Add-in Express™ Regions

for Microsoft® Outlook and VSTO

DEVELOPER'S GUIDE

Add-in Express™

www.add-in-express.com

Add-in Express Regions for Microsoft® Outlook and VSTO

Developer's Guide

Revised on 28-Apr-17

Copyright © Add-in Express Ltd. All rights reserved.

Add-in Express, ADX Extensions, ADX Toolbar Controls, Afalina, AfalinaSoft and Afalina Software are trademarks or registered trademarks of Add-in Express Ltd. in the United States and/or other countries. Microsoft, Outlook, and the Office logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Borland and the Delphi logo are trademarks or registered trademarks of Borland Corporation in the United States and/or other countries.

THIS SOFTWARE IS PROVIDED "AS IS" AND ADD-IN EXPRESS LTD. MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, ADD-IN EXPRESS LTD. MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

Table of Contents

Introduction

Add-in Express Regions for Microsoft® Outlook and VSTO supports creating custom panes in VSTO-based Outlook add-in projects in Visual Studio.

Technical Support

Add-in Express Regions for Outlook and VSTO is developed and supported by the Add-in Express Team, a branch of Add-in Express Ltd. The Add-in Express web site at www.add-in-express.com provides a wealth of information and software downloads for Add-in Express developers, including:

- Our technical blog provides the most recent information as well as How To and Video How To samples.
- The HOWTOs section contains sample projects answering most common "how to" questions.

For technical support through the Internet use our forums or e-mail us at support@add-in-express.com. We are actively participating in these forums.

If you are a subscriber of our Premium Support Service and need help immediately, 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 installation. First off, you have to specify the development environments in which you are going to use Add-in Express. Second, you need to activate the product.

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 setup program installs the following folders on your PC:

- *Bin* – binary files
- *Docs* – documentation including class reference
- *Images* – icons
- *Demo Projects* – a sample project (available in C# and VB.NET)
- *Sources* – source code (if the package includes it)

The setup program installs the following text files on your PC:

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

Redistributables

The only redistributable file is located in *{Add-in Express}\Bin*. Its name is *AddinExpress.Outlook.Regions.dll*.

Solving Installation Problems

Make sure you are an administrator on the PC. On Vista, Windows 7 and Windows 2008 Server, set UAC to its default level. In *Control Panel* | *System* | *Advanced* | *Performance* | *Settings* | *Data Execution Prevention*, set the *... for essential Windows programs and services only* flag. Remove the following registry key, if it exists:

```
HKEY_CURRENT_USER\Software\Add-in Express\{product identifier} {version} {package}
```

Run *setup.exe*, not the **.MSI*. Finally, use the Automatic activation option in the installer windows.

Other Add-in Express Products

Add-in Express provides a number of products for developers on its web site.

- Add-in Express for Microsoft Office and .NET

It simplifies the creation and deployment of version-neutral managed COM add-ins, smart tags, Excel Automation add-ins, XLL add-ins and RTD servers in Visual Studio 2010-2017 for Office 2000-2016 (without VSTO). See http://www.add-in-express.com/add-in-net/.

- Add-in Express for Microsoft Office and CodeGear VCL

It allows creating fast version-neutral native-code COM add-ins, smart tags, Excel automation add-ins, and RTD servers in Delphi. See http://www.add-in-express.com/add-in-delphi/.

- Add-in Express for Internet Explorer and .NET

It allows developing add-ons for IE 6, 7, 8, 9, 10 and 11 in .NET. Custom toolbars, sidebars and BHOs are already on board. See http://www.add-in-express.com/programming-internet-explorer/.

- Security Manager for Microsoft Outlook

This is a product designed for Outlook solution developers. It allows controlling the Outlook e-mail security guard by turning it off and on in order to suppress unwanted Outlook security warnings. See http://www.add-in-express.com/outlook-security/.

Getting Started

Here we guide you through the following steps of developing Add-in Express projects:

- Adding an Advanced Outlook Form Region Class to the project
- Configuring the Advanced Form Region
- Deploying your project to a target PC

Terms and Definitions

In Add-in Express terms, an advanced Outlook region is a sub-pane, or a dock, of Outlook windows that may host native .NET forms. There are two types of the advanced regions – Outlook view regions (sub-panes of the Outlook Explorer window) and Outlook form regions (sub-panes of the Outlook Inspector window).

An advanced form region is a descendant of `System.Windows.Forms.Form`, which is extended by the Add-in Express team to comply with Outlook windowing rules.

*You **never** create an instance of a form region in the way you create an instance of a Windows form. Instead, there is a manager class providing a collection, which you populate at design-time or run-time or both. Each item of the collection binds an advanced form region to the visualization and context settings (such as Outlook item types for which your region will be shown). And it is the forms manager that creates instances of the form region automatically or at your request.*

The terms above are translated into class names as below; the class names are located in the `AddinExpress.OL` namespace provided by *AddinExpress.Outlook.Regions.dll* (redistributable):

- Advanced Form Region – `ADXOlForm`
- Manager – `ADXOlFormsManager`
- Collection – `ADXOlFormsCollection`
- Collection item – `ADXOlFormsCollectionItem`

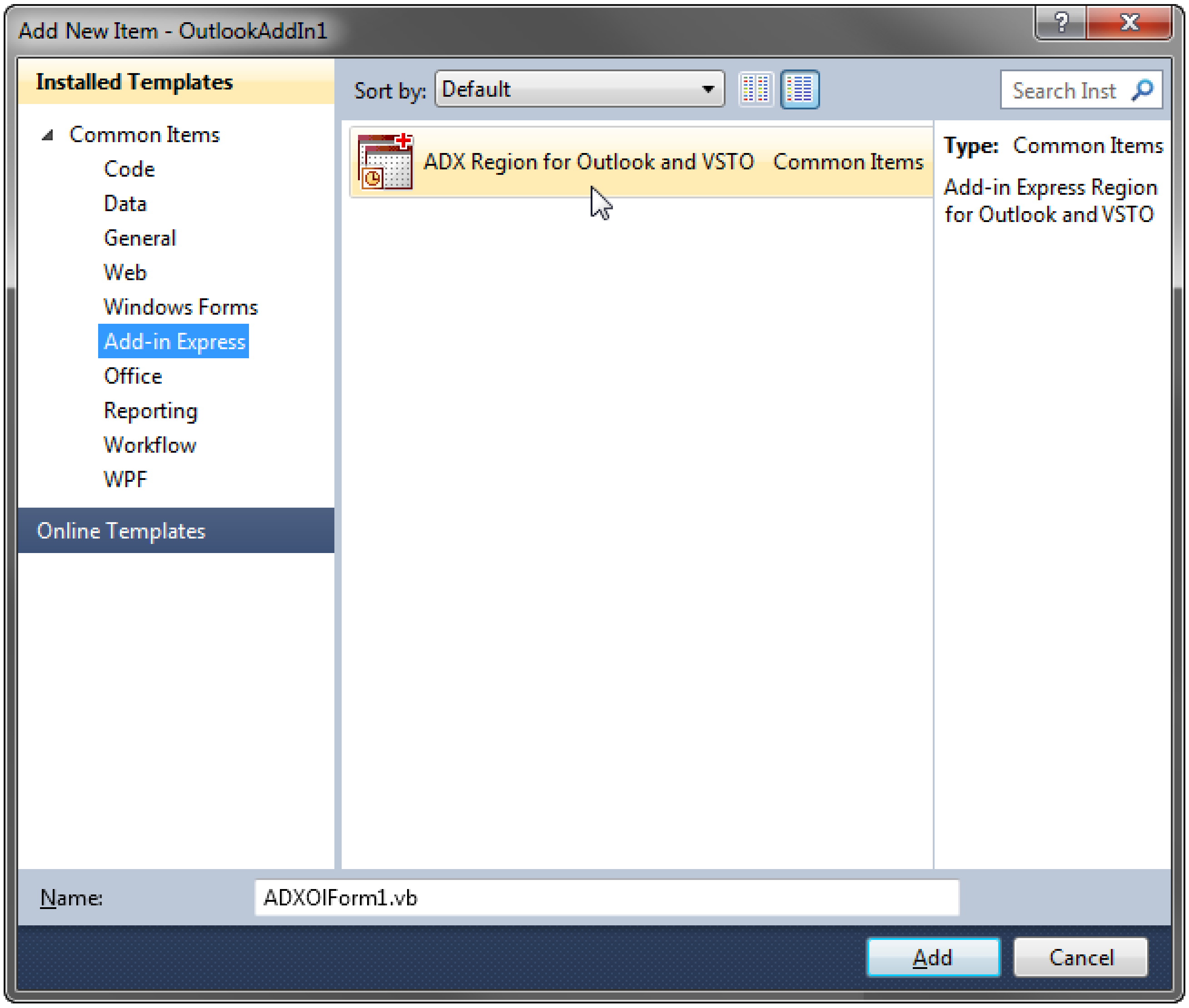
The `Visible` property of a form region instance is `true` when the instance is embedded into a window region (as specified by the visualization settings) regardless of the actual visibility of the instance.

The `Active` property of the form region instance is `true` when the instance is shown on top of all other instances in the same region.

You First Advanced Outlook Region

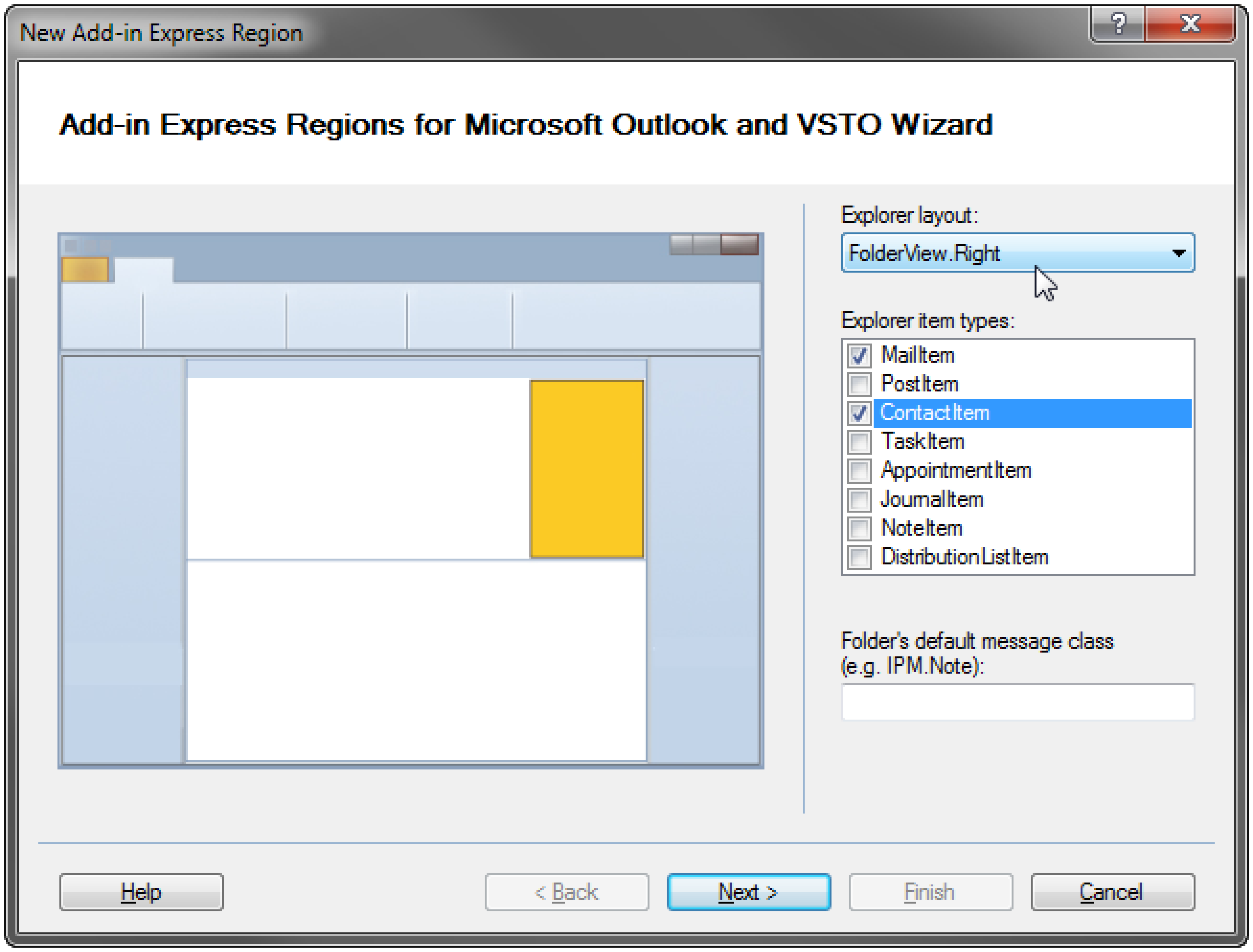
Adding an Advanced Outlook Form Region Class

So, you start with adding an advanced form region class to your Outlook add-in project. Open the *Add New Item* dialog of your VSTO project and select the *Add-in Express* node.



Clicking *Add* starts a three-step wizard. Let's see how to use it.

On **step #1**, you choose a value from the *Explorer Layout* dropdown list to specify how your form will be shown in the Outlook Explorer window. The values available in the dropdown list are described in Advanced Outlook Form and View Regions. Leaving the *Explorer Layout* set to `Unknown` (the default value) means that your form will not be shown in the Outlook Explorer.



Add-in Express Regions for Microsoft Outlook and VSTO Wizard

Explorer layout: FolderView.Right

Explorer item types:
- [✓] MailItem
- [ ] PostItem
- [✓] ContactItem
- [ ] TaskItem
- [ ] AppointmentItem
- [ ] JournalItem
- [ ] NoteItem
- [ ] DistributionListItem

Folder's default message class (e.g. IPM.Note):

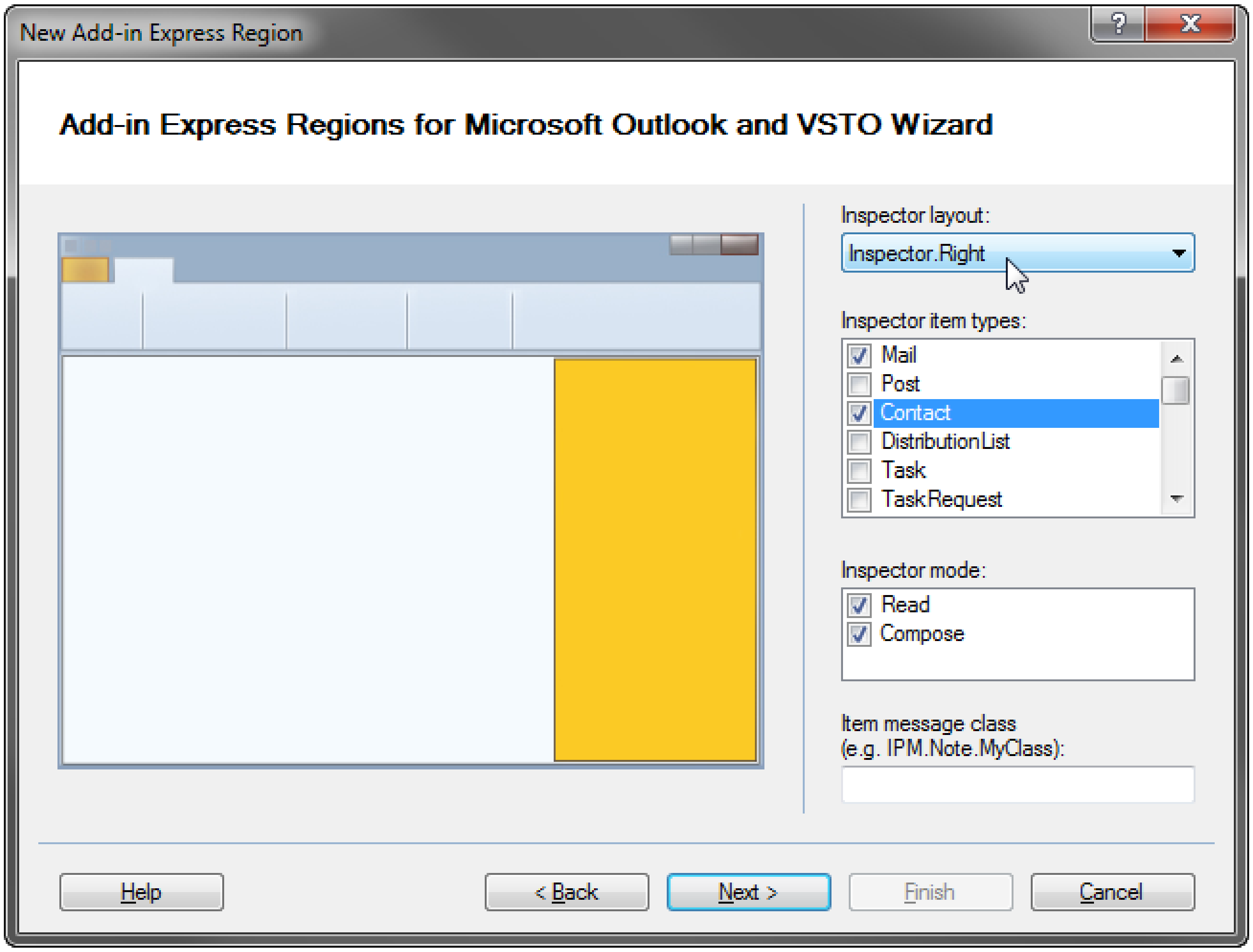
Specifying the layout is not enough, however. To have instances of the form displayed, you also need to set at least one of the three context-sensitivity properties – `ExplorerItemTypes`, `ExplorerMessageClass` or `FolderNames`. You use these properties to specify folders for which your form (or rather an instance of your form) will be shown. The properties refer to the properties of the `Outlook.Folder` (`Outlook.MAPIFolder`) class that specify the default item type and default message class for items in the folder.

At run-time, different [sets of] folders may correspond to the values chosen in the *Explorer Item Types* and *Explorer Message Class*. In this case, your form will be shown for all of the folders specified by these settings. This happens because all context-sensitive properties are calculated using Boolean `OR`, not `AND`.

Step #1 of the wizard provides the *Explorer Item Types* and *Explorer Message Class* controls (you set folders on step #3). In this sample project, you choose *RightSubpane* in the *Explorer Layout* dropdown and check *MailItem* and *ContactItem* in the *Explorer Item Types* list box. Now click *Next*.

On **step #2**, you specify how the form is shown on Outlook Inspector windows in the very similar fashion.

That is, you choose a value in the *Inspector Layout* dropdown list (see Advanced Outlook Form and View Regions for reference). Leaving the *Inspector Layout* set to its default value means your form will not be shown in the Outlook Inspector.


New Add-in Express Region
Add-in Express Regions for Microsoft Outlook and VSTO Wizard
Help
< Back
Next >
Finish
Cancel

Inspector layout: Inspector.Right

Inspector item types:
- [✓] Mail
- [ ] Post
- [✓] Contact
- [ ] DistributionList
- [ ] Task
- [ ] TaskRequest

Inspector mode:
- [✓] Read
- [✓] Compose

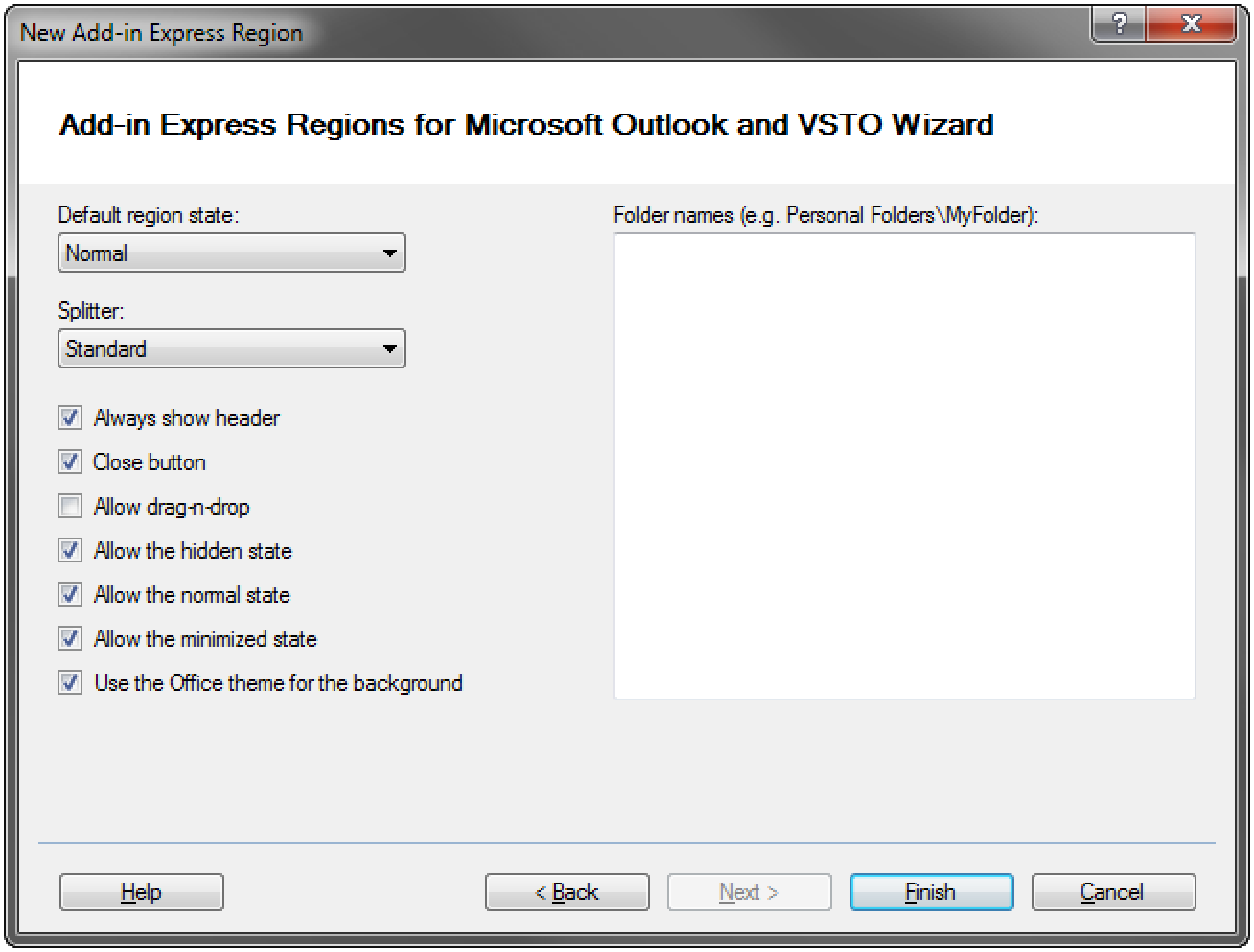
Item message class (e.g. IPM.Note.MyClass):

The *Inspector Item Types, Inspector Message Class* and *Inspector Mode* controls refer to the properties of the Outlook item shown in an Outlook Inspector; the properties define the message class and item type of the item. Please check Context-Sensitivity of Your Outlook Form.

Choosing any value in the Inspector Item Type Note guarantees that your form will be displayed for all inspectors showing items of the specified type(s). Similarly, specifying any particular message class guarantees that your form will be displayed for all inspectors showing items of the specified message class. This is due to the fact that all context-sensitive properties are calculated using Boolean *OR*, not *AND*.

In this sample, you choose the *Inspector.Right* layout and *Mail* and *Contact* item types. Click *Next*.

On **step #3**, you set up the features listed below:


New Add-in Express Region

Add-in Express Regions for Microsoft Outlook and VSTO Wizard

Default region state: Normal

Splitter: Standard

- [✓] Always show header
- [✓] Close button
- [ ] Allow drag-n-drop
- [✓] Allow the hidden state
- [✓] Allow the normal state
- [✓] Allow the minimized state
- [✓] Use the Office theme for the background

Folder names (e.g. Personal Folders\MyFolder):

- *Default region state* - You choose between normal (default), minimized and hidden states; this property works only if the Add-in Express Regions were not run. You can reproduce this situation by choosing *Reset Regions* in the context menu of the form's designer surface.
- *Splitter* - You choose between two values: *Standard* – the end-user is able to resize the form and the developer is not; another value is *None* – the end-user cannot resize the form and the developer is able to resize it programmatically.
- *Always show header* - This option helps to handle only one scenario: if the default value is chosen, the header for your form region **will not be shown** if the form is the only form shown in the given layout. In other words, if this option is set, the header for your form region **will be shown** even if the form is the only form in the given layout.
- *Close button* - This option determines if the *Close* button is shown in the header (naturally, if the header is shown).
- *Allow drag-n-drop* - If this is set to false, the end-user cannot move the form to another location. If it is set to true, the end-user can drag the form to a location specified in the *ExplorerAllowedDropRegions* and *InspectorAllowedDropRegions* properties of the *ADXOlFormsCollectionItem*.

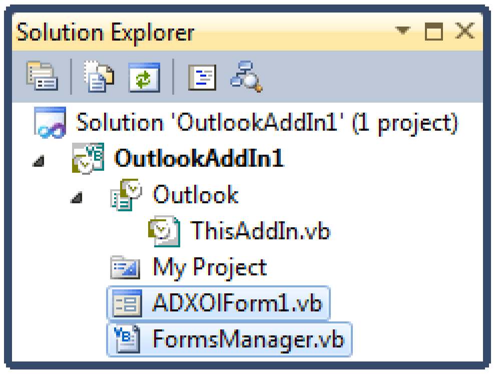
- *Allow the hidden state* - If set to false, the user cannot hide the form by clicking on the "dotted" mini-button or by double-clicking anywhere else on the splitter; nevertheless, you can do this programmatically; see also Region States and UI-related Properties and Events.
- *Allow the normal state* - If set to false, the form will be shown in the minimized state; the user will be able to expand it as described in Region States and UI-related Properties and Events.
- *Allow the minimized state* - If set to true, the form can be shown reduced to the size of the form's caption (the minimized state); see also Region States and UI-related Properties and Events.
- *Use the Office theme for the background* - if set to true, the background color of your form is set to match the current theme in Office 2007-2016.
- *Folder names* - If you chose to show your form in any given Explorer layout, specifying the full path to a folder(s) guarantees that your form region will be shown when you navigate to the folder(s) in the Outlook UI. Similarly, if you design your form to show up in any Inspector layout, specifying a folder(s) guarantees that the form region will be shown for all Outlook items you open in that folder(s). An example of the folder path is "\\Personal Folders\Inbox". See also Conte xt-Sensitivity of Your Outlook Form.

Set the *Always show header* and *Close button* options and click *Finish*.

The wizard allows populating the most often used properties. The detailed information on how to use all available properties is provided in The UI Mechanics.

Checking the Project

The wizard adds the following items to your project:



- *AddinExpress.Outlook.Regions.dll* to the *Reference* section,
- the advanced region, which is *ADXOlForm1.vb* in this case,
- *FormsManager.vb* (*FormsManager.cs*) code file.

Don't rename FormsManager.vb (FormsManager.cs)!

Also, initialization and finalization calls for the forms manager are added to *ThisAddin.vb* (*ThisAddin.cs*).

```
Public Class ThisAddIn
```

```
    Private Sub ThisAddIn_Startup() Handles Me.Startup
        ' <auto-generated>
        ' Add-in Express Regions generated code - do not modify
        Me.FormsManager = AddinExpress.OL.ADXOlFormsManager.CurrentInstance
        Me.FormsManager.Initialize(Me)
        ' </auto-generated>
    End Sub

    Private Sub ThisAddIn_Shutdown() Handles Me.Shutdown
        ' <auto-generated>
        ' Add-in Express Regions generated code - do not modify
        Me.FormsManager.Finalize(Me)
        ' </auto-generated>
    End Sub
End Class
```

The `FormsManager` mentioned in the code above is declared in *FormsManager.vb*. Let's study its code.

Configuring the Form Region

Open *FormsManager.vb* (*FormsManager.cs*); it extends ThisAddIn with the following code:

```
Imports AddinExpress.OL

Partial Public Class ThisAddIn

    Public WithEvents FormsManager As ADXOlFormsManager = Nothing

    ''' <summary>
    ''' Use this event to initialize regions and connect
    ''' to the events of ADXOlFormsManager
    ''' </summary>
    Private Sub FormsManager_OnInitialize() Handles FormsManager.OnInitialize

        '<ADXOlForm1>
        ' TODO: Use the ADXOlForm1Item properties to configure the region's
        ' location, appearance and behavior.
        ' See the "The UI Mechanics" chapter of the Add-in Express Developer's
        ' Guide for more information.

        Dim ADXOlForm1Item As ADXOlFormsCollectionItem = _
            New AddinExpress.OL.ADXOlFormsCollectionItem()
        ADXOlForm1Item.ExplorerLayout = ADXOlExplorerLayout.RightSubpane
        ADXOlForm1Item.ExplorerItemTypes = ADXOlExplorerItemTypes.olMailItem _
            Or ADXOlExplorerItemTypes.olContactItem
        ADXOlForm1Item.InspectorLayout = ADXOlInspectorLayout.RightSubpane
        ADXOlForm1Item.InspectorItemTypes = ADXOlInspectorItemTypes.olMail _
            Or ADXOlInspectorItemTypes.olContact
        ADXOlForm1Item.AlwaysShowHeader = True
        ADXOlForm1Item.CloseButton = True
        ADXOlForm1Item.UseOfficeThemeForBackground = True
        ADXOlForm1Item.FormClassName = GetType(ADXOlForm1).FullName
        Me.FormsManager.Items.Add(ADXOlForm1Item)
        '</ADXOlForm1>
    End Sub

#Region "RequestService"
    ''' <summary>
    ''' Required method for DockRight, DockLeft, DockTop and
    ''' DockBottom layout support.
    ''' </summary>
    Protected Overrides Function RequestService(ByVal serviceGuid As Guid) _
        As Object
        If serviceGuid = GetType(Office.ICustomTaskPaneConsumer).GUID Then
            Return AddinExpress.OL.CTPFactoryGettingTaskPane.Instance
        End If
        Return MyBase.RequestService(serviceGuid)
    End Function
#End Region

End Class
```

The code above consists of three parts:

- The declaration of the forms manager object, see *FormsManager*.
- The *FormsManager_OnInitialize* method, which is the event handler for the *OnInitialize* event of the forms manager. You use this method to configure advanced regions; say you can fill the *FolderName* (*FolderNames*) property of *ADXOlFomsCollectionItem* with actual values in that method. In the code above, you see how settings specified in the wizard are mapped to the properties of *ADXOlForm1Item,* which binds the name of the form region class, *ADXOlForm1,* together with the visualization and context-sensitivity settings specified, for example, in the *ExplorerItemTypes* property above; please check Context-Sensitivity of Your Outlook Form.
- The *RequestService* method, which is required if you use "*Dock**" Explorer layouts (see Outlook view regions).

Don't rename the FormsManager property and FormsManager_OnInitialize method!

Programming with Advanced Regions

In this sample project, we add a button (*Button1*) and label (*Label1*) onto the form.

Is It *Inspector* or *Explorer*?

Add an event handler for the *Load* event of the form and write the following code:

```
Public Class ADXOlForm1
...
    Private Sub ADXOlForm1_Load(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) Handles MyBase.Load
        If Me.InspectorObj Is Nothing Then Me.Button1.Visible = False
        If Me.ExplorerObj Is Nothing Then Me.Label1.Visible = False
    End Sub
```

The code above demonstrates the base principle:

- if *ADXOlForm.InspectorObj* returns a value other than *Nothing* (*null* in C#), then the form is shown in an Outlook Inspector window;
- similarly, if *ADXOlForm.ExplorerObj* returns a value other than *Nothing* (*null* in C#), then the form is shown in an Outlook Explorer window.

InspectorObj and *ExplorerObj* properties return COM objects that you must never release in your code because their state is crucial for the functionality of your advanced region. These properties will be released automatically when your form is removed from its region. This may occur several times during the lifetime of a given form instance because the manager may remove your form from a given region and then embed the form to the same region in order to comply with the Outlook windowing.

Accessing *ThisAddIn* from the Form

We suggest adding a Shared (static in C#) member in *ThisAddIn*. In this sample, we use the code below:

```
Public Class ThisAddIn

    Public Shared AddinInstance As ThisAddIn

    Private Sub ThisAddIn_Startup() Handles Me.Startup

        ' <auto-generated>
        ' Add-in Express Regions generated code - do not modify
        Me.FormsManager = AddinExpress.OL.ADXOlFormsManager.CurrentInstance
        Me.FormsManager.Initialize(Me)
        ' </auto-generated>

        ThisAddIn.AddinInstance = Me
    End Sub
...
End Class
```

The use of *AddinInstance* is demonstrated below.

Accessing the Outlook Object Model

Add the following to the *Click* event handler of the button:

```
Imports System.Windows.Forms

Public Class ADXOlForm1
...
    Private Sub Button1_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) Handles Button1.Click
        MessageBox.Show(ThisAddIn.AddinInstance. _
                        GetSubject(TryCast(Me.InspectorObj, Outlook.Inspector)))
    End Sub
```

And in *ThisAddIn*, add methods that deal with the Outlook object model:

```
Imports System.Runtime.InteropServices

Public Class ThisAddIn
...
    Public Function GetSubject(ByVal anInspector As Outlook.Inspector) As String

        Dim result As String = ""
        If anInspector Is Nothing Then
            Throw New Exception("The parameter must be an Outlook.Inspector.")
            result = "undefined"
        Else
            Dim item As Object = anInspector.CurrentItem
            result = GetSubject(item)
            Me.ReleaseComObject(item)
            item = Nothing
        End If
        ' The caller is responsible for releasing anInspector (a COM object)
        Return result
    End Function
    Private Function GetSubject(ByVal OutlookItem As Object) As String
        Dim result As String = ""
        If Not TypeOf OutlookItem Is Outlook.MailItem Then
            result = "This is not an e-mail."
        Else
            Dim mail As Outlook.MailItem = TryCast(OutlookItem, Outlook.MailItem)
            Try
                result = mail.Subject
                If result = "" Then result = "(no subject)"
            Catch ex As Exception
                result = "Exception: " + ex.Message
            End Try
        End If
        ' The caller is responsible for releasing OutlookItem (a COM object)
        Return result
    End Function
```

Note that the COM object standing for the Outlook Inspector is not released in the code above; that's because that COM object is controlled by the forms manager and it must not be released in your code (see below).

Releasing COM Objects - a must in Outlook Add-ins

A comprehensive review of typical problems (and solutions) related to releasing COM objects in Office add-ins is given in an article published on the Add-in Express technical blog – When to release COM objects in Office add-ins?.

Below are some useful tips.

Pay attention that *Inspector.CurrentItem* returns an *Object* (with an underlying RCW pointing to the COM object) and passing the *Object* to another method as well as casting it to *Outlook.MailItem* doesn't produce new COM objects (doesn't increase the reference counter).

Passing *Nothing* (*null* in C#) to *ReleaseComObject* produces an exception. Passing a non-COM object to *ReleaseComObject* produces an exception, too. These situations are handled by the following method that you add to *ThisAddIn*:

```
Public Class ThisAddIn
...
    Private Sub ReleaseComObject(ByVal obj As Object)
        If obj IsNot Nothing Then
            If Marshal.IsComObject(obj) Then
                Marshal.ReleaseComObject(obj)
            End If
        End If
    End Sub
```

You can also get the "COM object that has been separated from its underlying RCW cannot be used" exception if you pass a variable to *ReleaseComObject* and then use the variable in your code. We recommend assigning *Nothing* (*null* in C#) to all released variables.

The functionality provided by Add-in Express Regions for Outlook and VSTO is based on the combined use of the Outlook windowing and features provided by the Outlook object model. These two areas are essentially different and keeping objects from these areas in sync requires significant efforts. Some difficulties are impossible to overcome. One of them is born by the need to control the state of COM objects provided by in properties and events of Add-in Express Regions: releasing any of such objects may lead to a run-time exception.

Do not release any COM objects you acquire in the properties and events of ADXOlFormsManager *and* ADXOlForm*. On the contrary, release every COM object you create in your code.*

Dealing with Outlook Events

Let's handle the *SelectionChange* event provided by the *Outlook.Explorer* object.

```
Public Class ThisAddIn

    Private WithEvents theExplorer As Outlook.Explorer
...
    Private Sub ThisAddIn_Startup() Handles Me.Startup
...
        theExplorer = Me.Application.ActiveExplorer()
...
    End Sub

    Private Sub DoExplorerSelectionChange() Handles theExplorer.SelectionChange
        Dim result As String = GetSubject(theExplorer)
        'TODO: show the subject on the form
    End Sub

    Public Function GetSubject(anExplorer As Outlook.Explorer) As String
        Dim selection As Outlook.Selection = Nothing
        Try
            selection = anExplorer.Selection
        Catch
            ' Skip an exception that occurs if the current
            ' Outlook folder is one of the following types:
            ' - a top-level folder e.g Personal Folders,
            ' - RSS Feeds, etc.
        End Try

        Dim result As String = ""

        If selection Is Nothing Then
            result = "This is a top-level folder. It contains no items."
        Else
            Try
                If selection.Count = 0 Then
                    result = "The current folder contains no items."
                Else
                    Dim item As Object = selection.Item(1)
                    result = GetSubject(item)
                    Me.ReleaseComObject(item)
                    item = Nothing
                End If
            Finally
                Me.ReleaseComObject(selection)
                selection = Nothing
            End Try
        End If
        Return result
    End Function

    Private Sub DoExplorerClose() Handles theExplorer.Close
        Me.ReleaseComObject(theExplorer)
        theExplorer = Nothing
    End Sub
...
End Class
```

The *DoExplorerClose* method demonstrates how you disconnect from events provided by a COM object.

Accessing the Form from *ThisAddIn*

You get a collection item and call the *GetCurrentForm* method. That method provides a parameter the possible values of which address to the *Visible* and *Active* properties of *ADXOlForm*.

The Visible *property of a form region 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 region instance is* true *when the instance is shown on top of all other instances in the same region.*

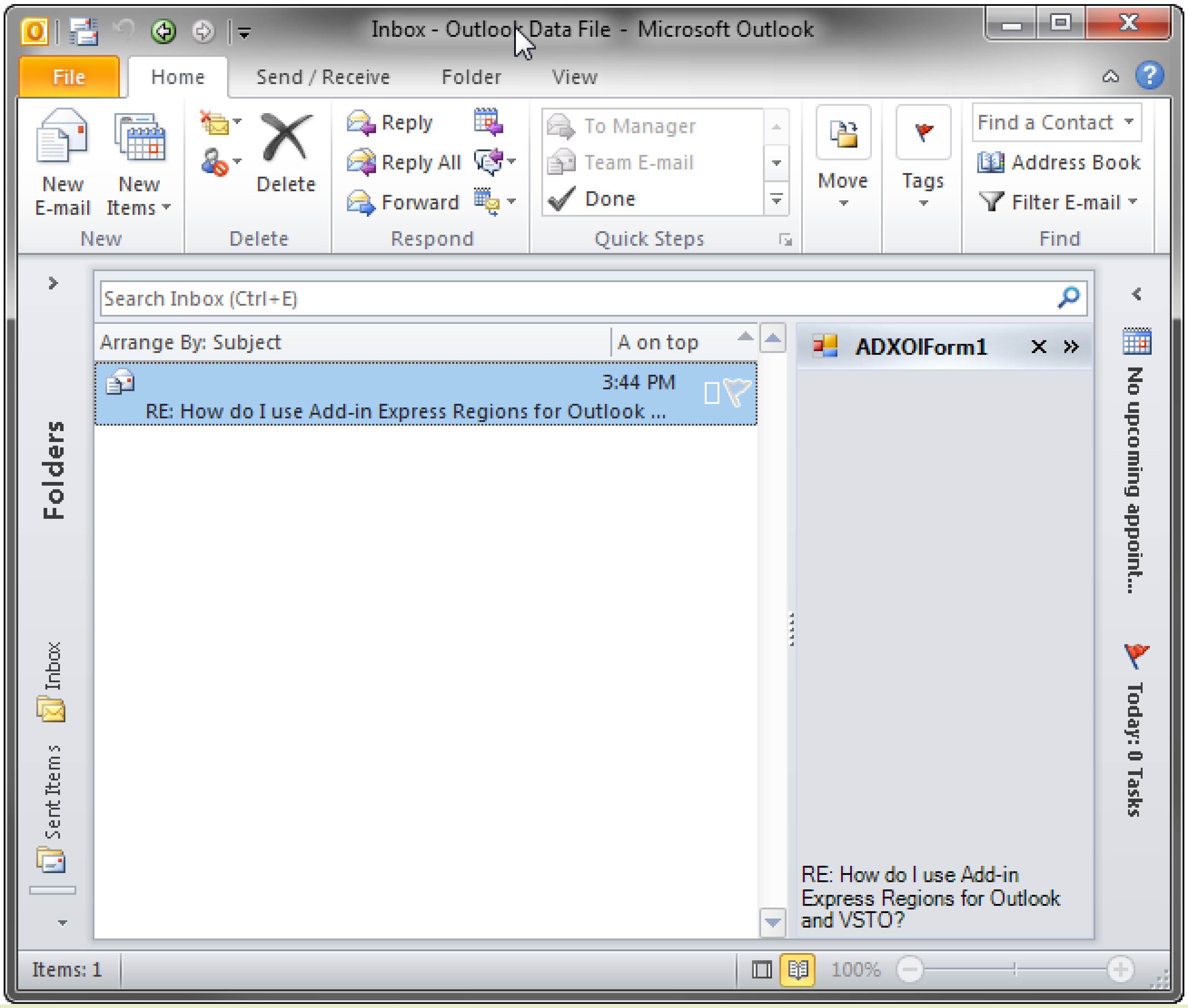
The code is shown below:

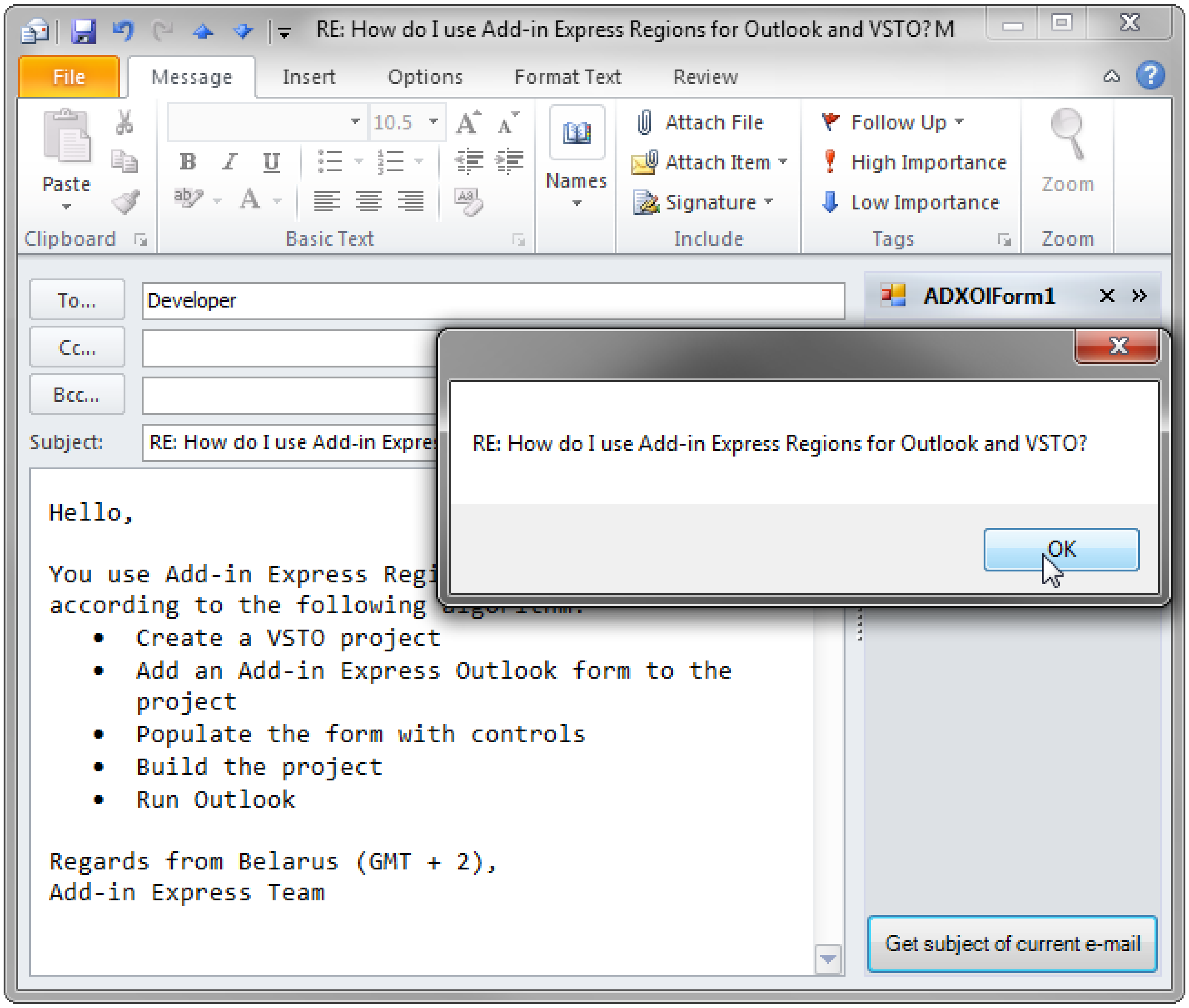
```
    Public Sub OnAction(control As IRibbonControl)
        Dim result As String = GetSubject()
        Dim form As ADXOlForm1 = _
            TryCast(FormsManager.Items(0).GetCurrentForm( _
                    AddinExpress.OL.EmbeddedFormStates.Visible), ADXOlForm1)
        If form IsNot Nothing Then form.SelectedItemSubject = result
    End Sub
```

The code refers to the `SelectedItemSubject` property defined in `ADXOlForm1`; the code of the property is omitted for brevity – it just assigns the string to `Label1.Text`.

Running the Add-in

Press {F5} to get a result similar to what is shown on the screenshots below:





Developer

RE: How do I use Add-in Expre

RE: How do I use Add-in Express Regions for Outlook and VSTO?

```
Hello,

You use Add-in Express Regi
according to the following algorithm:
    •  Create a VSTO project
    •  Add an Add-in Express Outlook form to the
       project
    •  Populate the form with controls
    •  Build the project
    •  Run Outlook

Regards from Belarus (GMT + 2),
Add-in Express Team
```

Deploying the Region

Make sure your setup project delivers *AddinExpress.Outlook.Regions.dll* to the target PC.

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

What's Next?

Please refer yourself to Advanced Outlook Form and View Regions and The UI Mechanics below.

Advanced Outlook Form and View Regions

Outlook view regions are specified in the `ExplorerLayout` *property of the item (=* `ADXOlFormsCollectionItem`*). Outlook form regions are specified in the* `InspectorLayout` *property of the item. That is, one* `ADXOlFormsCollectionItem` *may show your form in a view and form region. Note that you must also specify the* `ExplorerItemTypes` *and/or* `InspectorItemTypes` *properties of the item; otherwise, the form (an instance of* `ADXOlForm`*) will never be shown.*

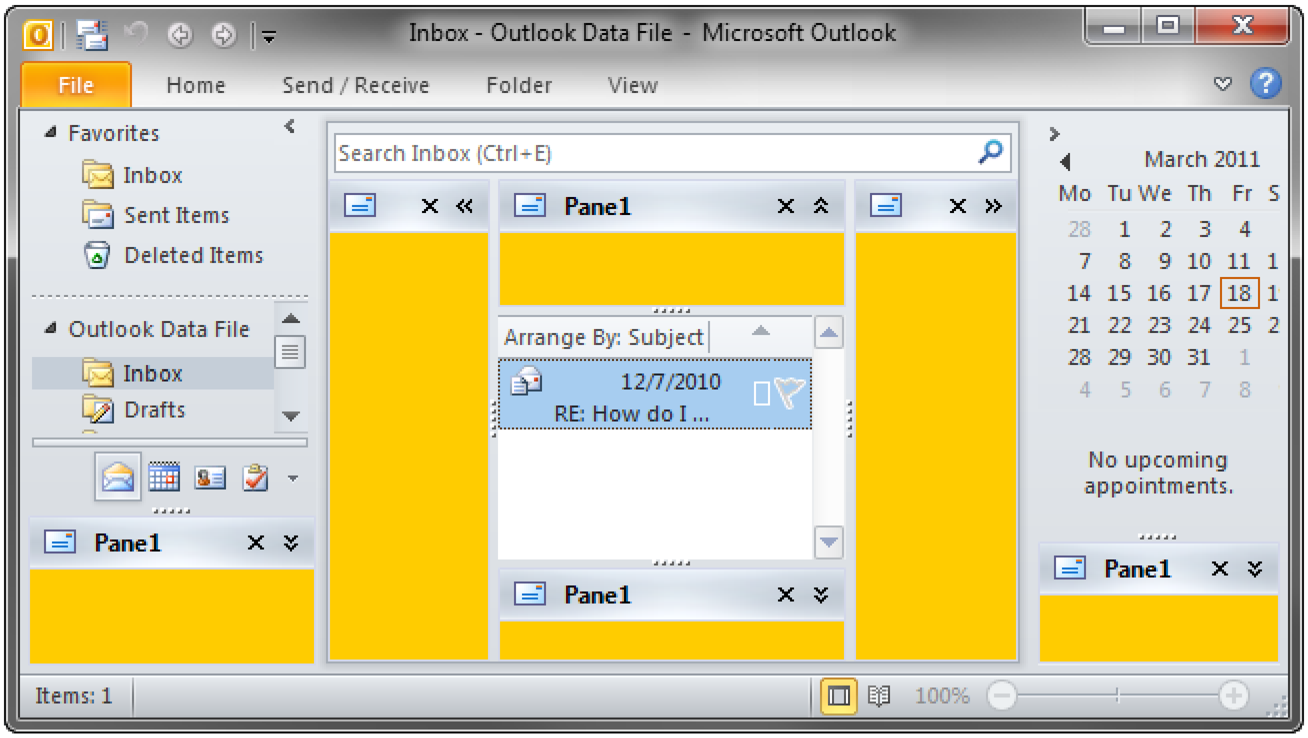
Outlook view regions

In Add-in Express terms, an advanced Outlook region is a sub-pane, or a dock, of Outlook windows that may host native .NET forms. There are two types of the advanced regions – Outlook view regions (sub-panes on the Outlook Explorer window) and Outlook form regions (sub-panes of the Outlook Inspector window).

Outlook view regions are specified in the `ExplorerLayout` property of the item (= `ADXOlFormsCollectionItem`). Outlook form regions are specified in the `InspectorLayout` property of the item. That is, one `ADXOlFormsCollectionItem` may show your form in a view and form region. 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.

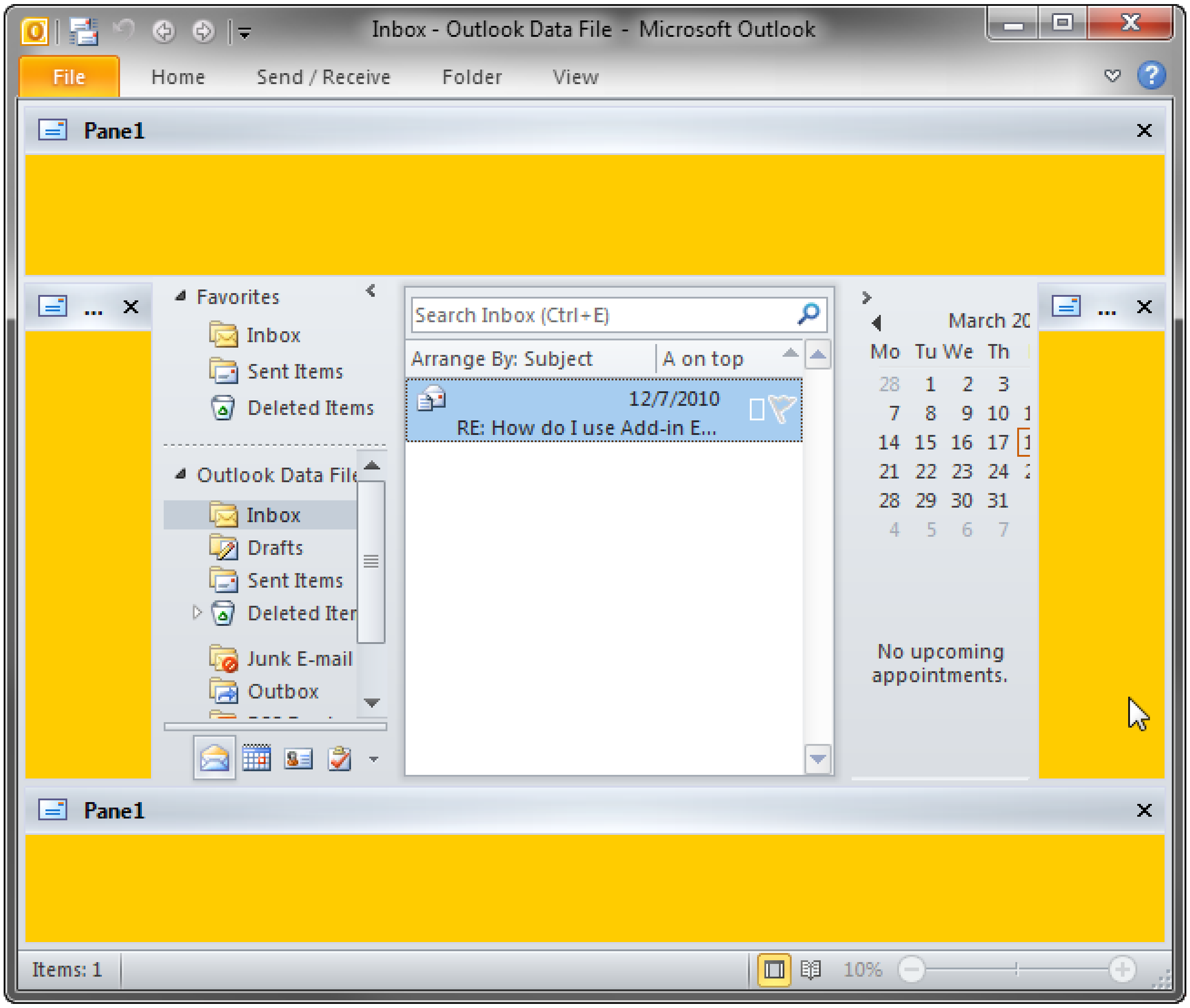
Here is the list of **Outlook view regions**:

- Four regions around the list of mails, tasks, contacts etc. The region names are `LeftSubpane`, `TopSubpane`, `RightSubpane`, `BottomSubpane` (see the screenshot below). **A restriction**: those regions are not available for Calendar folders in Outlook 2010-2016.
- One region below the Navigation Pane – `BottomNavigationPane` (see the screenshot below)
- One region below the To-Do Bar – `BottomTodoBar` (see the screenshot below). **A restriction**: this region is not available in Outlook 2013-2016.

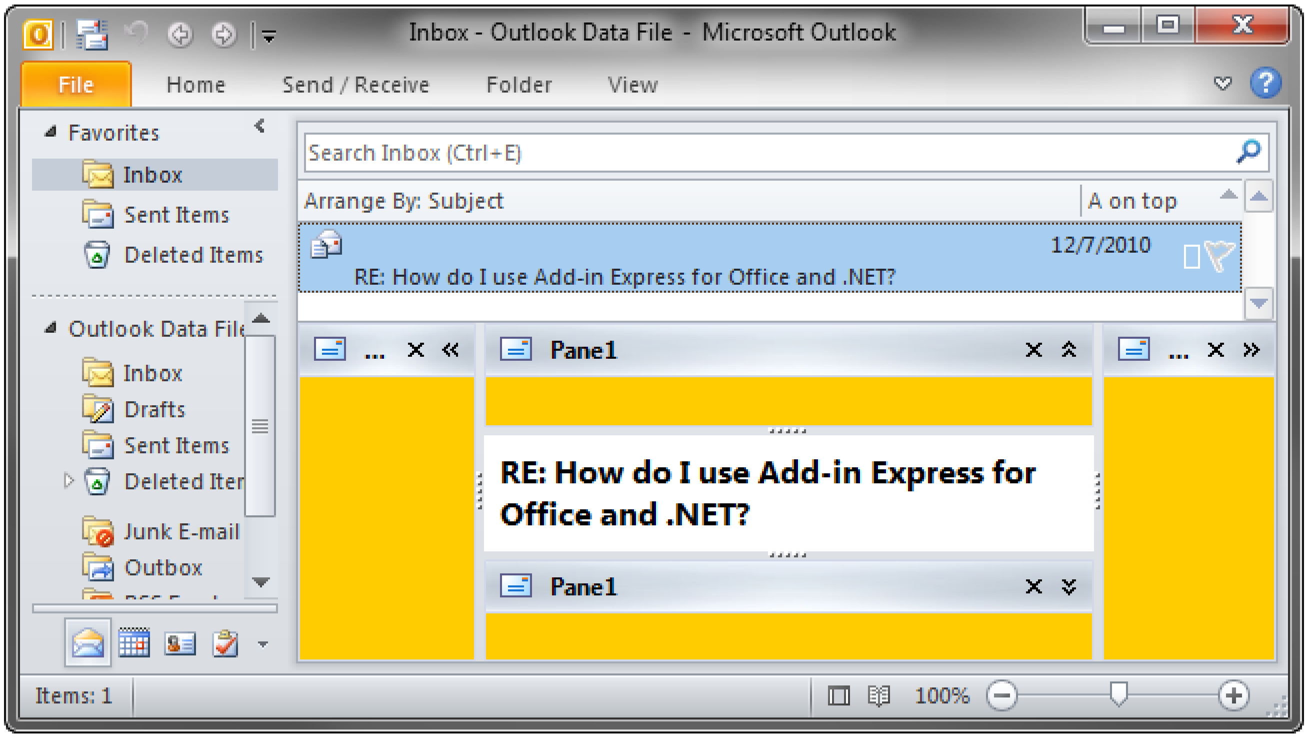


- Four regions around the Explorer window (Outlook 2007-2016) – `DockLeft`, `DockTop`, `DockRight`, `DockBottom` (see the screenshot below). The **restrictions** of these regions are:

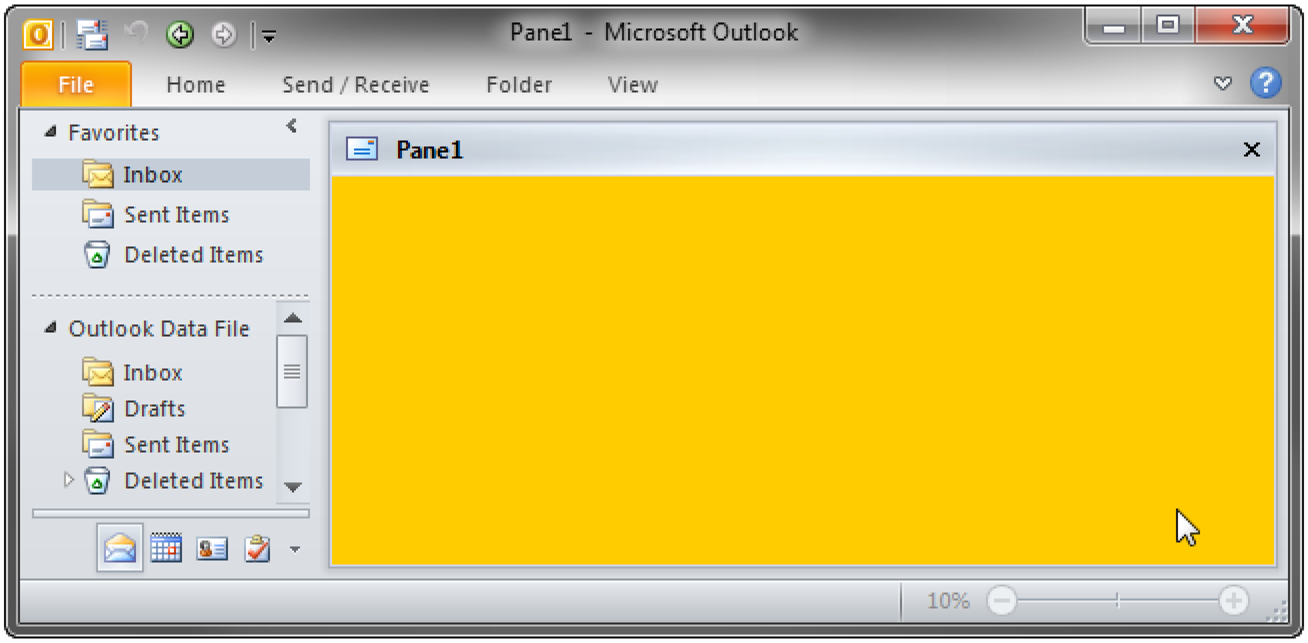
  1. The *Hidden* region state is not supported for docked regions
  2. Docked panes have limitations on the minimum height or width

- 

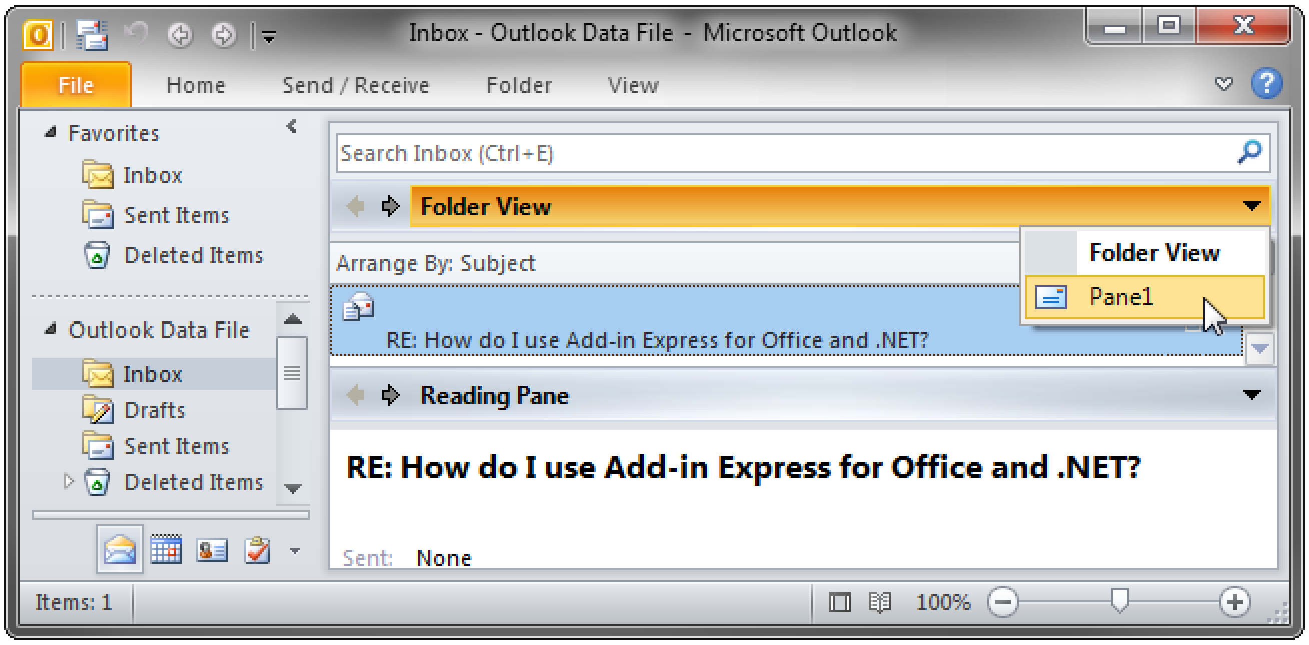
- Four regions around the Reading Pane – `LeftReadingPane`, `TopReadingPane`, `RightReadingPane`, `BottomReadingPane` (see the screenshot below)

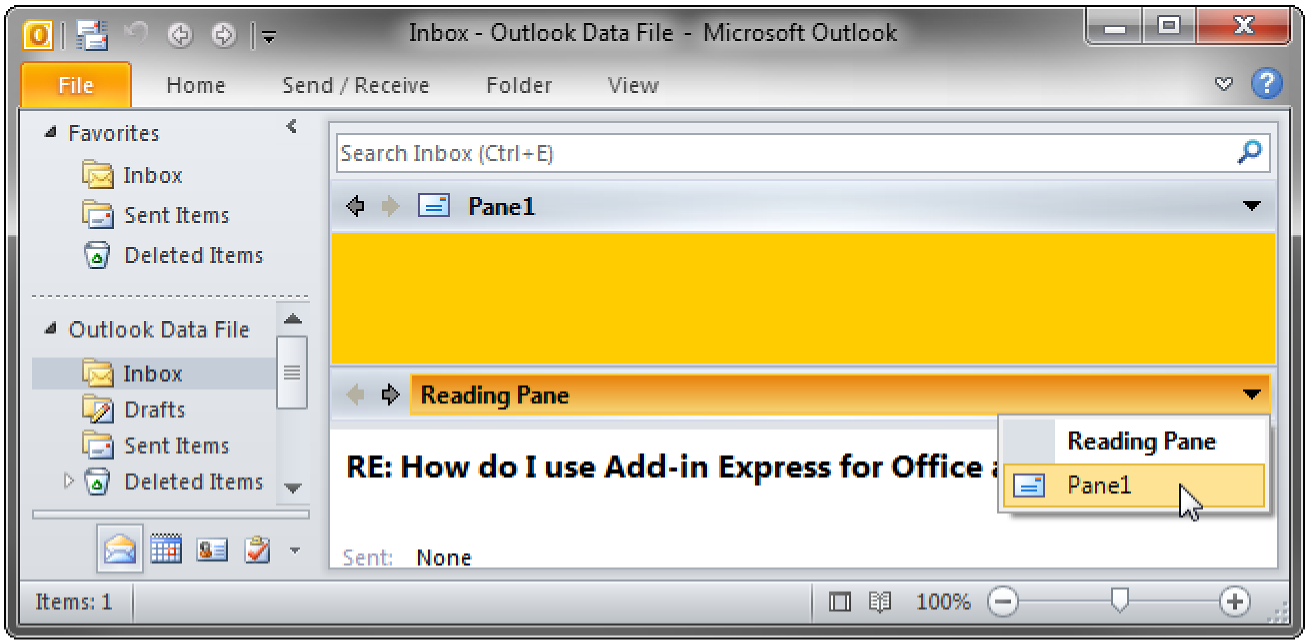


- The `WebViewPane` region (see the screenshot below). Note that it uses Outlook properties in order to replace the items grid with your form (see also [WebViewPane]()).



- The `FolderView` region (see two screenshots below). Unlike WebViewPane, it allows the user to switch between the original Outlook view and your form. **A restriction**: this region is not available for Calendar folders in Outlook 2010-2016.



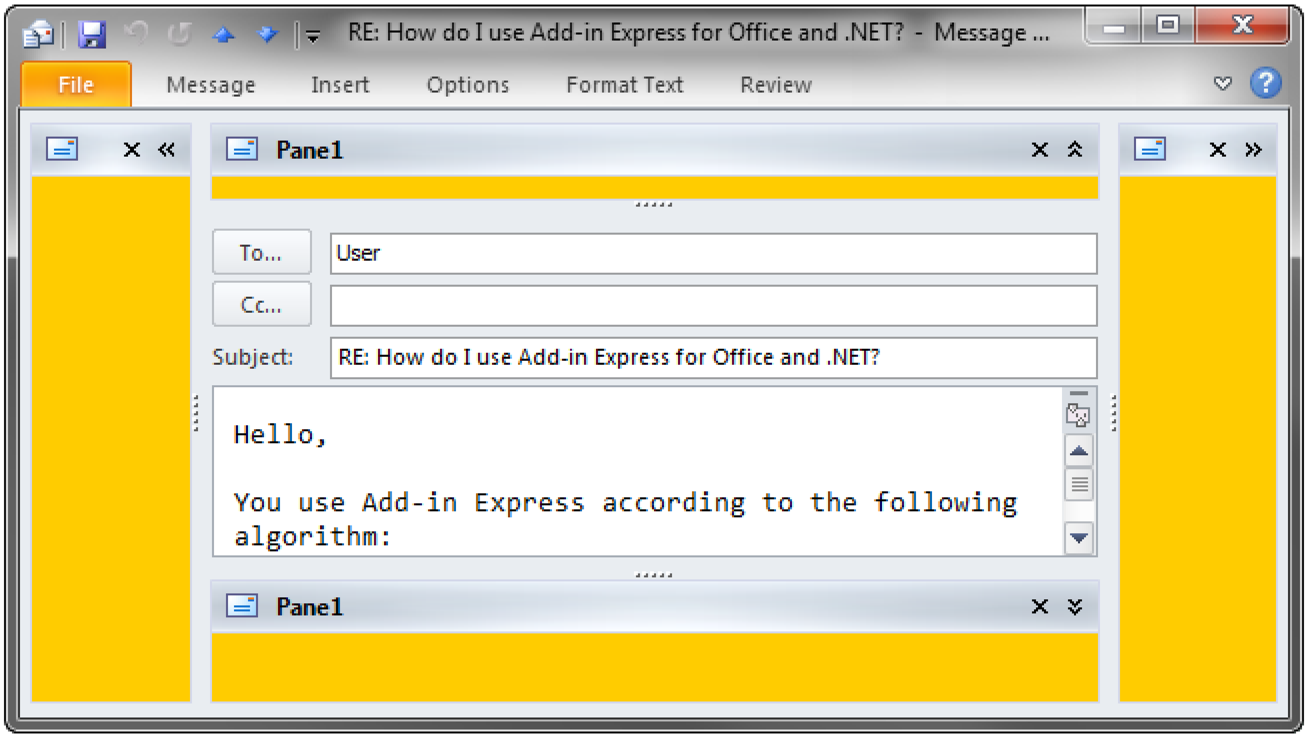


- The `ReadingPane` region (see two screenshots above)

Outlook form regions

And here is the list of **Outlook form regions**:

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



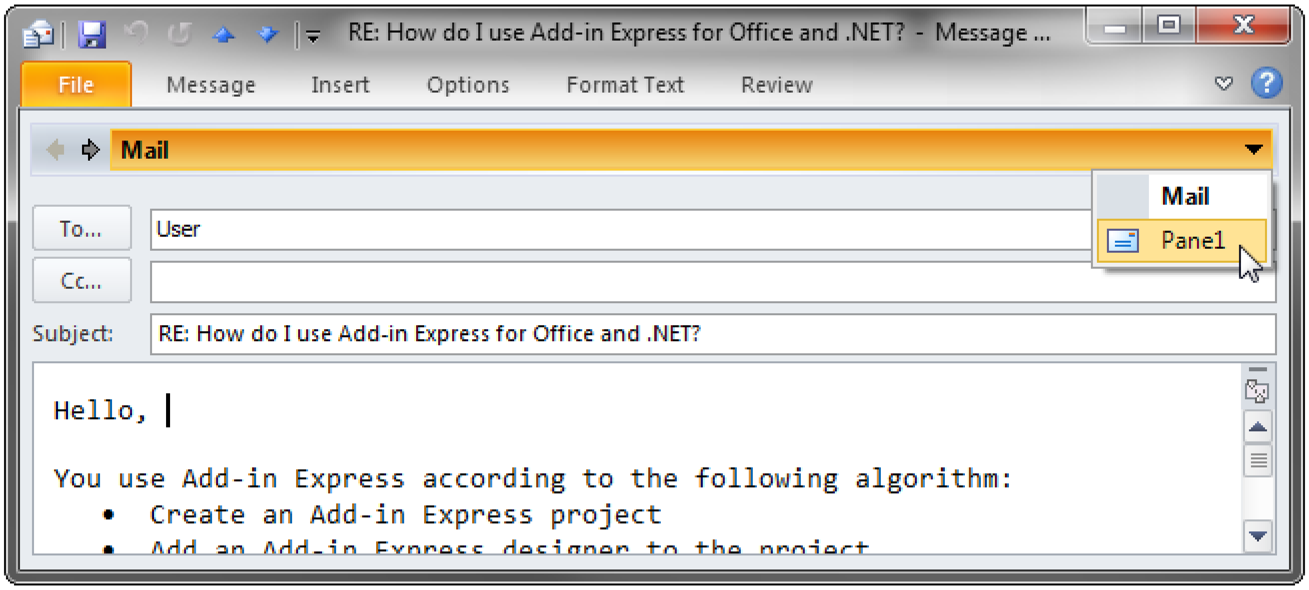
User

RE: How do I use Add-in Express for Office and .NET?

```
Hello,

You use Add-in Express according to the following
algorithm:
```

- The *InspectorRegion* region (see two screenshots below) allows switching between your form and the Outlook inspector pane.

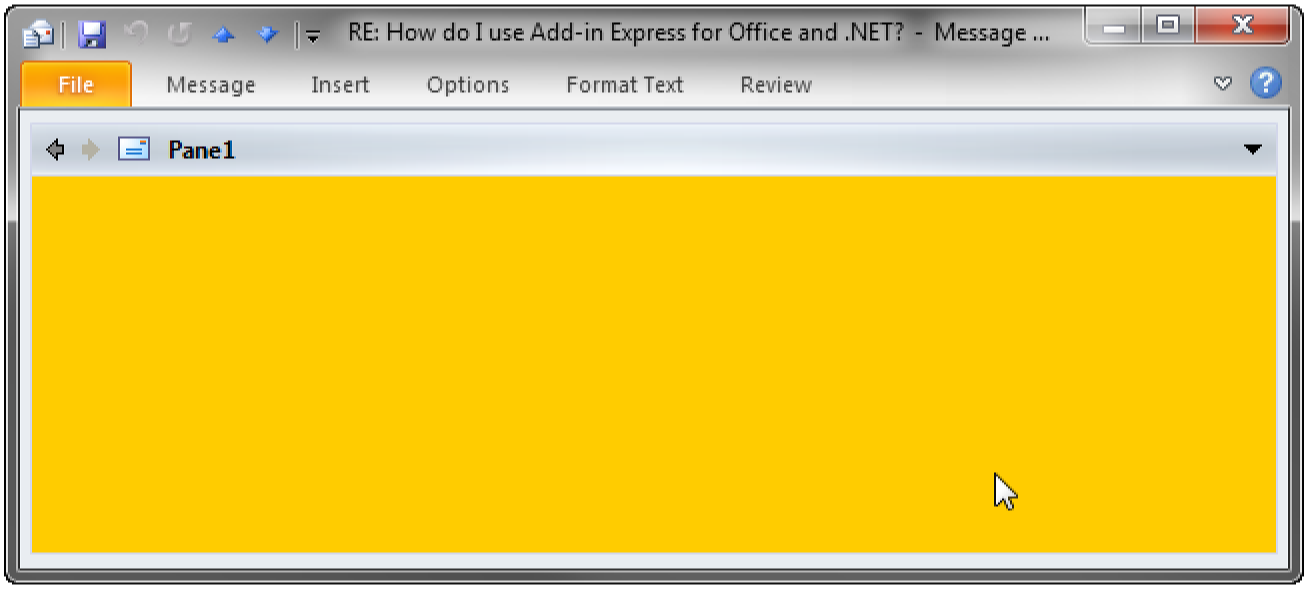


User

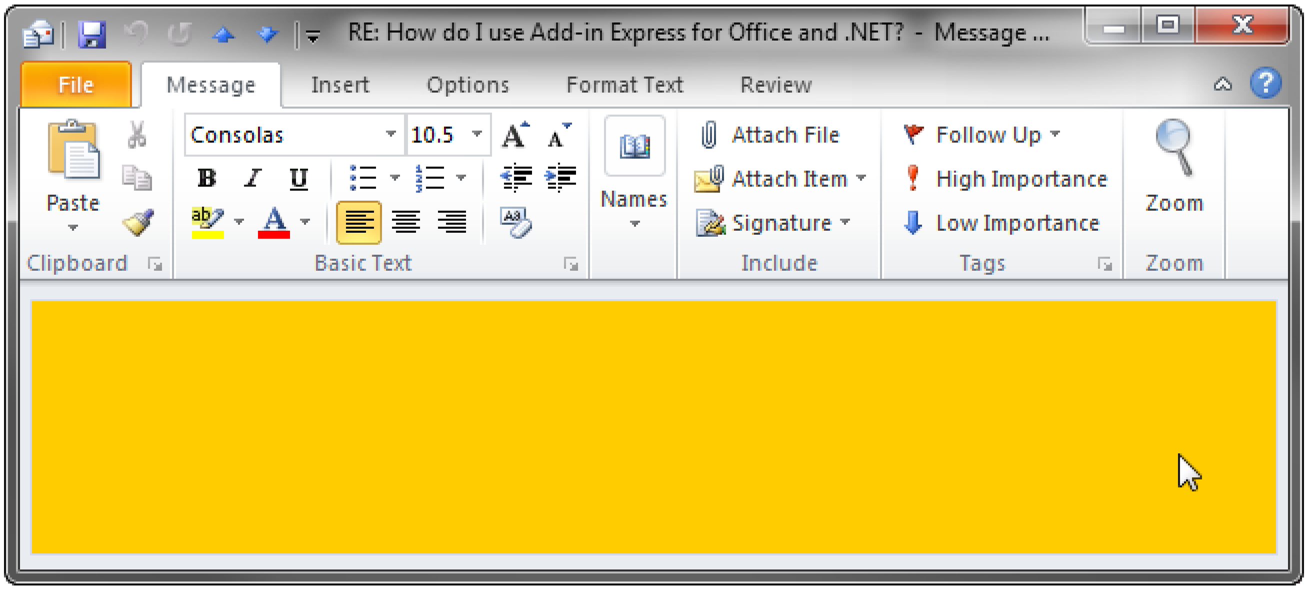
RE: How do I use Add-in Express for Office and .NET?

```
Hello,

You use Add-in Express according to the following algorithm:
    • Create an Add-in Express project
```


RE: How do I use Add-in Express for Office and .NET? - Message ...
File
Message
Insert
Options
Format Text
Review
Pane1

- The `CompleteReplacement` inspector region shown in the screenshot below is similar to the `InspectorRegion` with two significant differences: a) it doesn't show the header and in this way, it doesn't allow switching between your form and the Outlook inspector pane and b) it is activated automatically.



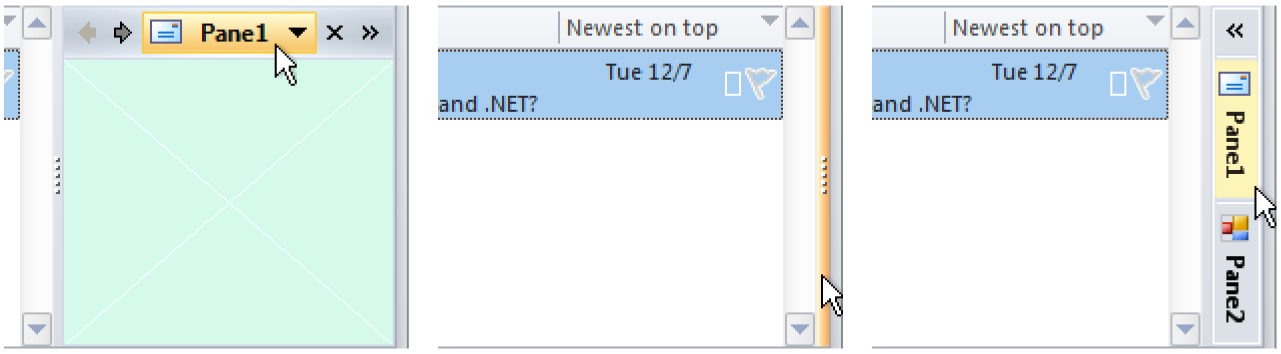
The UI Mechanics

A lot of useful information on the UI and programming interfaces that Advanced Outlook Form and View Regions provide.

Region States and UI-related Properties and Events

As mentioned in Terms and Definitions, the manager creates instances of the form region. An instance of the form region class (further on the instance is referenced as form) is considered visible if it is embedded into the specified sub-pane of an Outlook window. Note that the form may be actually invisible either due to the region state (see below) or because other forms in the same sub-pane hide it; anyway, in this case, `ADXOlForm.Visible` returns `true`. To prevent embedding the form into a sub-pane, you can set `ADXOlForm.Visible` to false in the event named `ADXBeforeFormShow`. When the form is shown in a sub-pane, the `Activated` event occurs and `ADXOlForm.Active` becomes `true`. When the user moves the focus onto the form, the `ADXOlForm` 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 manager removes the form from its sub-pane, `ADXOlForm.Visible` becomes `false` and the form generates the `ADXAfterFormHide` event.

In accordance to the value that you specify for the `DefaultRegionState` property of the item, 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).



Note however that `DefaultRegionState` will work only when you show the form in a particular sub-pane for the very first time and no other forms have been shown in that sub-pane before. You can reproduce this situation on your PC by choosing *Reset Regions* in the context menu of the form's designer surface.

You can change the state of your form at run-time using the `ADXOlForm.RegionState` property. When showing your form in certain sub-panes, you may need to show the native Outlook view or form that your form overlays; use the `ADXOlForm.ActivateStandardPane()` method.

When the region is in the hidden state, i.e. when it is collapsed to a 5px wide strip, the user can click on the splitter and the region will be restored (it will go to the normal state).

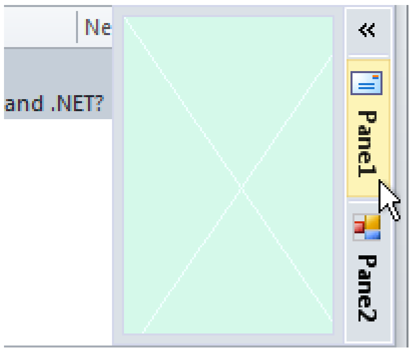
When the region is in the normal state, the user can choose any of the options below:

- change the region size by dragging 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 `ADXOlForm`; this option is not available for the end-user if you set `ADXOlFormsCollectionItem.IsHiddenStateAllowed = False`

- close the form by clicking on the *Close* button in the form header; this fires the `ADXCloseButtonClick` event of the `ADXOlForm`. The event is cancellable (see The Header and the Close Button); if the event isn't cancelled, the `Deactivate` event occurs, then the pane is being removed from the region (`ADXOlForm.Visible = false`) and finally, the `ADXAfterFormHide` event of the `ADXOlForm` 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 any of the three options below:

- restore 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 overlays a part of the Office application's window near the form region (see the figure at the right); 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 Header and the Close Button

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 `ADXOlFormsCollectionItem.AlwaysShowHeader` is set to `true`.

The *Close* button is shown if the `CloseButton` property of the `ADXOlFormsCollectionItem` is `true`.

Clicking on the *Close* button in the form header fires the `ADXCloseButtonClick` event of the `ADXOlForm`, 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

Outlook can show several instances of two window types (explorer and inspector) 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.

To access the form, which is currently active in Outlook, you use the *GetCurrentForm* method. To access all instances of the form in Outlook, you use the *FormInstances* method of *ADXOlFormsCollectionItem*.

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

You can check that a form is **not** available in the UI (say, you cancelled the *ADXBeforInstanceCreate* event or set *ADXOlForm.Visible = False* in the *ADXBeforeFormShow* event or the user closed it) by checking the *Visible* property of the form:

```
Function FindCurrentForm(ByVal item As ADXOlFormsCollectionItem) As ADXOlForm
    Dim hwndActive As IntPtr = Win32API.GetActiveWindow()
    For i As Integer = 0 To item.FormInstanceCount - 1
        Dim form As ADXOlForm = item.FormInstances(i)
        If form.Visible AndAlso form.Active Then
            'Active form in form region has Visible = true , Active = true
            'Inactive form in form region has Visible = true , Active = false
            If (hwndActive = form.Handle) OrElse _
                 (hwndActive = form.CurrentOutlookWindowHandle) Then
                Return form
            End If
        End If
    Next
    Return Nothing
End Function

Public Class Win32API
    <DllImport("User32")> _
    Public Shared Function GetActiveWindow() As IntPtr
    End Function
End Class
```

If the form is not available in the UI, you can show such a form in one step: call the *ApplyTo* method of the *ADXOlFormsCollectionItem*; the method accepts the parameter, which is either an *Outlook.Explorer* or *Outlook.Inspector*. This also changes the state of the region that shows the form to normal.

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 `ADXOlForm` to show the form on top of all other forms in that region. If the region was in either minimized or hidden state, calling `Activate` will also return it to the normal state.

Note that your form does not restore its `Active` state in subsequent sessions of the host application if several add-ins show their forms in the same region. In other words, if the current session ends with a given form on top of all other forms in that region, some other form may become active on the subsequent start of the host application. This is because add-ins are loaded 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 `ADXOlFormsManager`.

Due to the context-sensitivity features provided by the `ADXOlFormsCollectionItem`, an instance of your form will be created whenever the current Outlook context matches that specified by the corresponding `ADXOlFormsCollectionItem`.

Accessing Instances of Your Form Region

The user may open multiple Explorer and Inspector windows. That is, the Outlook Forms Manager will create multiple instances of your form region class now and then. How to retrieve the form instance shown in a particular Outlook window? How to get all form instances?

ADXOlFormsCollectionItem.GetForm()

This method returns an instance of your form region in the **specified** Outlook window.

ADXOlFormsCollectionItem.GetCurrentForm()

This method returns an instance of your form region in the **active** Outlook window.

Consider the following scenarios:

- In Accessing the Form from ThisAddIn, the algorithm relies on the fact that the `Click` event of a Ribbon button can occur in the active Outlook window only; accordingly, `GetCurrentForm()` returns the form instance embedded into the Inspector (Explorer) window in which the button is clicked.
- `GetCurrentForm()` will never find e.g. an Inspector form region if an Explorer window is active;
- Some add-in or antivirus may cause the `ExplorerSelectionChange` event to fire in an inactive Explorer window; that is, using `GetCurrentForm()` in an Explorer-related event may produce a wrong result. To avoid this, use `GetForm()` or make sure that `GetCurrentForm()` is called in the active window.

ADOlFormsCollectionItem.FormInstances()

This method allows enumerating all instances of your form region created for a given `ADOlFormsCollectionItem`.

From a Form Instance to the Outlook Object Model

The Outlook Forms Manager creates an instance of your form when the Outlook context matches the settings of the corresponding *ADOlFormCollectionItem*.

After creating the form instance, the manager sets a number of properties providing entry points to the Outlook object model; note that these properties are not set when the form region's constructor is running. The properties are listed below. Note that the state of the COM objects retuned by these properties is essential for Add-in Express functioning – you must not release them in your code because passing any of them to *Marshal.ReleaseComObject()* may cause Outlook to crash.

ADXOlForm.ExplorerObj	If the form is embedded (*ADXOlForm.Visible=True*) into an Outlook Explorer window, returns a reference to the corresponding Outlook.Explorer object (a COM object). Otherwise, returns *null* (*Nothing* in VB.NET).
ADXOlForm.InspectorObj	If the form is embedded (*ADXOlForm.Visible=True*) into an Outlook Inspector window, returns a reference to the corresponding Outlook.Inspector object (a COM object). Otherwise, returns *null* (*Nothing* in VB.NET).
ADXOlForm.FolderObj	If the form is embedded into an Outlook Explorer window (*ADXOlForm.ExplorerObj* is not *null*), returns a reference to the *Outlook.MAPIFolder* object (a COM object) representing the current folder in the Explorer window. If the form is embedded into an Outlook Inspector window (*ADXOlForm.InspectorObj* is not *null*), returns a reference to the *Outlook.MAPIFolder* object (a COM object) representing the parent folder of the Outlook item which is shown in the Inspector window.
ADXOlForm.FolderItemsObj	If the form is embedded into an Outlook Explorer window (*ADXOlForm.ExplorerObj* is not *null*), returns a reference to the *Outlook.Items* object (a COM object) representing the collection of items of the current folder in the Explorer window. If the form is embedded into an Outlook inspector window (*ADXOlForm.InspectorObj* is not *null*), returns a reference to the *Outlook.Items* object (a COM object) representing the collection of items in the parent folder of the Outlook item which is shown in the Inspector window.
ADXOlForm.OutlookAppObj	Returns a reference to the *Outlook.Application* object (a COM object) representing the Outlook application into which the add-in is loaded.

Resizing the Form

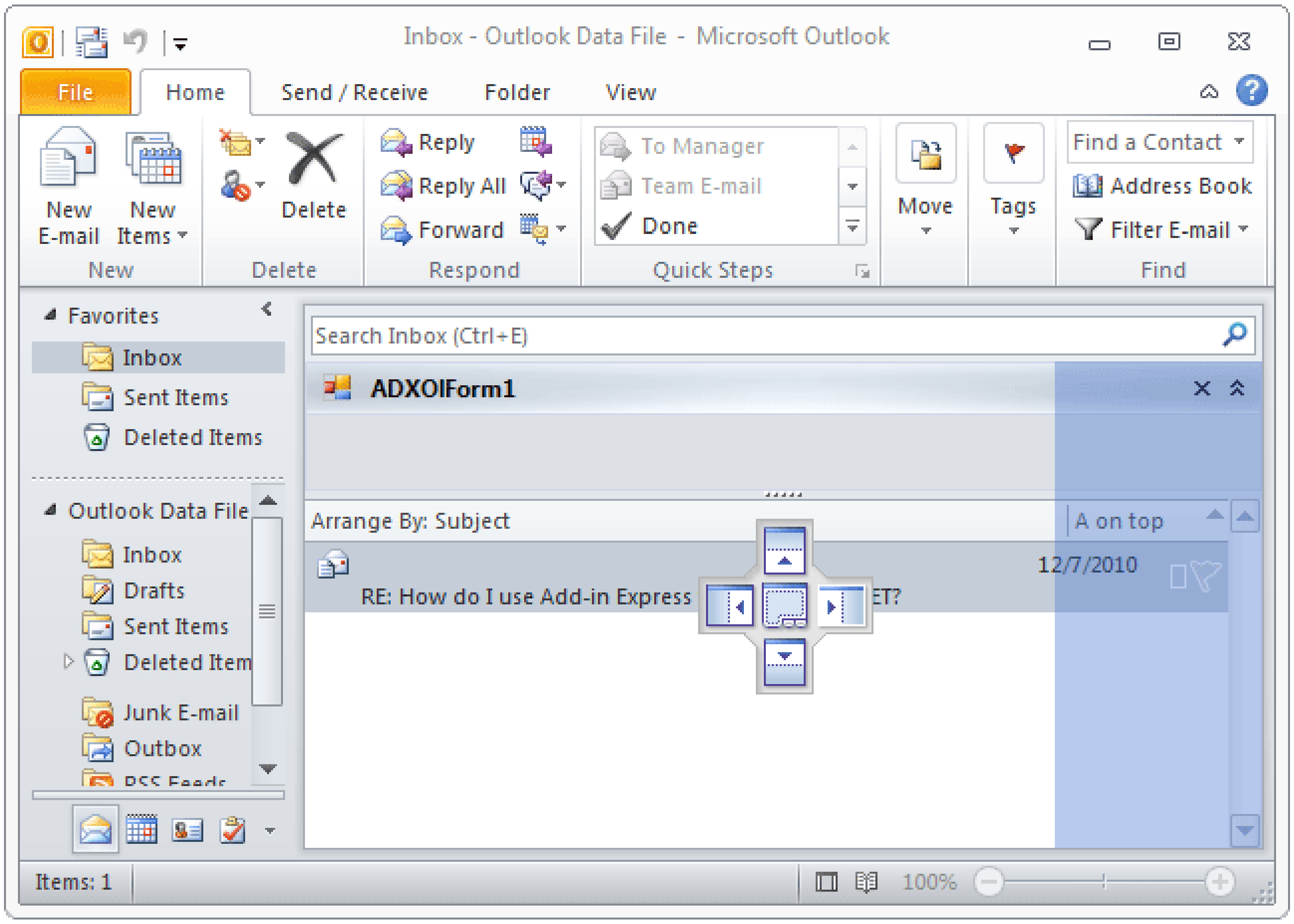
There are two values of the `ADXOlFormsCollectionItem.Splitter` property. The default one is `Standard`. This value shows the splitter, the user drags the splitter to change the form size as required. The form size is stored in the registry so that the size is restored whenever the user starts the host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

You can only resize your form programmatically, if you set the `Splitter` property to `None`. This prevents the user form resizing the form. Changing the `Splitter` property at run time does not 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 form was ever shown in this region, the form will be shown using the appropriate dimensions that you set at design-time. During subsequent sessions of the host application, the form will be shown using the dimensions set by the user.

Drag-n-Drop and Advanced Form Regions

The form can be dragged only if all of the following conditions are met A) it has the header (see The Header and the Close Button), B) you set `ADXOlFormsCollectionItem.IsDragDropAllowed=True` and C) you specify the positions in which your form can be dropped (see the `ADXOlFormsCollectionItem.ExplorerAllowedDropRegions` property). The form is dragged in the VS style, see the screenshot below.



Coloring up the Form

By default, the background color of the form is set automatically to match the current Office 2007-2016 color scheme. To use the background color of your own in these Office versions, you need to set *ADXOlFormsCollectionItem.UseOfficeThemeForBackground = True*.

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 *OnInitialize* event of the *ADXOlFormsManager* class.

Changing the *Enable*, *Cached*, *FormClassName* properties at run-time deletes all form instances created by the *ADXOlFormsCollectionItem*.

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

Changing the *ExplorerLayout* or *InspectorLayout* properties of the *ADXOlFormsCollectionItem* changes the position for all visible form instances.

Changing the *Splitter* and *Tag* properties of the *ADXOlFormsCollectionItem* doesn't do anything for the currently visible form instances. You will see the changed splitter when the manager shows a new instance of the *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. An example of the folder path is "\\Personal Folders\Inbox".

A special value in *FolderName* is an asterisk (*'*'*), which means "all folders". You can also specify **message class(es)** for which your form will be shown. Note that all context-sensitivity properties of an *ADXOlFormsCollectionItem* are processed using the **OR** Boolean operation. That is, specifying e.g. folder names extends, but not limits, the list of contexts for which your form will be shown.

In advanced scenarios, you can also use the *ADXOlFormsManager.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 `ADXFolderSwitch` or `ADXSelectionChange` events of `ADXOlForm`.

Caching Forms

By default, whenever the forms manager 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, the forms manager creates another set of cached forms. Forms shown in Inspector windows cannot be cached.

WebViewPane

When this value (see Advanced Outlook Form and View Regions) is chosen in the `ExplorerLayout` property of `ADXOlFormsCollectionItem`, Advanced Outlook Form Regions uses the `WebViewUrl` and `WebViewOn` properties of `Outlook.MAPIFolder` (also `Outlook.Folder` in Outlook 2007-2016) in order to show your form as a home page for a given folder(s).

Add-in Express doesn't allow using `WebViewPane` for *PublicFolders*, *Outbox* and *Sync Issues* folders in Outlook as well as all folders below them.

Please note that `WebViewPane` cannot be used for an Outlook folder if the `WebViewOn` and `WebViewUrl` properties of the folder object cannot be set. A possible solution is to enable *Allow Script in shared folders* and *Allow Script in Public Folders* options in the security settings. See also http://support.microsoft.com/kb/923933.

Tips and Notes

These are some useful tips from the development team.

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. This may occur several times during the lifetime 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.

Useful Links

Please check the Learning Center and our blog archive.

Reset Regions

Form regions preserve their position, size and state between sessions. To reproduce the "first-start" scenario on the development PC, you use the "*Reset Regions*" command located in the context menu of the designer surface of the form region.

Identifying Outlook Windows

You pass an object corresponding to an Outlook window (such as `Outlook.Explorer`) to `ADXOlFormsManager.GetOutlookWindowHandle`; it returns an `IntPtr`, which is the WinAPI handle of the Outlook window. Then you loop through instances of your forms querying `ADXOlForm.CurrentOutlookWindowHandle`; it returns the handle of the Outlook window in which the form is shown.

Finally

If your questions are not answered here, please see the HOWTOs section on our web site: see http://www.add-in-express.com/support/add-in-express-howto.php. You can also search our forums for an answer; the search page is http://www.add-in-express.com/forum/search.php. Another useful resource is our blog – see http://www.add-in-express.com/creating-addins-blog/.