shown in the screenshot above are:

- AlwaysShowHeader - specifies that the pane header will be shown even if the pane is the only pane in the current region
- CloseButton - specifies if the Close button will be shown in the pane header. Obviously, there's not much sense in setting this property to true when the header isn't shown.
- Position - specifies the region in which an instance of the pane will be shown. Excel panes are allowed in four regions docked to the four edges of the main Excel window: Right, Bottom, Left, and Top.
- TaskPaneClassName - specifies the class name of the Excel task pane.

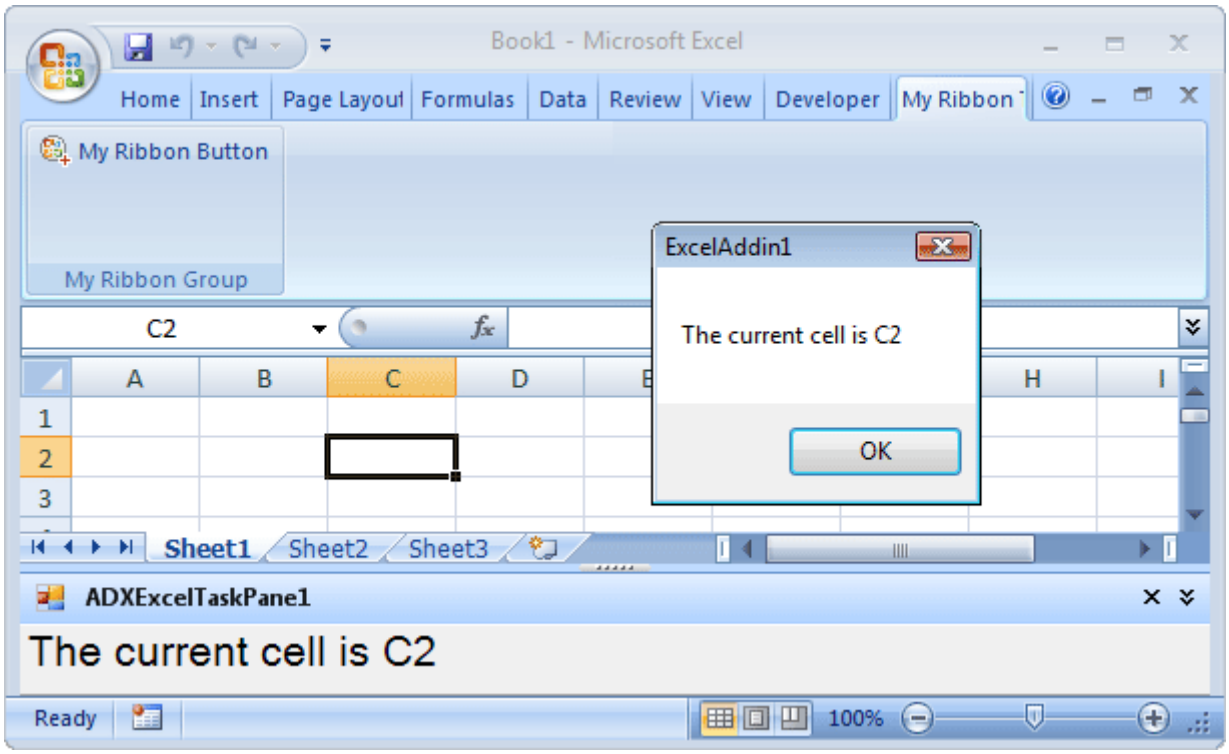
Now you add a label onto the form and set the label up in the following code:

```
Private Sub RefreshTaskPane()
    Dim Pane As ADXExcelTaskPane1 = _
                TryCast(Me.AdxExcelTaskPanesCollectionItem1.TaskPaneInstance,  _
                    ADXExcelTaskPane1)
    If Pane IsNot Nothing Then
        Pane.Label1.Text = Me.GetAddress()
    End If
End Sub
```

See also Advanced Custom Task Panes and Excel Task Panes.

Step #12 - Running the Add-in

Choose the Build <Add-in Project Name> item in the Build menu, then restart Excel, and find your command bars, ribbon tabs, and custom task panes in place. You also find your add-in in the COM Add-ins Dialog.



Step #13 - Debugging the Add-in

Just press F5 or choose Debug | Start Debugging in the menu.

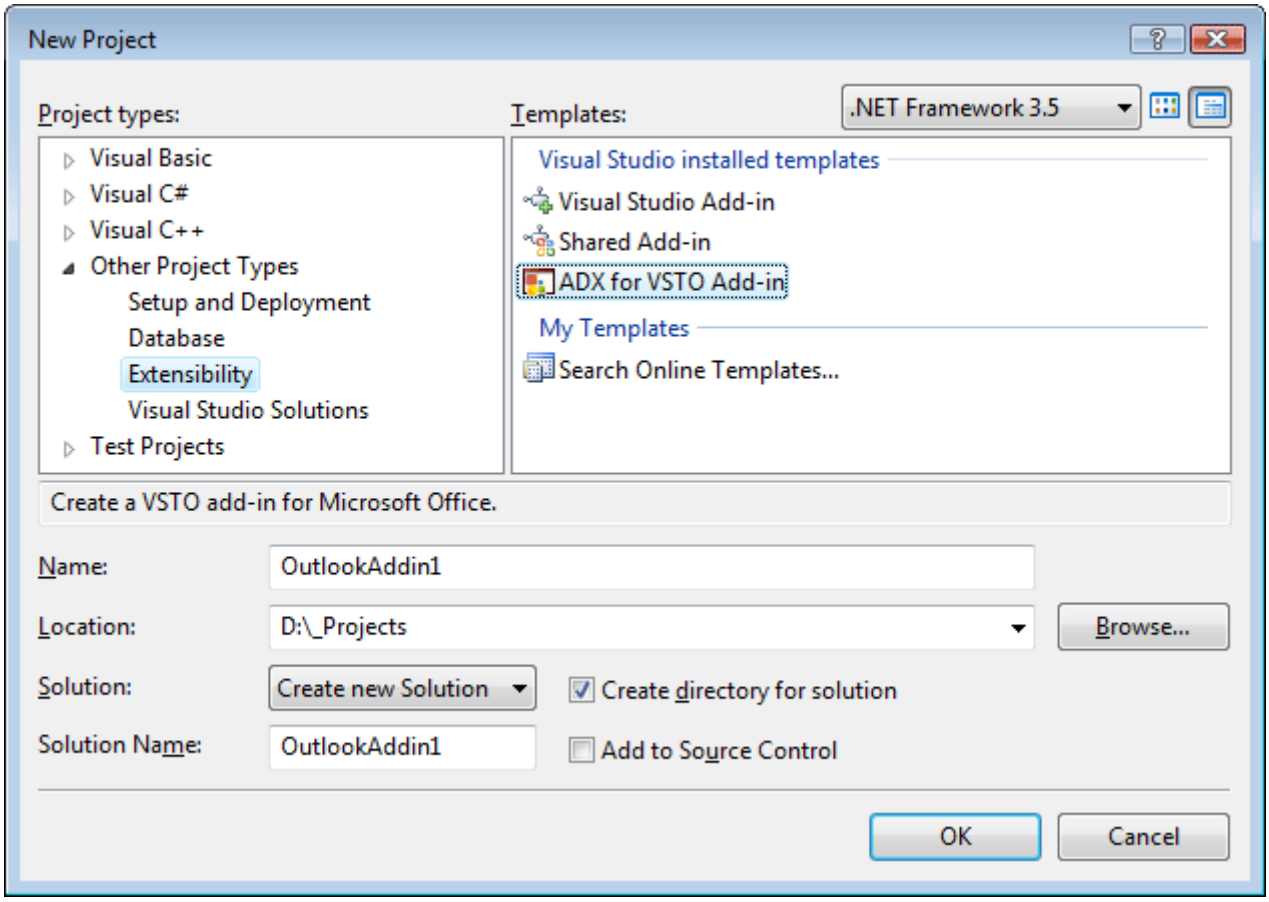
Step #14 - Deploying the Add-in

Build the setup project, transfer the files to the target PC and run the setup.exe. See also VSTO Deployment Support in Add-in Express.

Your First Microsoft Outlook Add-in

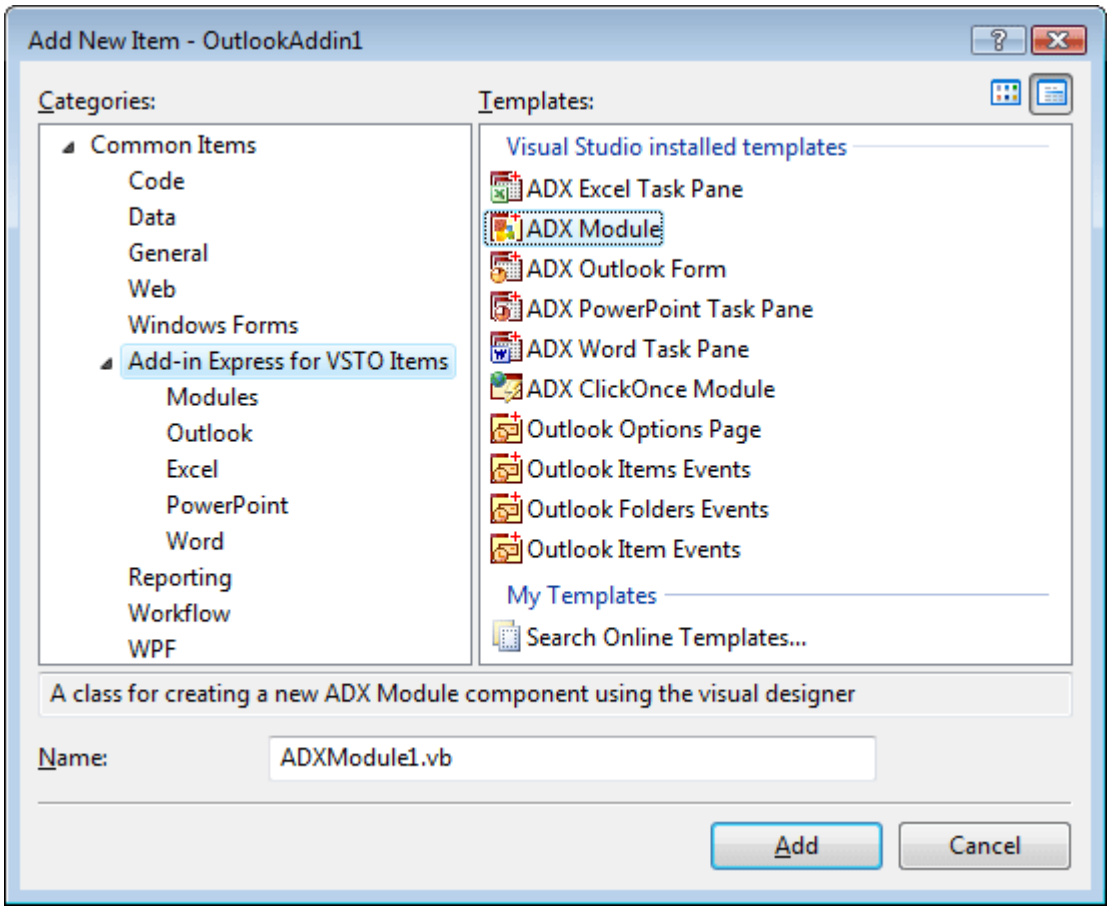
Step #1 – Creating an Outlook Add-in Project

If you use VS 2008, then choose File | New | Project… in the menu and find the Add-in Express for VSTO Add-in item in the Extensibility node of the New Project dialog:



This starts the Add-in Express project wizard. In the project wizard window choose the programming language, the host application of your add-in and its version, make sure that the Generate the Setup Project option is on and click Finish.

If you use VS 2005 with VSTO 2005 SE installed, then create a new Outlook 2003 add-in solution and, in the Add New Item Dialog, choose the Add-in Express Module item as shown on the screenshot below:



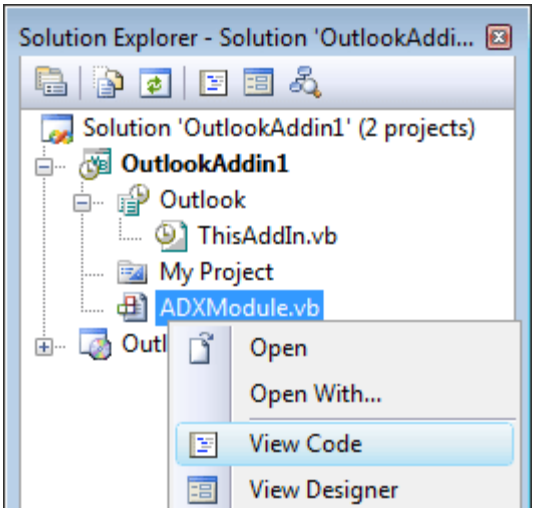
This modifies the code of ThisAddin.vb (or ThisAddin.cs) as follows:

```
Public Class ThisAddIn
    Private Sub ThisAddIn_Startup(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Startup
        'Add-in Express for VSTO generated code
        ADXModule.Initialize(Me, System.Type.GetType("OutlookAddin1.ADXModule"))
    End Sub

    Private Sub ThisAddIn_Shutdown(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Shutdown
        'Add-in Express for VSTO generated code
        ADXModule.Finalize(Me)
    End Sub
End Class
```

Also, this adds the ADXModule.vb (or ADXModule.cs) file to your add-in project. We will look at this file in the next step.

Step #2 - Add-in Module

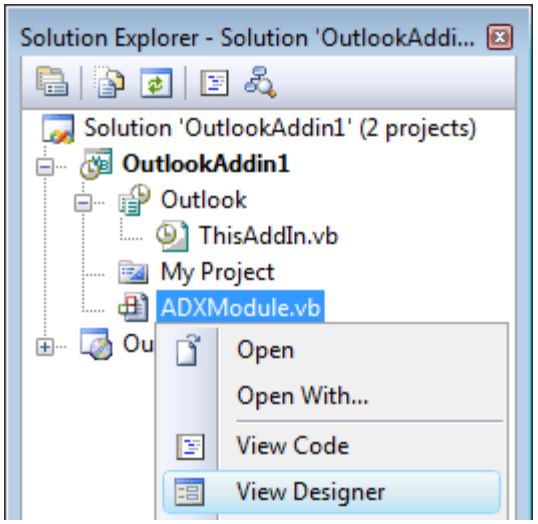
The ADXModule.vb (or ADXModule.cs) is the core part of Add-in Express based add-in projects. It is a container for the Add-in Express components, which allow you to concentrate on the functionality of your add-in. You specify the add-in properties in the module's properties, add Add-in Express components to the module's designer, and write the functional code of your add-in in this module. To review its source code, in Solution Explorer, right-click the file and choose the View Code popup menu item. The code for ADXModule.vb is as follows:




```
Imports System.Runtime.InteropServices
Imports System.ComponentModel
Imports System.Windows.Forms
Imports Outlook =
Microsoft.Office.Interop.Outlook
Imports Office = Microsoft.Office.Core

'Add-in Express for VSTO Module
<ComVisible(True)> _
Public Class ADXModule1
   Inherits AddinExpress.VSTO.ADXOutlookAddin

#Region " Component Designer generated code. "
   'Required by designer
   Private components As System.ComponentModel.IContainer

   'Required by designer - do not modify the following method
   Private Sub InitializeComponent()

   End Sub
#End Region

#Region " ADX automatic code "
   'Required by Add-in Express - do not modify
   'the methods within this region

   Public Overrides Function GetContainer() As _
      System.ComponentModel.IContainer
      If components Is Nothing Then
         components = New System.ComponentModel.Container
      End If
      GetContainer = components
   End Function

#End Region

   Public Sub New(ByVal Application As Object)
```

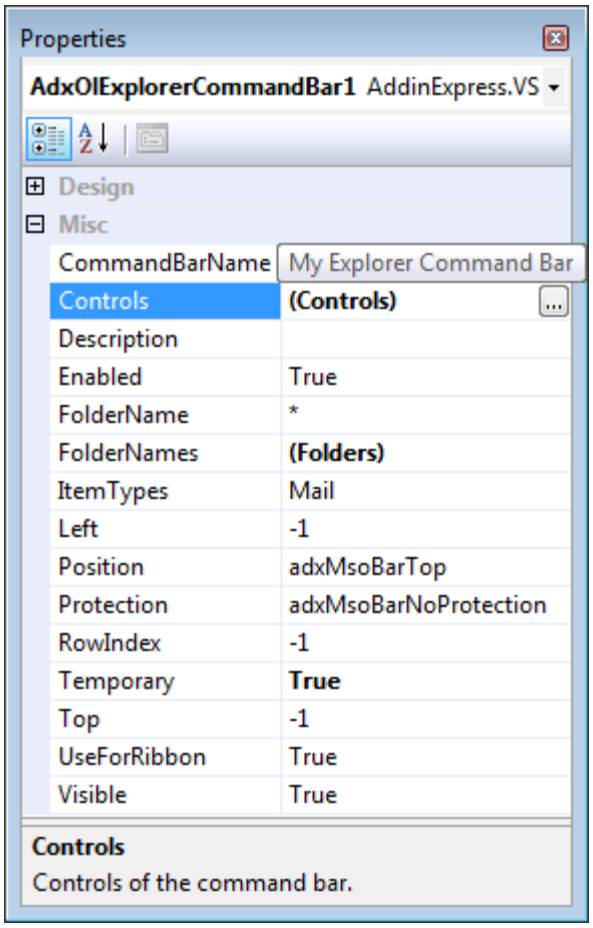
```
      MyBase.New(Application)
      'This call is required by the Component Designer
      InitializeComponent()
      'Add any initialization after the InitializeComponent() call
   End Sub

   Public Sub New()
      MyBase.New()
      'This call is required by the Component Designer
      InitializeComponent()
      'Add any initialization after the InitializeComponent() call
   End Sub

   Public ReadOnly Property OutlookApp() As Outlook._Application
      Get
         Return HostApplication
      End Get
   End Property
End Class

Partial Public Class ThisAddIn
   Protected Overrides Function RequestService(ByVal serviceGuid As Guid) _
      As Object
      If serviceGuid = GetType(Office.IRibbonExtensibility).GUID Then
         ADXModule1.Initialize(Me, _
            System.Type.GetType("OutlookAddIn1.ADXModule1"))
         Return ADXModule1.CurrentInstance
      End If
      Return MyBase.RequestService(serviceGuid)
   End Function
End Class
```

Step #3 - Add-in Module Designer



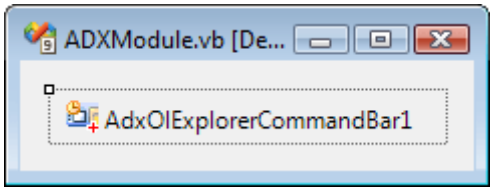
To show the Add-in Express Module Designer, in Solution Explorer, right-click the add-in module file and choose the View Designer popup menu item. The designer provides add-in properties and a number of events including add-in events, application-level Outlook events, Ribbon control events, and Office 2007 task pane related events. It is also a container for Add-in Express components that can be added to the module via the context menu of the designer (see also Add-in Module Commands).

In the Properties window for the add-in module designer, specify the name of your add-in, say **My Outlook add-in**.



Step #4 - Adding a New Explorer Command Bar

To add a command bar to Outlook Explorer windows, use the Add Explorer CommandBar command that adds an ADXOlExplorerCommandBar to the add-in module (see also Adding Components to the Add-in Module).



Select the Explorer command bar component, then, in the Properties window, specify the command bar name in the CommandBarName property and choose its position (see the Position property). The Explorer command bar component provides context-sensitive properties. They are FolderName, FolderNames, and ItemTypes (see Outlook CommandBar Visibility Rules and Outlook Add-ins – Template Characters in FolderName).

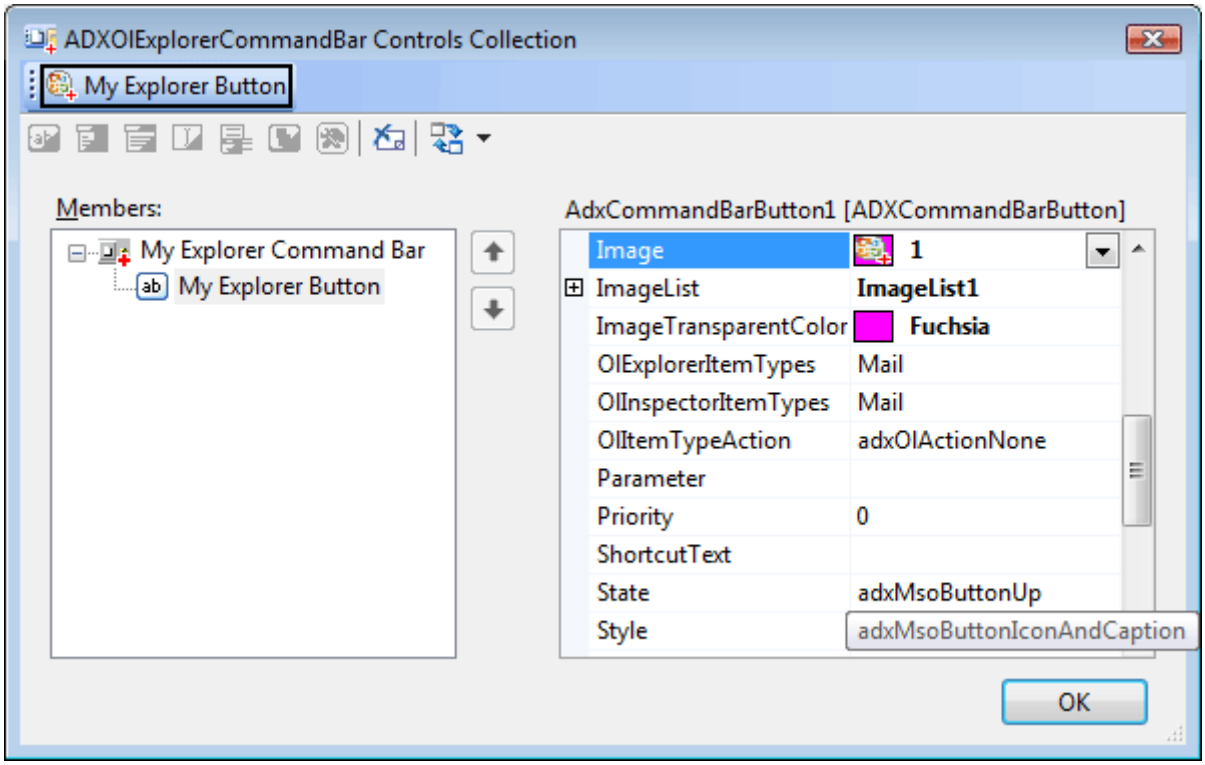
In the screenshot, you see an Outlook Explorer command bar component that creates a custom toolbar to be shown for every Outlook folder (FolderName = "*") the default item type of which is Mail. In Outlook 2003, this command bar will be positioned at the top of the Outlook Explorer window. In Outlook 2007, this command bar will be shown in the Add-ins tab if the Visible property of the command bar is set to True **and** if the controls of the command bar are visible.

See also Command Bars: Toolbars, Menus, and Context Menus.

Step #5 - Adding a New Command Bar Button

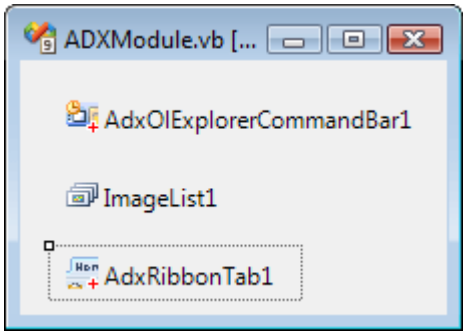
To add a new button to the Explorer command bar, select the Explorer command bar component and, in the Properties window, choose the Controls property, and click the property editor button. This starts the command bar visual designer. In its UI, you can add /move /delete any command bar control (see also Command Bar Controls).

To add an icon to the button, add an ImageList to the add-in module. Then specify the button's Caption property and set the ImageList, Image, and ImageTransparentColor properties of the button. Finally, set the Style property because its default value doesn't show the button image (see the screenshot below).



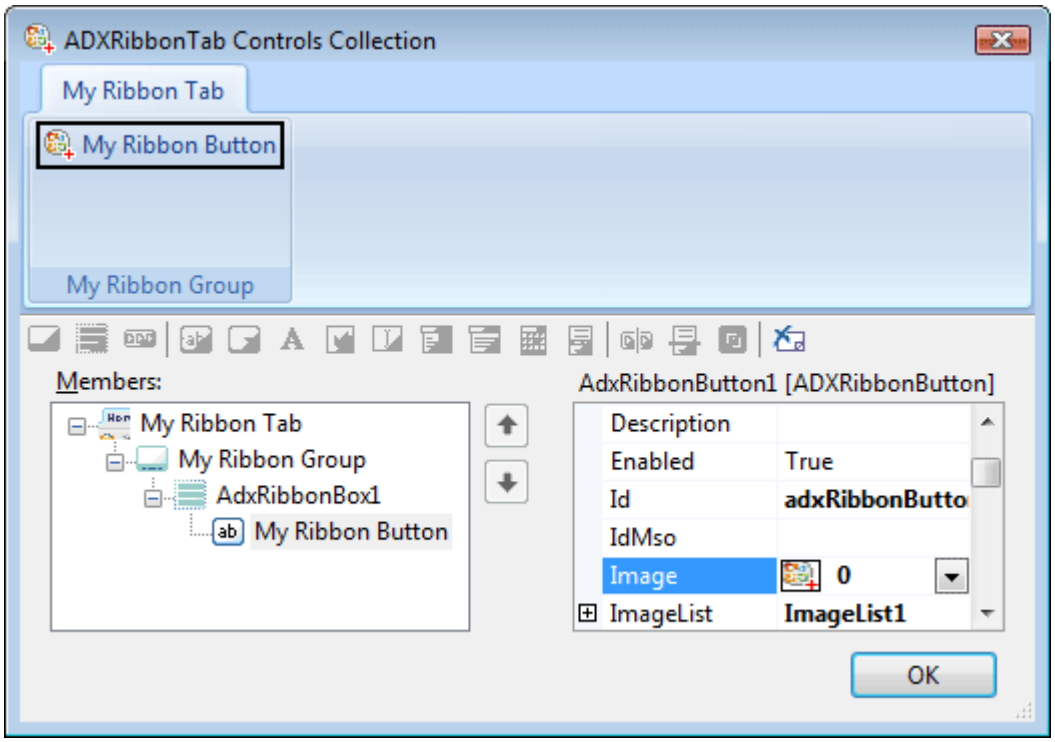
Step #6 – Customizing the Outlook Ribbon User Interface

To add a new tab to the Ribbon UI, you use the Add Ribbon Tab command that adds an ADXRibbonTab component to the module.



In the Properties window, run the editor for the Controls collection of the Ribbon tab component. In the visual designer, use the toolbar buttons or context menu to add or delete Add-in Express components that form the Ribbon interface of your add-in. First, you add a Ribbon tab and change its caption to My Ribbon Tab. Then, you select the tab component, add a Ribbon group, and change its caption to My Ribbon Group. Next, you

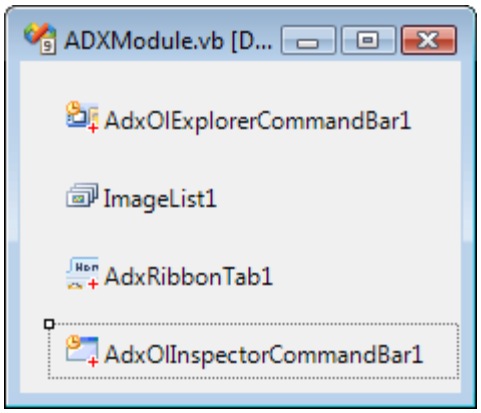
select the group, and add a button group. Finally, you select the button group and add a button. Set the button caption to My Ribbon Button. Use the ImageList, Image, and ImageTransparentColor properties to set the icon for the button.



Remember, the Ribbon Tab visual designer validates the Ribbon XML automatically, so from time to time you will run into the situation when you cannot add a control to some Ribbon level. It is a restriction of the Ribbon XML schema.

Unlike other Ribbon-based applications, Outlook has numerous ribbons. Please use the Ribbons property of your ADXRibbonTab components to specify the ribbons you customize with your tabs.

See also Office Ribbon Components.



Step #7 - Adding a New Inspector Command Bar

To add a command bar to Outlook Inspector windows, use the Add Inspector CommandBar command that adds an ADXOlInspectorCommandBar component to the Add-in Module.

The Inspector command bar component provides the same properties as the Explorer command bar component. If you specify the full path to a folder in the FolderName (FolderNames) property of an inspector command bar component, the corresponding toolbar is displayed for inspectors that open Outlook items the Parent properties of which point to that folder.

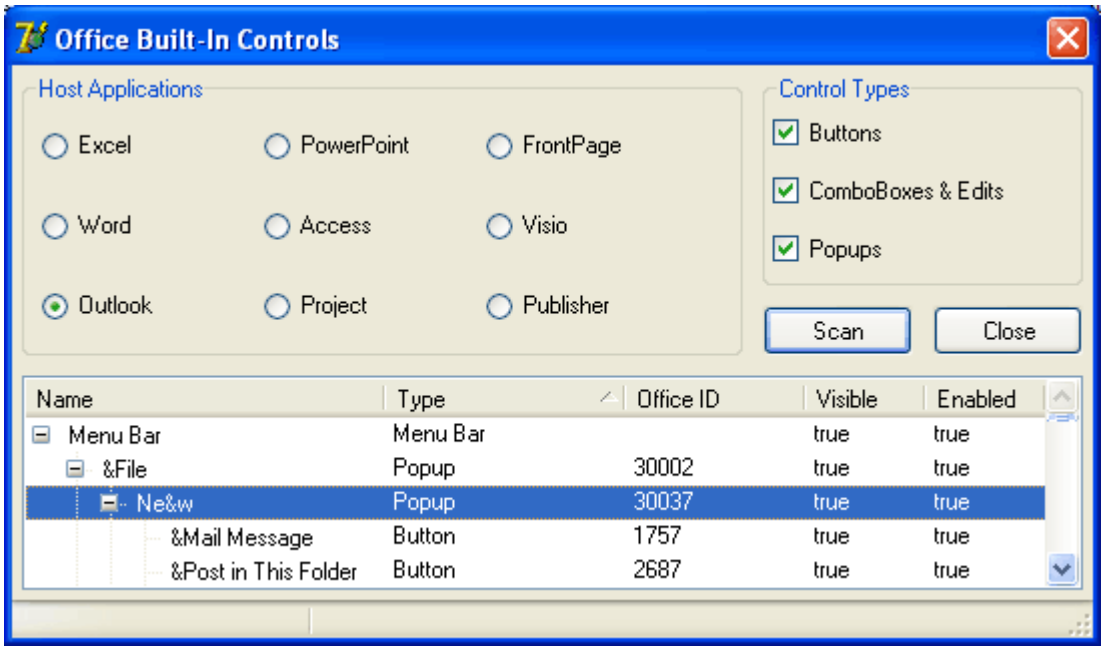
For adding a new command bar button onto the inspector toolbar see Step #5 – Adding a New Command Bar Button for details).

See also Command Bars: Toolbars, Menus, and Context Menus, Outlook CommandBar Visibility Rules and Outlook Add-ins – Template Characters in FolderName.

Step #8 - Customizing Main Menus in Outlook

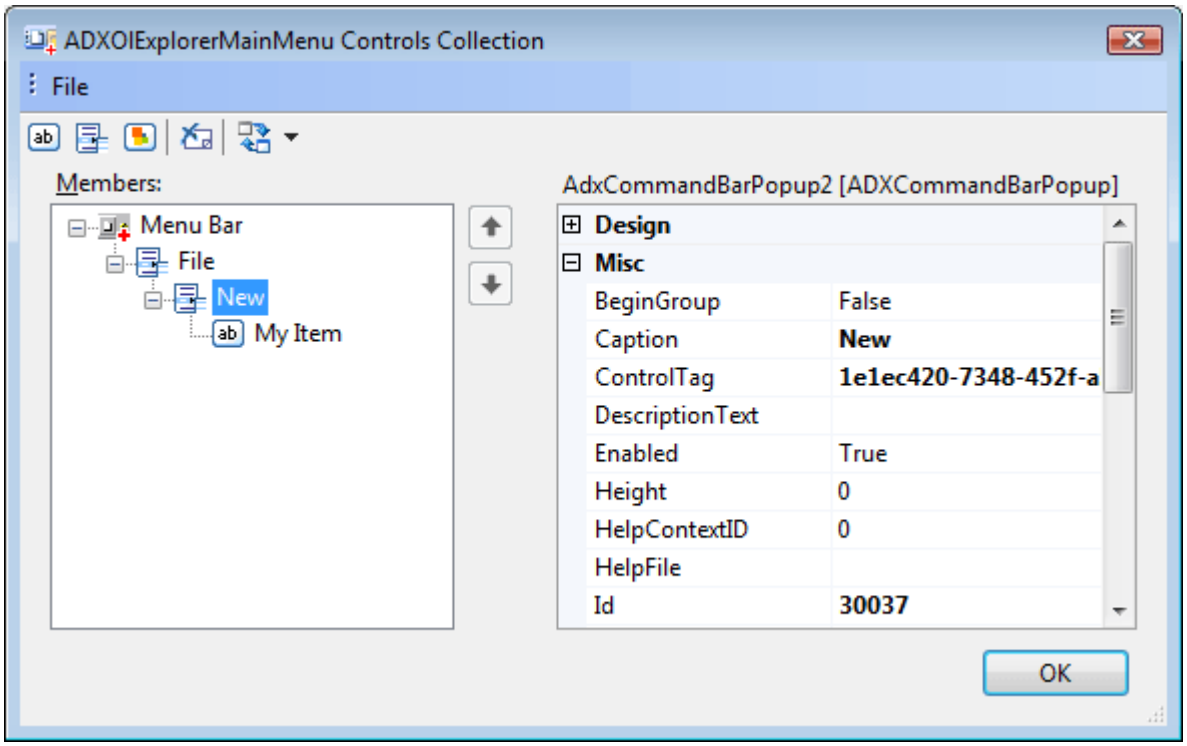
Outlook 2003 provides main menu types. They are available for two main types of Outlook windows: Explorer and Inspector. Accordingly, Add-in Express provides two main menu components: Explorer Main Menu component and Inspector Main Menu component. You add either of them using the context menu of the add-in module. Then you use the visual designer provided for the Controls property of the component.

For instance, to add a custom control to the popup shown by the File | New item in all Outlook Explorer windows, you start our free Built-in Control Scanner to scan the command bars and controls of Outlook. The screenshot below shows the result of scanning. You need the Office IDs you see in the screenshot to bind Add-in Express controls to them:



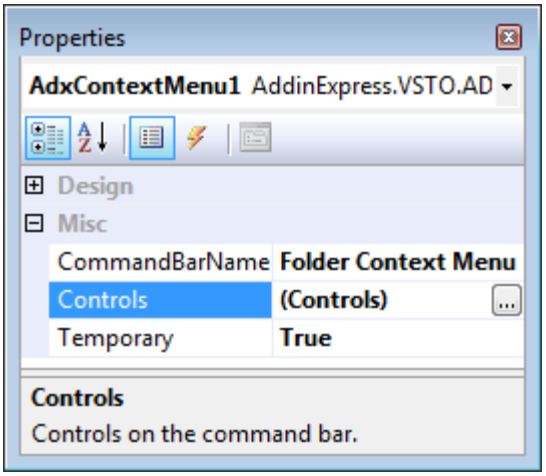
- Add a popup control to the menu component and set its Id property to 30002
- Add a popup control to the popup control above and set its Id to 30037
- Add a button to the popup above and specify its properties.

The following screenshot shows the settings of the popup created at step 3 above:



When testing this sample, pay attention to the Caption property of the New popup above: whatever value it has, it doesn't change the name of the New popup in Outlook. This is how Add-in Express connects to existing command bar controls.

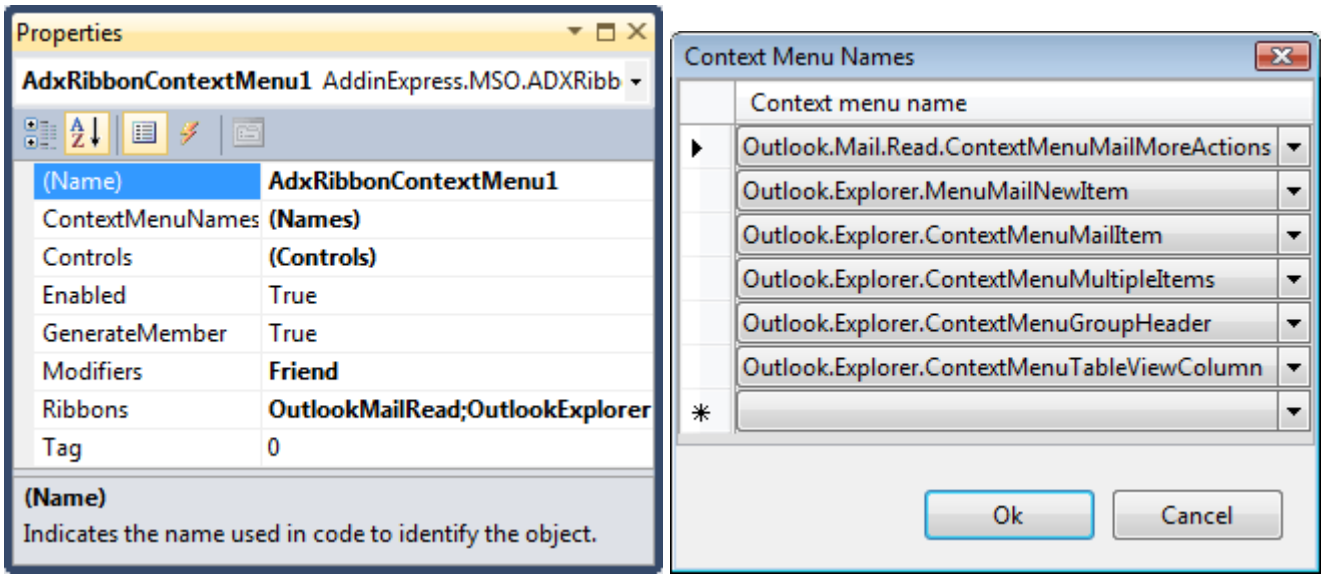
Step #9 - Customizing Context Menus in Outlook



Add-in Express provides the ADXContextMenu component that allows customizing any Outlook context menu. You use this component in the same way as you use the main menu components. Note that the control types available for Office main and context menus are button and popup only.

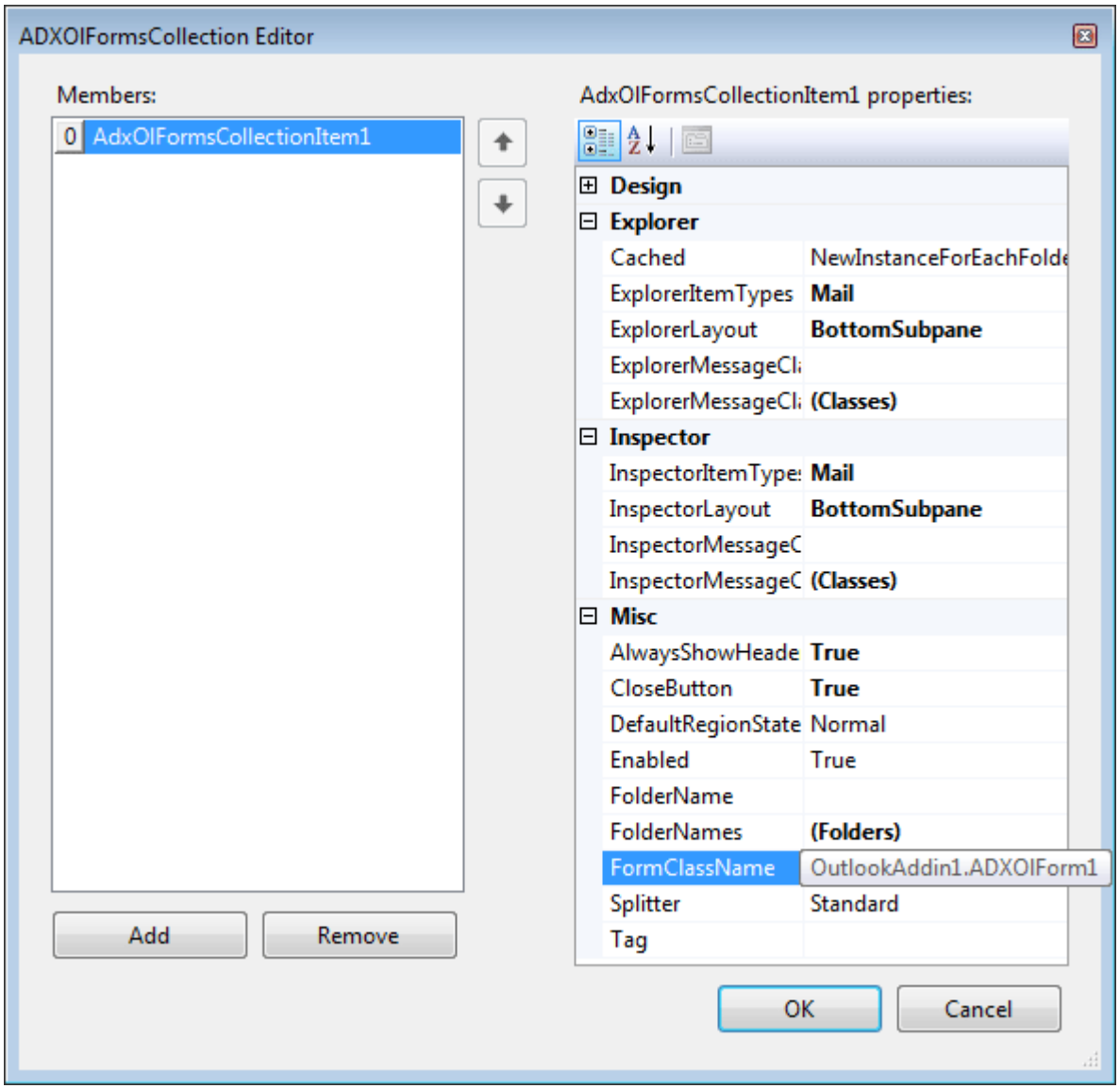
The sample add-in described in this chapter adds a custom item to the Folder Context Menu command bar that implements the context menu which is shown when you right-click a folder in the folder tree.

Also, you can customize many Ribbon-based context menus in Outlook 2010. The Add ADXRibbonContextMenu command of the add-in module adds an ADXRibbonContextMenu component that allows specifying Ribbons that supply context menu names for the ContextMenuNames property. You use the ContextMenuNames property editor to choose the context menu(s) that will display your custom controls specified in the Controls property.



Step #10 – Adding a Custom Task Pane in Outlook 2003-2010

You start with adding an Add-in Express Outlook Form to your project (see Add New Item Dialog). Then you add an Outlook Forms Manager component onto your add-in module (see Adding Components to the Add-in Module). Finally, you add an item to the Items collection of the manager component and set the following properties of the item:



- ExplorerItemTypes = Mail – your form will be shown for all mail folders
- ExplorerLayout = BottomSubpane – the task pane will be shown below the list of mails in Outlook Explorer
- InspectorItemTypes = Mail – an instance of the form will be shown whenever you open an e-mail
- InspectorLayout = BottomSubpane – your task pane will be shown to the right of the message body
- AlwaysShowHeader = True – the header containing the icon (a 16x16 .ico) and the caption of your form will be shown for your form even if it is a single form in the given region
- CloseButton = True – the header will contain the Close button; a click on it generates the OnADXBeforeCloseButtonClick event of the form
- FormClassName = OutlookAddin1.ADXOlForm1 – the class name of the form

See also Step #12 – Handling Outlook Events, Advanced Custom Task Panes and Advanced Outlook Regions.

Step #11- Accessing Outlook Objects

Add the following method to the add-in module:

```
Friend Function GetSubject(ByVal InspectorOrExplorer As Object) As String
    Dim result As String = ""
    If InspectorOrExplorer Is Nothing Then Return result

    Dim outlookItem As Object = Nothing
    Dim mailItem As Outlook.MailItem = Nothing
    Dim selection As Outlook.Selection = Nothing

    If TypeOf InspectorOrExplorer Is Outlook.Explorer Then
        Try
            selection = CType(InspectorOrExplorer, Outlook.Explorer).Selection
            If selection.Count > 0 Then outlookItem = selection.Item(1)
        Catch
        Finally
            If selection IsNot Nothing Then Marshal.ReleaseComObject(selection)
        End Try
    ElseIf TypeOf InspectorOrExplorer Is Outlook.Inspector Then
        Try
            outlookItem = CType(InspectorOrExplorer, _
                Outlook.Inspector).CurrentItem
        Catch
        End Try
    End If

    If outlookItem IsNot Nothing Then
        If TypeOf outlookItem Is Outlook.MailItem Then
            mailItem = CType(outlookItem, Outlook.MailItem)
            result = mailItem.Subject
        End If
        Marshal.ReleaseComObject(outlookItem)
    End If
    Return result
End Function
```

The code of the GetSubject method emphasizes the following:

- Outlook 2007 fires an exception when you try to obtain the Selection object in some situations.
- There may be no items in the Selection object.

For information about the use of Marshal.ReleaseComObject, see Releasing COM objects.

Now select the buttons added in previous steps in the Properties window combo one by one and create the following event handlers:

```
Private Sub DefaultActionInExplorer(ByVal sender As System.Object) _
   Handles AdxCommandBarButton1.Click
    Dim explorer As Outlook.Explorer = Me.OutlookApp.ActiveExplorer
    If explorer IsNot Nothing Then
        MsgBox("The subject is:" _
           + vbCrLf _
           + GetSubject(explorer))
        Marshal.ReleaseComObject(explorer)
    End If
End Sub

Private Sub DefaultActionInInspector(ByVal sender As System.Object) _
         Handles AdxCommandBarButton2.Click, AdxCommandBarButton6.Click
    Dim inspector As Outlook.Inspector = Me.OutlookApp.ActiveInspector
    If inspector IsNot Nothing Then
        MsgBox("The subject is:" _
           + vbCrLf _
           + GetSubject(inspector))
        Marshal.ReleaseComObject(inspector)
    End If
End Sub

Private Sub AdxRibbonButton1_OnClick(ByVal sender As System.Object, _
    ByVal control As AddinExpress.MSO.IRibbonControl, _
    ByVal pressed As System.Boolean) Handles AdxRibbonButton1.OnClick
    DefaultActionInInspector(Nothing)
End Sub
```

Step #12 - Handling Outlook Events

The add-in module provides all application-level Outlook events. For instance, the following code handles the BeforeFolderSwitch event of the Outlook.Explorer class:

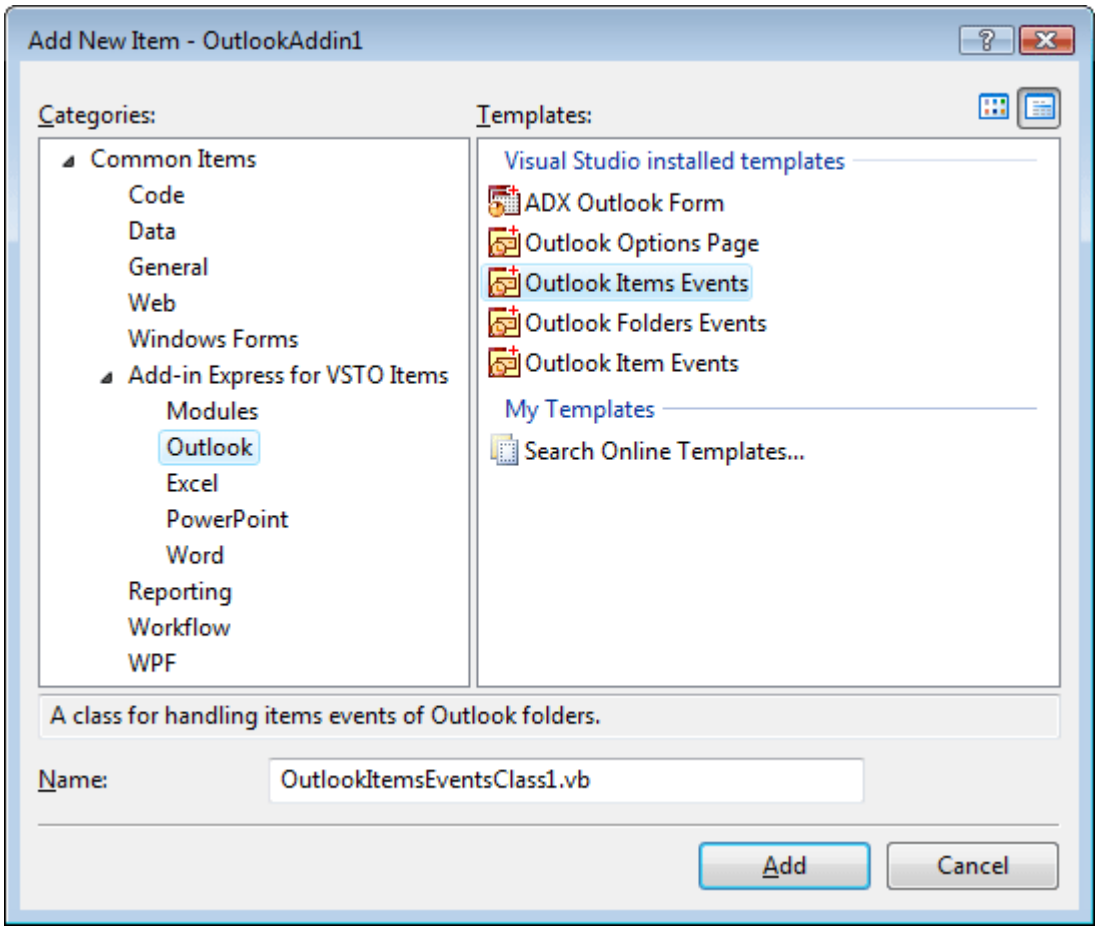
```
Private Sub ADXModule1_ExplorerBeforeFolderSwitch (ByVal sender As Object, _
   ByVal e As AddinExpress.VSTO.ADXOlExplorerBeforeFolderSwitchEventArgs) _
      Handles Me.ExplorerBeforeFolderSwitch
   MsgBox("You are switching to the " + e.NewFolder.Name + " folder")
   'e.Cancel = True
End Sub
```

The form added in Step #10 – Adding a Custom Task Pane in Outlook 2003- also provides a number of Outlook events. Say, you can handle the ADXSelectionChange event as follows (requires adding a label onto the form):

```
Private Sub ADXOlForm1_ADXSelectionChange() Handles MyBase.ADXSelectionChange
    Dim theModule As OutlookAddin1.ADXModule = _
        CType(Me.AddinModule, OutlookAddin1.ADXModule)
    If Me.ExplorerObj IsNot Nothing Then
        Me.Label1.Text = theModule.GetSubject(Me.ExplorerObj)
    ElseIf Me.InspectorObj IsNot Nothing Then
        Me.Label1.Text = theModule.GetSubject(Me.InspectorObj)
    End If
End Sub
```

Step #13 - Handling Events of Outlook Items Object

The Outlook MAPIFolder class provides the Items collection, which has the following events: ItemAdd, ItemChange, and ItemRemove. To process these events, open the Add New Item Dialog and choose the Outlook Items Events item located in the Add-in Express for VSTO Items folder:



This adds the OutlookItemsEventsClass1.vb file to the add-in project. You handle the ItemAdd event by entering some code into the ProcessItemAdd procedure of the OutlookItemsEventsClass1 class:

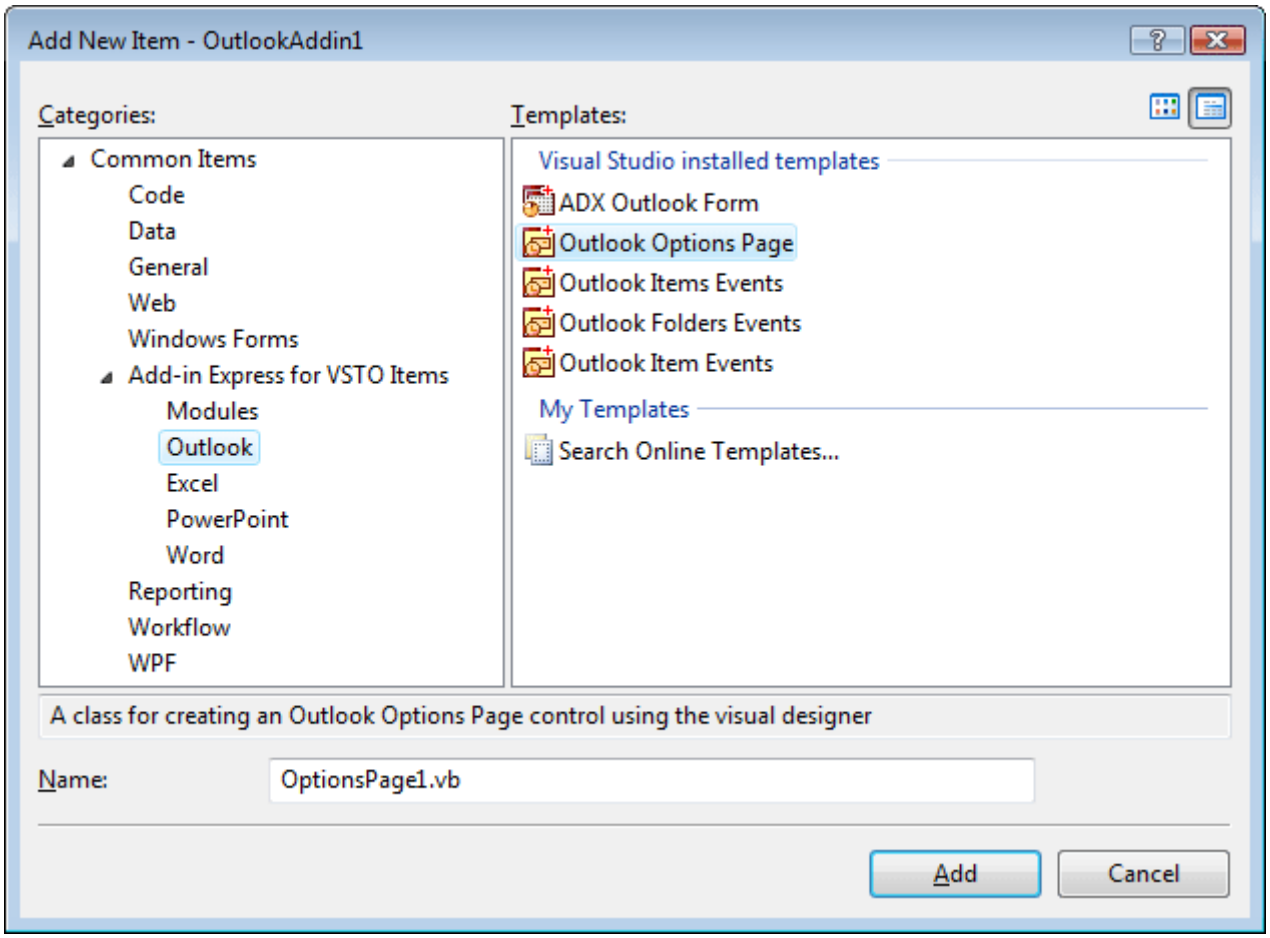
```
Imports System

'Add-in Express for VSTO Outlook Items Events Class
Public Class OutlookItemsEventsClass1
   Inherits AddinExpress.VSTO.ADXOutlookItemsEvents

   Public Sub New(ByVal ADXModule As AddinExpress.VSTO.ADXAddinModule)
      MyBase.New(ADXModule)
   End Sub

   Public Overrides Sub ProcessItemAdd(ByVal Item As Object)
      MsgBox("The item with subject '" + Item.Subject + _
         "' has been added to the Inbox folder")
   End Sub

   Public Overrides Sub ProcessItemChange(ByVal Item As Object)
      'TODO: Add some code
   End Sub

   Public Overrides Sub ProcessItemRemove()
      'TODO: Add some code
   End Sub
End Class
```

This requires adding the following declarations and code to the add-in module:

```
   Dim ItemsEvents As OutlookItemsEventsClass1 = _
         New OutlookItemsEventsClass1(Me)
...
   Private Sub ADXModule_OnBeginShutdown(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.OnBeginShutdown
      If ItemsEvents IsNot Nothing Then
         ItemsEvents.RemoveConnection()
         ItemsEvents = Nothing
      End If
   End Sub

   Private Sub ADXModule_OnStartupComplete(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.OnStartupComplete
      ItemsEvents.ConnectTo( _
            AddinExpress.VSTO.ADXOlDefaultFolders.olFolderInbox, True)
   End Sub
```

Step #14 - Adding Folder Property Pages

Unlike other Office applications, Outlook allows you to add custom option pages to the Options dialog box (the Tools | Options menu) and / or to the Properties dialog box of any folder. To automate this task, Add-in Express provides the Outlook Property Page component. You find it in the Add New Item Dialog box.



Click the Add button to add a new property page instance, a descendant of the ADXOlPropertyPage class that implements the IPropertyPage COM interface:

```
Imports System.Runtime.InteropServices

'Add-in Express for VSTO Outlook Options Page
Public Class OptionsPage1
   Inherits AddinExpress.VSTO.ADXOlPropertyPage

#Region " Component Designer generated code "

   Public Sub New()
      MyBase.New()
```

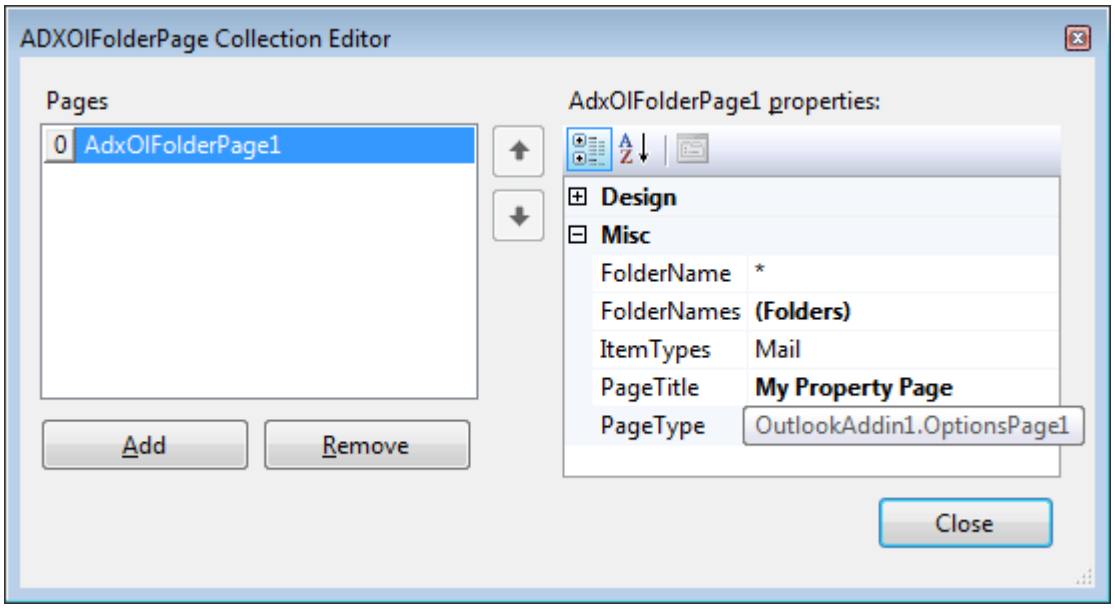
```
        'This call is required by the Component Designer
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Clean up any resources being used
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by designer
    Private components As System.ComponentModel.IContainer

    'Required by designer - do not modify
    'the following method
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
        '
        'OptionsPage1
        '
        Me.Name = "OptionsPage1"
        Me.Size = New System.Drawing.Size(413, 358)
    End Sub

#End Region
End Class
```

You can customize the page as an ordinary form: add controls and handle their events.

To add this property page to the <FolderName> Properties dialog box of an Outlook folder(s), follow the steps below:

- In the add-in module properties, run the collection editor of the FolderPages property
- Click the Add button
- Specify the folder of your choice in the FolderName property
- Set the PageType property to the property page component you've added
- Specify the Title property and close the dialog box

The screenshot below shows the settings you need to have in order to display your page in the Folder Properties dialog for all Mail folders (FolderName = '*' and ItemTypes = Mail).



To show this property page for the Inbox folder only, change the code of the OnStartupComplete event in the add-in module as follows:

```
Private Sub ADXModule_OnStartupComplete(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.OnStartupComplete
    ItemsEvents.ConnectTo( _
        AddinExpress.VSTO.ADXOlDefaultFolders.olFolderInbox, True)
    Dim ns As Outlook.NameSpace = Me.OutlookApp.GetNamespace("Mapi")
    Dim inbox As Outlook.MAPIFolder = _
       ns.GetDefaultFolder(Outlook.OlDefaultFolders.olFolderInbox)
    Dim FullPath As String = inbox.FolderPath
    'remove leading backslashes
    Me.FolderPages.Item(0).FolderName = _
       FullPath.Substring(2, FullPath.Length - 2)
    Marshal.ReleaseComObject(inbox)
    Marshal.ReleaseComObject(ns)
End Sub
```

To control the events of the folder, add a checkbox onto the property page and handle its CheckedChanged event as well as the Dirty, Apply, and Load events of the page as follows:

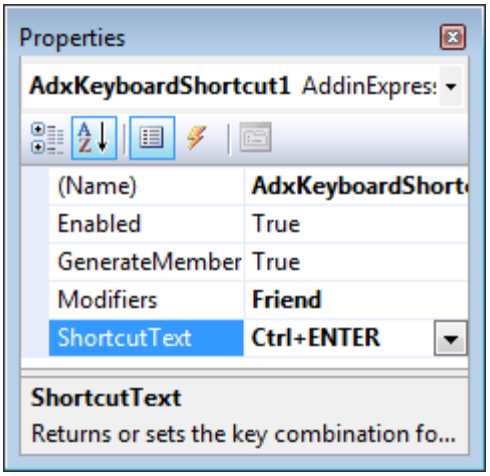
```
   ...
   Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox
   Private TrackStatusChanges As Boolean
   ...
```

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
   If Not TrackStatusChanges Then Me.OnStatusChange()
End Sub

Private Sub OptionsPage1_Apply(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Me.Apply
   CType(AddinExpress.VSTO.ADXOutlookModule.Instance, _
      OutlookAddIn1.ADXModule1).IsFolderTracked = _
         Me.CheckBox1.Checked
End Sub

Private Sub OptionsPage1_Dirty(ByVal sender As Object, _
   ByVal e As AddinExpress.VSTO.ADXDirtyEventArgs) Handles Me.Dirty
   e.Dirty = True
End Sub

Private Sub OptionsPage1_Load(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Me.Load
   TrackStatusChanges = True
   Me.CheckBox1.Checked = _
      CType(AddinExpress.VSTO.ADXOutlookModule.Instance, _
      OutlookAddIn1.ADXModule1).IsFolderTracked
   TrackStatusChanges = False
End Sub
```

Finally, you add the following property to the code in the add-in module:

```
   ...
   Friend Property IsFolderTracked() As Boolean
      Get
         Return ItemsEvents.IsConnected
      End Get
      Set(ByVal value As Boolean)
         If value Then
            ItemsEvents.ConnectTo _
               (AddinExpress.VSTO.ADXOlDefaultFolders.olFolderInbox, True)
         Else
            ItemsEvents.RemoveConnection()
         End If
      End Set
   End Property
```

To add this or any other property page to the main Options dialog box, you use the PageType and PageTitle properties of the add-in module.



Step #15 - Intercepting Keyboard Shortcut

To intercept a keyboard shortcut, you add an ADXKeyboardShortcut component to the add-in module using its Add Keyboard Shortcut command.

In the Properties window, you select (or enter) the desired shortcut in the ShortcutText property, say here is how you intercept the Send button in the Standard command bar of mail inspectors.

To use keyboard shortcuts, you need to set the HandleShortcuts property of add-in module to True.

Now you handle the Action event of the component:

```
Private Sub AdxKeyboardShortcut1_Action(ByVal sender As System.Object) _
   Handles AdxKeyboardShortcut1.Action
   MsgBox("You've pressed " + _
      CType(sender, AddinExpress.VSTO.ADXKeyboardShortcut).ShortcutText)
End Sub
```

Step #16 - Running the Outlook Add-in

Choose the Build <Add-in Project Name> item in the Build menu, then restart Outlook, and find your option page(s), command bars, ribbon tabs, and custom task panes. You find your add-in in the COM Add-ins Dialog.

Step #17 - Debugging the Outlook Add-in

Just press F5 or choose Debug | Start Debugging in the menu.

Step #18 - Deploying the Outlook Add-in

Build the setup project, transfer the files to the target PC and run the setup.exe. See also VSTO Deployment Support in Add-in Express. You can find another bit of useful information in VSTO solution deployment.

VSTO Deployment Support in Add-in Express

Add-in Express simplifies creating and debugging MSI-based setup projects for VSTO add-ins and provides a ClickOnce deployment model of its own.

Files to deploy

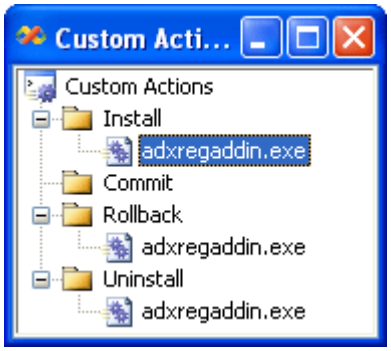
Make sure that your installer delivers the following DLL assemblies to the target PC:

Add-in Express for VSTO	AddinExpress.VSTO.dll
Advanced Outlook Form and View Region	AddinExpress.OL.VSTO.2005.DLL
Advanced Excel Task Pane	AddinExpress.XL.VSTO.2005.DLL
Advanced Word Task Pane	AddinExpress.WD.VSTO.2005.DLL
Advanced PowerPoint Task Pane	AddinExpress.PP.VSTO.2005.DLL
Advanced Commandbar Control	AddinExpress.ToolbarControls.VSTO.2005.dll

There's also an optional DLL - intResource.dll (intResource64.dll). It ensures compatibility between various Add-in Express add-ins. If it doesn't exist in the add-in folder, it gets unpacked to the Temporary Files folder and loaded into the host application.

MSI Deployment

When deploying VSTO-based solutions, you have to follow Microsoft recommendations. Add-in Express makes easier for you to follow some of them. When you create a project, Add-in Express automatically adds ready-to-use custom actions to the setup project. The custom actions are provided by a special .NET assembly called Add-in Registrator (adxregaddin.exe). It is located in the Redistributables folder of Add-in Express setup folder.



- Install

```
/install="[TARGETDIR]\MyVstoAddin1.manifest" /displayerrors=1
```

- Rollback

```
/uninstall="[TARGETDIR]\MyVstoAddin1.manifest"
```

- Uninstall

```
/uninstall="[TARGETDIR]\MyVstoAddin1.manifest"
```

When you add-in is installed, the registrator creates the User / Code groups / All code / <Add-in Module GUID> <ProgId> code group, and grants it the "Full Trust" permission. You can find the group in Control Panel / Administrative Tools / .NET Framework 2.0 configuration.

Also, Add-in Express specifies the following command line in the PostBuildEvent property of the setup project:

```
"%AddinExpressInstallFodler%\Bin\DisableUAC.exe" "$(BuiltOuputPath)" /UAC=Off
```

If you set the /UAC parameter to On, the setup will require administrative privileges, when run by a non-admin user on Vista. In addition, DisableUAC.exe disables the "Everyone / Just Me" choice in the setup UI. Note that the DefaultLocation property of the setup project targets to [AppDataFolder] and not to [ProgramFilesFolder].

ClickOnce Deployment

ClickOnce Overview

What follows below is a brief compilation of the following Internet resources:

- ClickOnce article from Wikipedia
- ClickOnce FAQ on windowsclient.net
- Introduction to ClickOnce deployment on msdn2.microsoft.com (also compares ClickOnce and MSI)
- ClickOnce Deployment in .NET Framework 2.0 on 15seconds.com

ClickOnce is a deployment technology introduced in .NET Framework 2.0. Targeted to non-administrator-privileges installations it also allows updating your applications. Subject to many restrictions, it isn't a panacea in any way. Say, if your prerequisites include .NET Framework 2.0 and the user doesn't have it installed, your application (as well as an add-in) will not be installed without administrator privileges. In addition, ClickOnce will not allow installing shared components, such as custom libraries. It is quite natural, though.

When applied to a Windows forms application, ClickOnce deployment implies the following steps:

- Publishing an application

You deploy the application to either File System (CD/DVD included) or Web Site. The files include all application files as well as application manifest and deployment manifest. The application manifest describes the application itself, including the assemblies, dependencies and files that make up the application, required permissions, and the location where updates will be available. The deployment manifest describes how the application is deployed, including the location of the application manifest, and the version of the application that

the user should run. The deployment manifest also contains an update location (a Web page or network file share) where the application checks for updated versions. ClickOnce Publish properties are used to specify when and how often the application should check for updates. Update behavior can be specified in the deployment manifest, or it can be presented as user choices in the application's user interface by means of the ClickOnce API. In addition, Publish properties can be employed to make updates mandatory or to roll back to an earlier version.

- Installing the application

The user clicks a link to the deployment manifest on a web page, or double-clicks the deployment manifest file in Windows Explorer. In most cases, the end user is presented with a simple dialog box asking the user to confirm installation, after which installation proceeds and the application is launched without further intervention. In cases where the application requires elevated permissions, the dialog box also asks the user to grant permission before the installation can continue. This adds a shortcut icon to the Start menu and lists the application in the Control Panel/Add Remove Programs. Note, it does not add anything to the registry, the desktop, or to Program Files. Note also that the application is installed into the ClickOnce Application Cache (per user).

- Updating the application

When the application developer creates an updated version of the application, they also generate a new application manifest and copy files to a deployment location—usually a sibling folder to the original application deployment folder. The administrator updates the deployment manifest to point to the location of the new version of the application. When the user opens the deployment manifest, it is run by the ClickOnce loader and in this way updates the application.

Add-in Express ClickOnce Solution

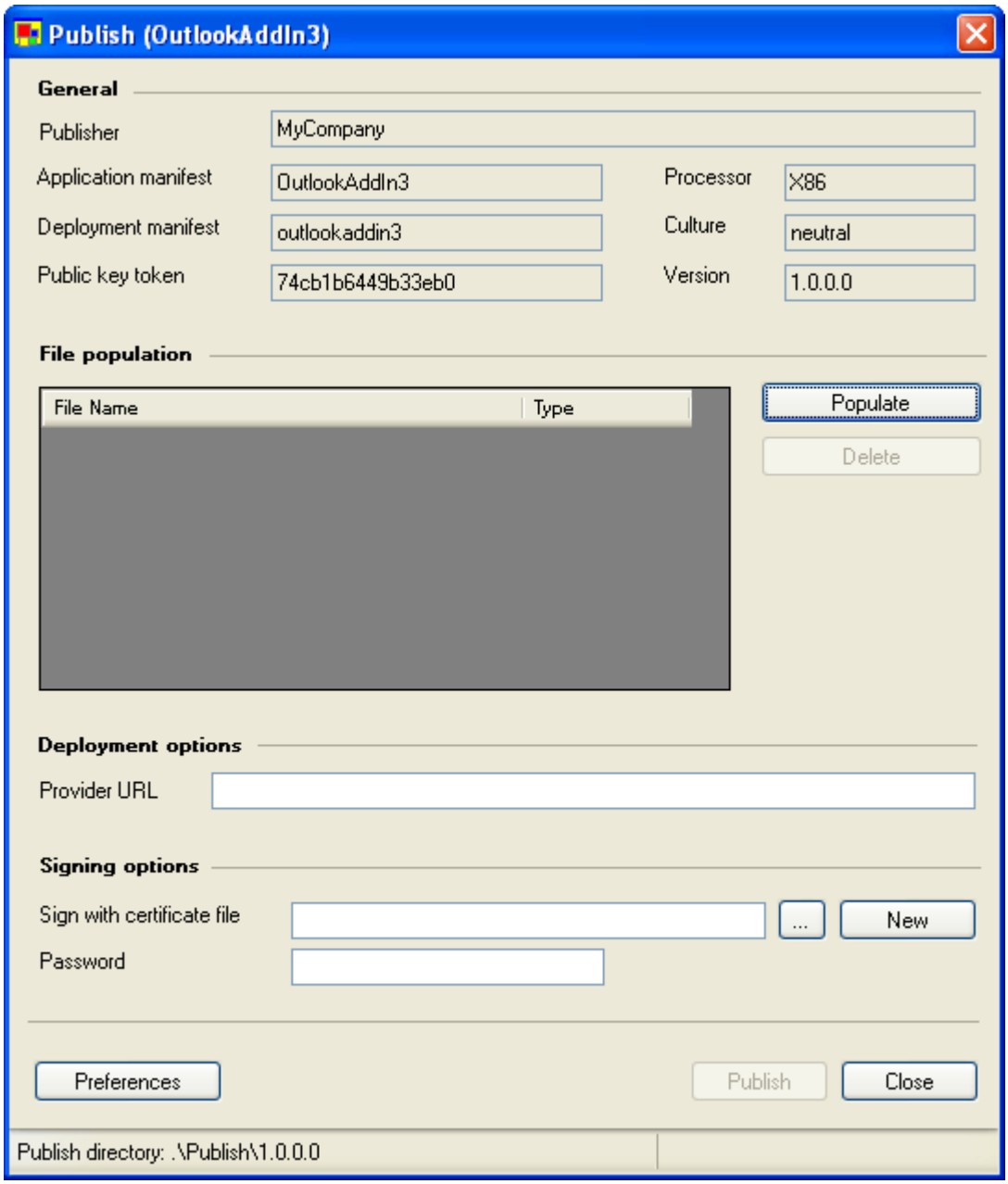
Add-in Express adds the Publish Add-in Express Project item to the Build menu in Visual Studio 2005 and 2008. When you choose this item, Add-in Express shows the Publish dialog that generates the deployment manifest and places it into the Publish subfolder of the solution folder. In addition, the dialog generates the application manifest and places it to the Publish / <AssemblyVersion> folder. Then the dialog copies the add-in files and dependencies (as well as the Add-in Express loader and its manifest) to the same folder.

One more file copied to the Publish / <AssemblyVersion> folder is called the Add-in Express Launcher for ClickOnce Applications or the launcher. Its file name is adxlauncher.exe. This file is the heart of the Add-in Express ClickOnce Solution. The launcher is a true ClickOnce application. It will be installed on the user's PC and listed in the Start menu and Add / Remove Programs. The launcher registers and unregisters your add-in, and it provides a form that allows the user to register, unregister, and update your add-in. It also allows the user to switch between two latest versions of your add-in. Overall, the launcher takes upon itself the task of communicating with the ClickOnce API.

The launcher (adxlauncher.exe) is located in the Redistributables folder of the Add-in Express setup folder. You can check its properties (name, version, etc) in Windows Explorer. Subsequent Add-in Express releases will replace this file with its newer versions. And this may require you to copy a new Launcher version to your Publish / <AssemblyVersion> folder.

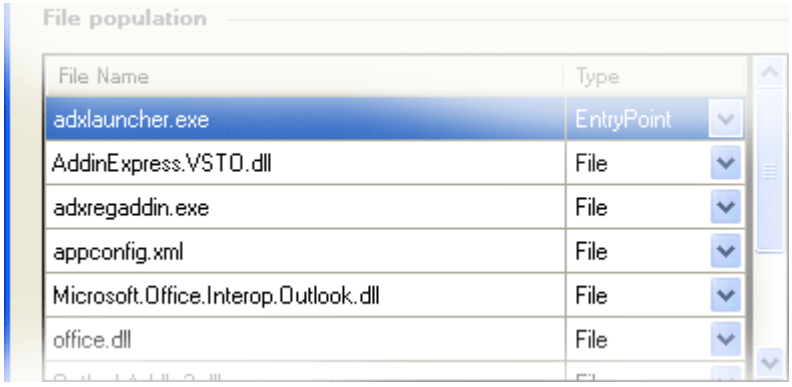
On the Development PC

The Add-in Express Publish dialog helps you create application and deployment manifests. In the current release, it shows the following form:



Step #1 – Populating the Application Manifest

Just click the Populate button. This is the moment when all the above-mentioned folders are created and files are copied.



To set a custom icon for the launcher, you can add a .ico file and mark it as Icon File in the Type column of the File Population list box.

How do I add additional files to the application manifest?

The current release doesn't provide the user interface for adding additional files and/or folders. However, you can copy the files and/or folders required by your add-in to the Publish / <AssemblyVersion> folder and click the Populate button again.

Step #2 - Specifying the Deployment / Update Location

You fill the Provider URL textbox with the URL of your deployment manifest (remember, it is located in the Publish folder). For Web-site based deployment, the format of the URL string is as follows:

```
http://<WebSitePath>/<deployment manifest name>.application
```

Case-dependent

Please note that <deployment manifest name> must be entered in lower case. You can copy it from the Deployment manifest textbox in the Publish dialog window.

Say, you can create a Virtual Directory on your IIS server and bind it to the folder where your deployment manifest is located. For testing purposes, we recommend using the Publish folder. In this case, the Provider URL could be like this:

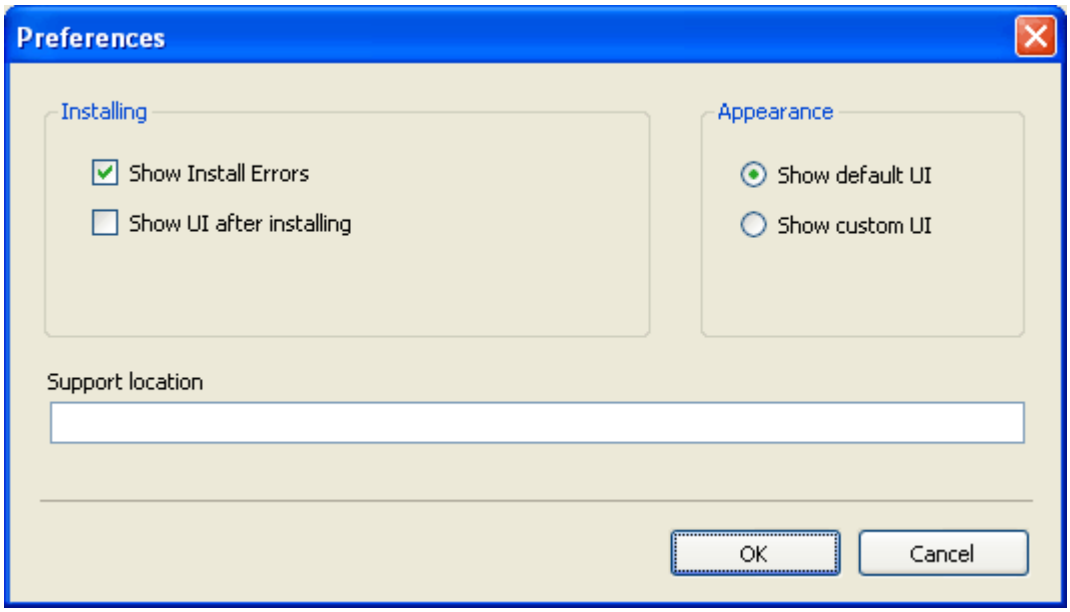
```
http://localhost/clickonceoutlook/outlookaddin3.application
```

Step #3 - Signing the Manifests

Browse the existing certificate file or click New to create a new one. Enter the password for the certificate (optional).

Step #4 - Preferences

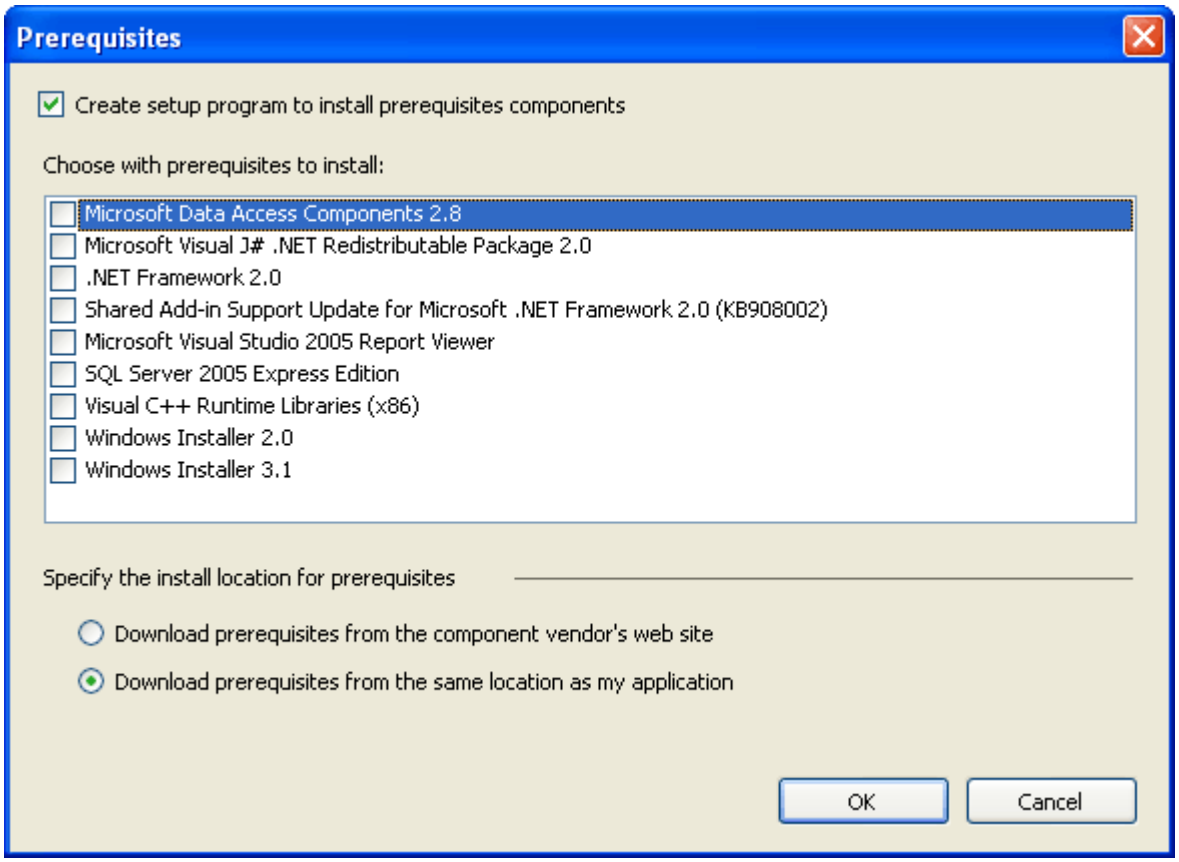
Click the Preferences button to open the following dialog window:



In this dialog, you specify if the ClickOnce module will get the OnShowCustomUI event (it allows the add-in to show the custom UI), and provide the Support Location option for the Add Remove Programs dialog.

Step #5 - Prerequisites

When you click this button and select any prerequisites in the dialog, Add-in Express gathers the prerequisites you've specified and creates a setup.exe to install them. Then you can upload all files to any appropriate location. When the user starts the setup.exe, it installs the prerequisites and invokes the ClickOnce API to install your add-in. Naturally, it may happen that a prerequisite can be installed by the administrator only. In this case, you may want to create a separate setup project that installs that prerequisites only and to supply it to the administrator.



Step #6 – Publishing the Add-in

When you click on the Publish button, Add-in Express generates (updates) the manifests.

Deployment manifest - <SolutionFolder>/Publish/<projectname>.application

Application manifest - <SolutionFolder>/Publish/<ProjectVersion>/<ProjectName>.exe.manifest

Now you are able to copy files and folders of the Publish folder to the deployment location, say a web server. Please note, for testing purposes, you can just double-click the deployment manifest.

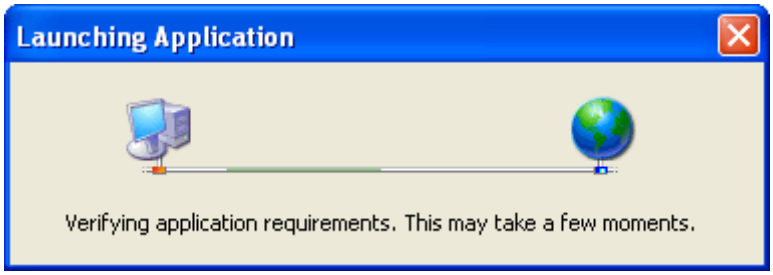
Step #6 – Publishing a New Add-in Version

In AssemblyInfo, change the version number and build the project. Click Publish and add the add-in files (button Populate). Fill in all the other fields. You can use the Version check box to switch to the data associated with any previous version.

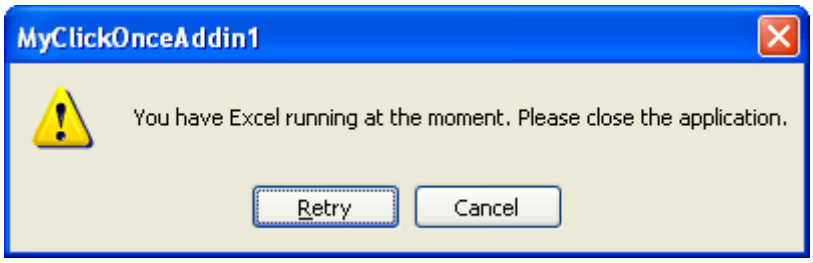
On the Target PC

Installing: User Perspective

The user browses the deployment manifest (<projectname>.application) in either Internet Explorer or Windows Explorer and runs it. The following window is shown:

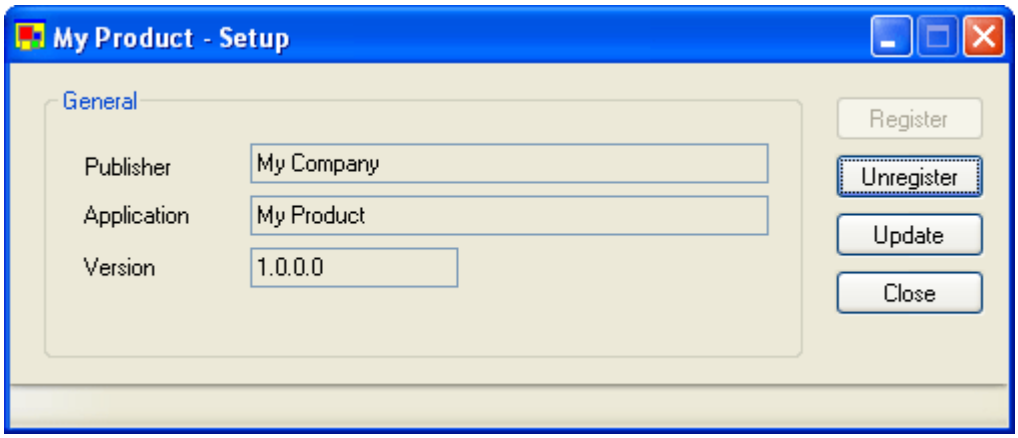


In accordance with the manifests, the ClickOnce loader will install the files and run the Launcher application. When run in this mode, it registers the add-in. If the add-in's host application is running at this moment, the user will be prompted to close it.



If the user clicks Cancel, the Launcher will be installed, but the add-in will not be registered. However, in any appropriate moment, the user can click the Launcher entry in the Start menu to run the Launcher and register/unregister the add-in through the Launcher GUI.

The current release relies on the name and location of the product entry in the Start Menu. Please, add this information to your user's guide.



Installing: Developer Perspective

If a ClickOnce module is added to your add-in project, you are able to handle all the actions applicable to add-ins: install, uninstall, register, unregister, update to a newer version, and revert to the previous version. For instance, you can easily imagine a form or wizard allowing the user to tune up the settings of your add-in. The ClickOnce module also allows you to show a custom GUI whenever the Launcher Application is required to show its GUI. Please note that if you don't process the corresponding event, the standard GUI of the Add-in Express ClickOnce application will be shown.

You can also make use of the ComRegisterFunction and ComUnRegisterFunction attributes in any assembly listed in the loader manifest (see `assemblyIdentity` tags). The methods marked with the ComRegisterFunction attribute will run when the add-in is registered. See MSDN for the description of the attributes.

Updating: User Perspective

The user can check for add-in updates in the Launcher GUI (or in the GUI supplied by you). To run it, the user clicks the entry in the Start Menu. If there is no update in the update location specified in the deployment manifest, an information message box is shown. If there is an update, the ClickOnce Loader updates files in the ClickOnce Cache, the Launcher unregisters the current add-in version, restarts itself (this will run the Launcher application supplied in the update files), and registers the add-in.

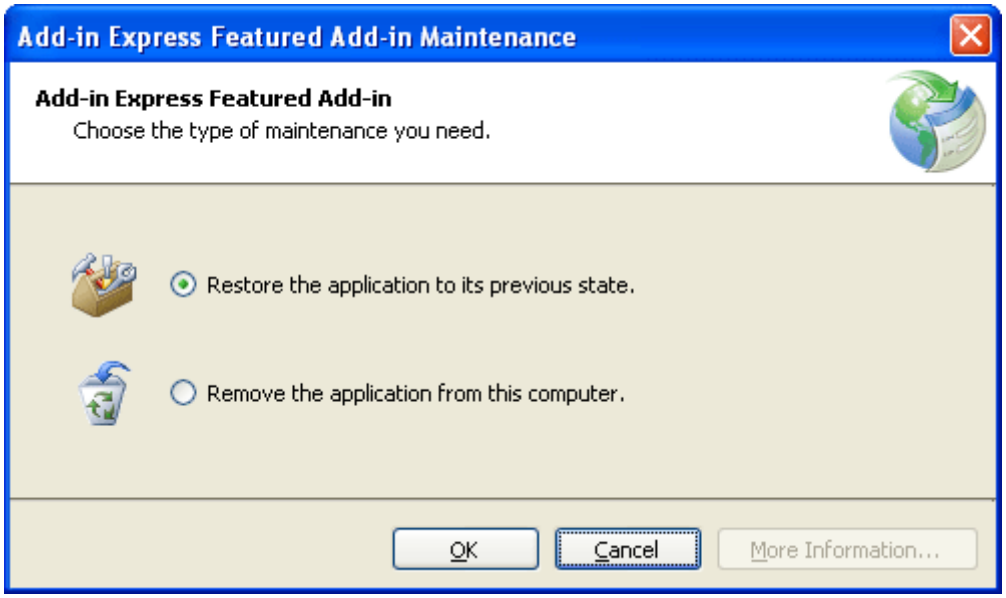
Updating: Developer Perspective

The add-in module provides you with the CheckForUpdates method. This method can result in one of the following ways:

- the add-in becomes updated;
- the ClickOnce module invokes the OnError event handler.

Uninstalling: User Perspective

To uninstall the add-in, the user goes to Add Remove Programs and clicks on the product name entry. This opens the following dialog.



- Restore the application to its previous state.

This option is disabled, if the add-in was never updated. If the user chooses this option, the Launcher is run, then it requires the user to close the host applications of your add-in, unregisters the add-in, requests ClickOnce API to start the Launcher application of the previous add-in version, and quits. After that the Launcher application of the previous add-in version registers the add-in.

- Remove the application from this computer

This runs the Launcher that will require the user to close the host applications of your add-in. Then the Launcher unregisters the add-in and requests the ClickOnce API to delete both the add-in and the Launcher files.

Uninstalling: Developer Perspective

Handle the corresponding event of the ClickOnce module or use the ComUnRegisterFunction attribute to run your actions when the add-in is unregistered.

Restrictions of Add-in Express ClickOnce Solution

In the Web-based deployment scenario, the user can install such add-ins using Internet Explorer only. The ClickOnce article from Wikipedia states that Firefox allows ClickOnce-based installations too, but this was neither tested nor even verified.

Several notes

Don't you have an impression that creating add-ins is a very simple task? Sure, Add-in Express makes embedding your code into Office applications very simple, but you should write the applied code yourself, and we guess it would be something more complex than a single call of MessageBox.

Here we describe some tips and important issues you will need when developing your add-ins.

Terminology

In this document, on our site, and in all our texts we use the terminology suggested by Microsoft for all toolbars, their controls, and for all interfaces of the Office Type Library. For example:

- Command bar is a toolbar, a menu bar, or a context menu.
- Command bar control is one of the following: a button, an edit box, a combo box, or a pop-up.
- Pop-up can stand for a pop-up menu, a pop-up button on a command bar or a submenu on a menu bar.

Add-in Express uses interfaces from the Office Type Library. We do not describe them here. Please refer to the VBA help and to application type libraries.

Getting Help on COM Objects, Properties and Methods

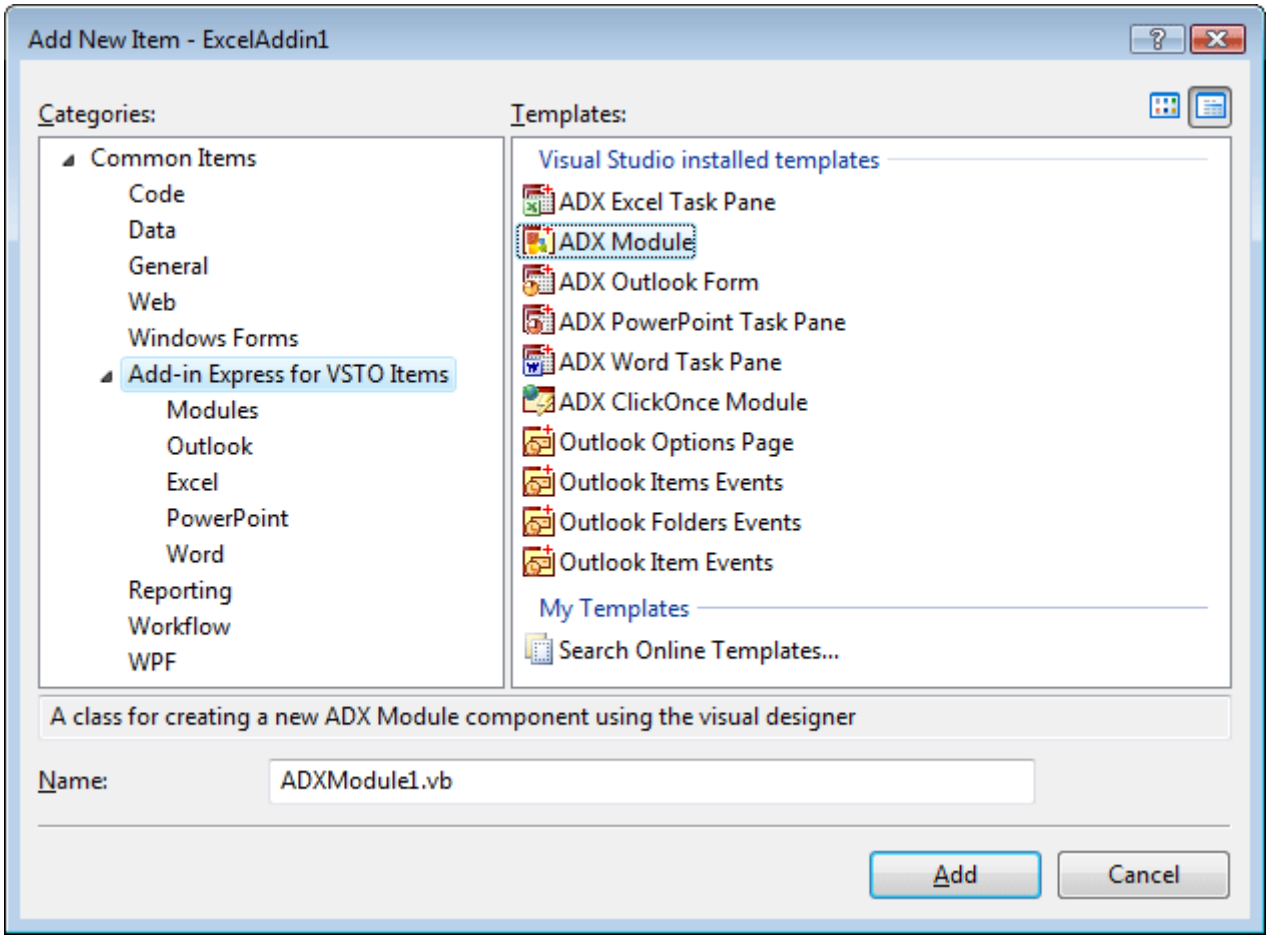
To get assistance with host applications' objects, their properties and methods as well as help info, use the Object Browser. Go to the VBA environment (in the host application, choose menu Tools / Macro / Visual Basic Editor or just press Alt+F11), press F2, select the host application (also Office and MSForms) in the topmost combo and/or specify a search string in the search combo. Select a class/property/method and press F1 to get the help topic that relates to the object. Those helps were written in the times when the one-sentence standard was unknown.

Add New Item Dialog

Add-in Express for VSTO adds several new templates to this dialog:

- Add-in Express Excel Task Pane – a form designed for being embedded into Excel windows. See Excel Task Panes.
- Add-in Express Module – the core of any Add-in Express COM add-in. See Add-in Express Module.
- Add-in Express Outlook Form – a form designed for being embedded into Outlook Explorer and Inspector windows. See Advanced Outlook Regions.
- Add-in Express PowerPoint Task Pane – a form designed for being embedded into PowerPoint.
- Add-in Express Word Task Pane – a form designed for being embedded into Word documents.

- Add-in Express ClickOnce Module – allows accessing ClickOnce-related features in ClickOnce Deployment.
- Outlook Options Page – the form designed for extending Outlook Options and Folder Properties dialogs with custom pages. See Outlook Property Page and Your First Microsoft Outlook Add-in.
- Outlook Items Events – provides easy access to the events of the Items class of Outlook (see Event Classes).
- Outlook Folders Events – provides easy access to the events of the Folders class of Outlook (see Event Classes).
- Outlook Item Events – provides easy access to the events of the MailItem, TaskItem, ContactItem, etc classes of Outlook (see Event Classes).



Add-in Module Commands

The following commands add the following components to the module:

- Add CommandBar – adds a command bar to your add-in (see Command Bars: Toolbars, Menus, and Context Menus)
- Add Explorer CommandBar – adds an Outlook Explorer command bar to your add-in (see Command Bars: Toolbars, Menus, and Context Menus)
- Add Inspector CommandBar – adds an Outlook Inspector command bar to your add-in (see Command Bars: Toolbars, Menus, and Context Menus)
- Add Built-in Control Connector – adds a component that allows intercepting the action of a built-in control of the host application(s) (see Built-in Control Connector)
- Add Keyboard Shortcut– adds a component that allows intercepting application-level keyboard shortcuts (see Keyboard Shortcut)
- Add Outlook Bar Shortcut Manager – adds a component that allows adding Outlook Bar shortcuts and shortcut groups (see Outlook Bar Shortcut Manager)
- Add Outlook Forms Manager – adds a component that allows embedding custom .NET forms into Outlook windows (see Advanced Outlook Regions)
- Add Ribbon Tab – adds a Ribbon tab to your add-in (see Office Ribbon Components)
- Add Ribbon Quick Access Toolbar – adds a component that allows customizing the Ribbon Quick Access Toolbar in your add-in (see Office Ribbon Components)
- Add Ribbon Office Menu – adds a component that allows customizing the Ribbon Office Menu in your add-in (see Office Ribbon Components)

Downloading Sample Projects

You can download the sample projects described in this manual using the following links:

Excel:

- VB.NET (VS2008, Office 2003) - http://www.add-in-express.com/support/addin-vb-net.php#office-addin
- C# (VS2008, Office 2003) - http://www.add-in-express.com/support/addin-c-sharp.php#office-addin

Outlook:

- VB.NET (VS2008, Office 2003) - http://www.add-in-express.com/support/addin-vb-net.php#outlook-addin
- C# (VS2008, Office 2003) - http://www.add-in-express.com/support/addin-c-sharp.php#outlook-addin

When you open the projects with Office 2007 installed on your PC, VS 2008 may run the Conversion Wizard that replaces Office 2003 PIAs with Office 2007 ones. To bypass this, you may want to uncheck the "Always upgrade to installed version of Office" flag located in Tools | Options | Office Tools | Project Upgrade.

COM Add-ins Dialog

In version 2007 of Word, Excel, PowerPoint and Access you click the Office Menu button, then click {Office application} options and choose the Add-ins tab. Now choose COM Add-ins in the Manage dropdown and click Go.

In Office 2003, you need to add the COM Add-ins command to a toolbar or menu of your choice. To do so, follow the steps below:

- Open the host application (Outlook, Excel, Word, etc)
- On the Tools menu, click Customize.
- Click the Commands tab.
- In the Categories list, click the Tools category.
- In the Commands list, click COM Add-Ins and drag it to a toolbar or menu of your choice.

How to Get Access to the Add-in Host Applications

For your convenience, the Add-in Module provides host-related properties to the Add-in module, such as OutlookApp and ExcelApp. Also, the ThisApplication property returns the ThisApplication property of the VSTO add-in.

Registry Entries

COM Add-ins registry entries are located in the following registry branches:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\<OfficeApplication>\AddIns\<Add-in ProgID>
```

Outlook CommandBar Visibility Rules

Add-in Express displays the Explorer command bar for every folder, which name **AND** type correspond to the values of FolderName, FolderNames, and ItemTypes properties. For the inspector toolbar, the same rule applies to the folder in which an Outlook Item is opened or created.

Event classes

An Event class cannot be connected to several event sources (say, Items collections of several folders). Instead, you create several instances of the Event class.

Wait a Little

Some things aren't possible right at the moment; say, you can't close the inspector of an Outlook item in the Send event of that item. A widespread approach is to use a timer. Add-in Express provides a way to do this by using the <SendMessage> method and <OnSendMessage> event; when you call <SendMessage>, it posts the Windows message that you specified in the methods' parameters and the execution continues. When Windows delivers this message to an internal window, Add-in Express raises the <OnSendMessage> event. Make sure that you filter incoming messages; there will be quite a lot of them.

The actual names of the <SendMessage> method and <OnSendMessage> event are listed below:

<SendMessage>

- ADXAddinModule.SendMessage
- ADXOlForm.ADXPostMessage
- ADXExcelTaskPane.ADXPostMessage
- ADXWordTaskPane.ADXPostMessage
- ADXPowerPointTaskPane.ADXPostMessage

<OnSendMessage>

- ADXAddinModule.OnSendMessage
- ADXOlForm.ADXPostMessageReceived
- ADXExcelTaskPane.ADXPostMessageReceived
- ADXWordTaskPane.ADXPostMessageReceived
- ADXPowerPointTaskPane.ADXPostMessageReceived

ControlTag vs. Tag

Add-in Express identifies all its controls (command bar controls) using the ControlTag property (the Tag property of the Office.CommandBarControl interface). The value of this property is generated automatically and you don't need to change it. For your own needs, use the Tag property instead.

Pop-ups

According to the Microsoft's terminology, the term "pop-up" can be used for several controls: pop-up menu, pop-up button, and submenu. With Add-in Express you can create your own pop-up as a command bar control and populate it using the Controls property.

But pop-ups have a very annoying feature: if an edit box or a combo box is added to a pop-up, their events are fired very oddly. Don't regard this bug as that of Add-in Express. It looks like it was introduced by MS intentionally.

CommandBar.Position = adxMsoBarPopup

This option allows displaying the command bar as a popup (context) menu. In the appropriate event handler, you write the following code:

```
AdxOlExplorerCommandBar1.CommandBarObj.GetType.InvokeMember("ShowPopup", _
    Reflection.BindingFlags.InvokeMethod, Nothing, _
    AdxOlExplorerCommandBar1.CommandBarObj, Nothing)
```

The same applies to other command bar types.

CommandBar.Position = adxMsoBarMenuBar

This option can be used in some scenarios with Excel solutions only.

Edits, Combos, and the Change Event

The Change event appears only after the control's value is changed **AND** the focus is shifted. This is not our bug either, but MS guy's "trick".

Removing Custom Command Bars and Controls

Add-in Express removes custom command bars and controls while the add-in is uninstalled. However, this doesn't apply to Outlook and Access add-ins. You should set the Temporary property of custom command bars (and controls) to true to notify the host application that it can remove them itself. If you need to remove a toolbar or button yourself, use the Tools | Customize dialog.

Temporary or Not?

According to the help reference for the Office object model contained within Office.DLL, temporary command bars and controls are removed by the host application when it is closed.

In the common case, the developer has the following alternative: if command bars and controls are temporary, they are recreated whenever the add-in starts; if they are non-temporary, the installer removes those command bars and controls from the host. Looking from another angle, you will see that the real alternative is the time required for start-up against the time required for uninstalling the add-in (the host must be run to remove command bars).

Outlook and Word are two exceptions. It is strongly recommended that in Outlook add-ins you use temporary command bars and controls. If they are non-temporary, Outlook will be run to remove them. Now imagine password-protected PST and multiple-profiles scenarios.

In Word add-ins, we strongly advise making **both** command bars and controls non-temporary. Word removes temporary command bars. However, it doesn't remove temporary command bar controls, at least some of them. When the add-in starts for the second time, Add-in Express finds such controls and just connects to them. In this way it processes the user-moved-or-deleted-the-control scenario. Accordingly, the controls are missing in the UI.

Note that main and context menus are command bars. That is, in Word add-ins, custom controls added to these components must have `Temporary = False`, too. If you set `Temporary` to true for such controls, they will not be removed when you uninstall your add-in. That happens because Word has another peculiarity: it saves temporary controls when they are added to a built-in command bar. And all context menus are built-in command bars. To remove such controls, you will have to write some code or use a simple way: set Temporary to false for all controls, register the add-in on the affected PC, run Word. At this moment, the add-in finds this control and traces it from this moment on. Accordingly, when you unregister the add-in, the control is removed in a standard way.

Built-in Controls and Command Bars

You can connect an ADXCommandBar instance to any built-in command bar. For example, you can add your own controls to the "Standard" command bar or remove some controls from it. To do this, just add to the add-in module a new ADXCommandBar instance and specify the name of the built-in command bar you need via the CommandBarName property.

Also, you can add a built-in control to a custom command bar: just add an ADXCommandBarControl to the ADXCommandBar.Controls collection and specify the Id of the required built-in control in the ADXCommandBarControl.Id property. Pay attention that in this case, you add the built-in control onto the command bar but you do not handle its events. To handle events of built-in controls use the ADXBuiltInControl component.

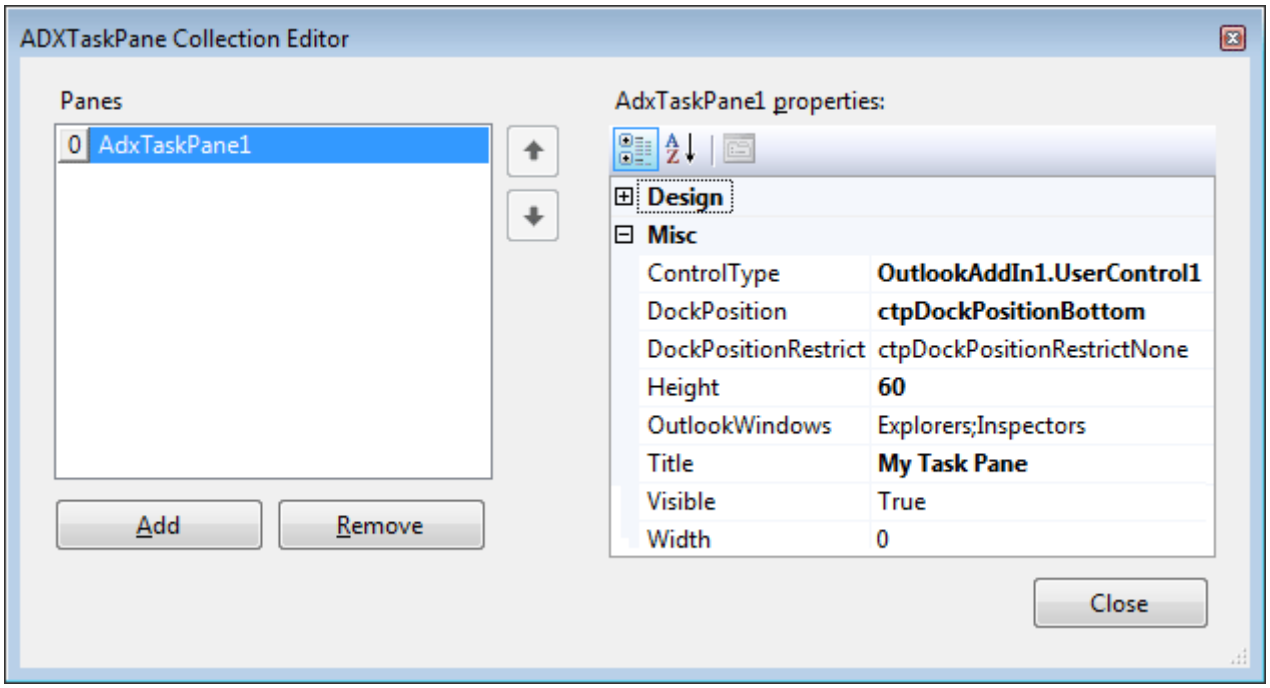
Outlook Add-ins – Template Characters in FolderName

Notwithstanding the fact that the default value of the FolderName property is '*' (asterisk), which means "every folder", the current version doesn't support template characters in the FolderName(s) property value. Moreover, this is the only use of the asterisk recognizable in the current version.

Office Custom Task Panes

To add a new task pane, you add a UserControl to your project and populate it with controls. Then you add an item to the TaskPanes collection of the add-in module and specify its properties:

- Caption – the caption of your task pane (required!);
- Height, Width – the height and width of your task pane (applies to horizontal and vertical task panes, correspondingly);
- DockPosition – you can dock your task pane to the left, top, right, or bottom edges of the host application window;
- ControlProgID – the UserControl just added.



In Add-in Express you work with the task pane component and task pane instances. The TaskPanes collection of the add-in module contains task pane components. When you set, say, the height or dock position of the component, these properties apply to every task pane instance that the host application shows. To modify a property of a task pane instance, you should get the instance itself. This can be done through the Item property of the component (in C#, this property is the indexer for the ADXTaskPane class). The property accepts the Outlook Explorer or Inspector window object that displays the task pane as a parameter. For instance, the method below finds the currently active instance of the task pane in Outlook 2007 and refreshes it. For the task pane to be refreshed in a consistent manner, this method should be called in appropriate event handlers.

```
Private Sub RefreshTaskPane( _
    ByVal TaskPaneInstance As _
        AddinExpress.VSTO.ADXTaskPane.ADXCustomTaskPaneInstance)
    If TaskPaneInstance IsNot Nothing Then
        Dim uc As UserControl1 = TaskPaneInstance.Control
        If uc IsNot Nothing _
            And TaskPaneInstance.Window IsNot Nothing _
            Then uc.InfoString = GetSubject(TaskPaneInstance.Window)
```

```
    End If
End Sub
```

The InfoString property just gets or sets the text of the Label located on the UserControl1. The GetSubject method is shown below.

```
Private Function GetSubject(ByVal ExplorerOrInspector As Object) As String
    Dim mailItem As Outlook.MailItem = Nothing
    Dim selection As Outlook.Selection = Nothing

    If TypeOf ExplorerOrInspector Is Outlook.Explorer Then
        Try
            selection = CType(ExplorerOrInspector, Outlook.Explorer).Selection
        Catch
        End Try
        If selection IsNot Nothing Then
            If selection.Count > 0 Then mailItem = selection.Item(1)
            Marshal.ReleaseComObject(selection)
        End If
    ElseIf TypeOf ExplorerOrInspector Is Outlook.Inspector Then
        mailItem = CType(ExplorerOrInspector, Outlook.Inspector).CurrentItem
    End If

    If mailItem Is Nothing Then Return ""

    Dim subject As String = "The subject is:" + "'" + mailItem.Subject + "'"
    If mailItem IsNot Nothing Then Marshal.ReleaseComObject(mailItem)
    Return subject
End Function
```

The code of the GetSubject method emphasizes the following:

- The ExplorerOrInspector parameter was originally obtained through parameters of Add-in Express event handlers. That is why we do not release it (see Releasing COM objects).
- The `selection` and `mailItem` variables were obtained "manually" so they must be released.
- All Outlook versions fire an exception when you try to obtain the Selection object in some situations.

Below is another sample that demonstrates how the same things can be done in Excel.

```
Private Sub RefreshTaskPane()
    Dim Window As Excel.Window = Me.ExcelApp.ActiveWindow
    If Window IsNot Nothing Then
        RefreshTaskPane(AdxTaskPane1.Item(Window))
        Marshal.ReleaseComObject(Window)
    End If
```

```
End Sub

Private Sub RefreshTaskPane( _
    ByVal TaskPaneInstance As  _
        AddinExpress.VSTO.ADXTaskPane.ADXCustomTaskPaneInstance)
    If TaskPaneInstance IsNot Nothing Then
        Dim uc As UserControl1 = TaskPaneInstance.Control
        If uc IsNot Nothing And _
            TaskPaneInstance.Window IsNot Nothing _
            Then
            Dim ActiveCell As Excel.Range = _
                CType(TaskPaneInstance.Window, Excel.Window).ActiveCell
            If ActiveCell IsNot Nothing Then
                'relative address
                Dim Address As String = _
                    ActiveCell.AddressLocal(False, False)
                Marshal.ReleaseComObject(ActiveCell)
                uc.InfoString = "The current cell is " + Address
            End If
        End If
    End If
End Sub
```

The InfoString property mentioned above just updates the text of the label located on the UserControl. Please pay attention to releasing COM objects in this code. See also Releasing COM objects.

VSTO solution deployment

To understand the deployment of VSTO projects, use the MSDN site. Below there are some of the links that could be of interest for you:

1. Deploying Office Solutions: http://msdn2.microsoft.com/en-us/library/hesc2788.aspx

2. Deploying Visual Studio 2005 Tools for Office Solutions Using Windows Installer (Part 1 of 2): http://msdn.microsoft.com/office/default.aspx?pull=/library/en-us/odc_vsto2005_ta/html/OfficeVSTOWindowsInstallerOverview.asp

Notes

1. Add-in Express 2008 for VSTO adds a single assembly to your setup project. The assembly is AddinExpress.VSTO.dll. In your setup project, you should add it to the Application Folder.

2. The links were correct at the moment of writing.

Releasing COM objects

The list of rules is very short:

- You **must never** release COM objects obtained through the parameters of events provided by Add-in Express.
- You **must always** release COM objects retrieved by you ("manually") from any COM object.

Note, just set a variable of say, Outlook.MailItem type to null (Nothing in VB) has nothing to do with releasing its underlying COM object. To release it, you call the Marshal.ReleaseComObject method (System.Runtime.InteropServices namespace) and pass the variable as a parameter.

An extensive review of typical problems related to releasing COM objects in Office add-ins is given in an article published on the Add-in Express technical blog – On Releasing COM objects in .NET.

Sharing Ribbon Controls Across Multiple Add-ins

First off, you assign the same string value to the Namespace property (see properties of the add-in module) for every add-in that will share your Ribbon controls. This makes Add-in Express to add two xmlns attributes to the customUI tag in the resulting Xml markup:

- xmlns:default="%ProgId of your add-in, see the ProgID attribute of the AddinModule class%",
- xmlns:shared="%the value of the AddinModule.Namespace property%".

Originally, all Ribbon controls are located in the default namespace (id="%Ribbon control's id%" or idQ="default:%Ribbon control's id%") and you have full control over them via the callbacks provided by Add-in Express. When you specify the Namespace property, Add-in Express changes the markup to use idQ's instead of id's.

Then, in all add-ins that should share a Ribbon control, for the control with the same Id (you can change the Id's to match), you set the Shared property to True. For the Ribbon control whose Shared property is True, Add-in Express changes its idQ to use the shared namespace (idQ="shared:%Ribbon control's id%") instead of the default one. Also, for such Ribbon controls, Add-in Express cuts out all callbacks and replaces them with "static" versions of the attributes. Say, getVisible="getVisible_CallBack" will be replaced with visible="%value%".

The shareable Ribbon controls are the following Ribbon container controls:

- Ribbon Tab - ADXRibbonTab
- Ribbon Box - ADXRibbonBox
- Ribbon Group - ADXRibbonGroup

- Ribbon Button Group - ADXRibbonButtonGroup

When referring to a shared Ribbon control in the BeforeId and AfterId properties of another Ribbon control, you use the shared controls' idQ: %namespace abbreviation% + ":" + %control id%. The abbreviations of these namespaces are "default" and "shared" string values.

The resulting XML markup may look like this:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
          xmlns:default="MyOutlookAddin1.AddinModule"
          xmlns:shared="MyNameSpace" [callbacks omitted]>
 <ribbon>
  <tabs>
   <tab idQ=" shared:adxRibbonTab1" visible="true" label="My Tab">
    <group idQ="default:adxRibbonGroup1" [callbacks omitted]>
     <button idQ="default:adxRibbonButton1" [callbacks omitted]/>
    </group>
   </tab>
  </tabs>
 </ribbon>
</customUI>
```

In the XML-code above, the add-in creates a shared tab, containing a private group, containing a button (private again).

Deploying Office Add-ins

- Make sure Vista, Windows XP, and Office have all available updates installed: Microsoft eventually closes their slips and blunders with service packs and other updates. Keep an eye on Visual Studio updates, too.
- If a non-admin user will install your add-in, use [AppDataFolder] as a default location. Make sure the user is instructed to run .msi. On Vista, if the user runs setup.exe, a non-admin will get the elevation dialog and this can end with installing the add-in to the admin profile. In this case, the add-in will not be available for the standard user.

Finally

If your questions are not answered here, please see the HOWTOs section on www.add-in-express.com. From time to time, we update these pages ;-).