Official Guide

ADX Extensions™
for Microsoft® Outlook®

Getting Started
The ADX Extensions for Microsoft Outlook, .NET and VCL Editions

The ADX Extensions™ for Microsoft® Outlook®

Document version **1.3**
Revised at **28-Jul-06**
Product version **1.3**

Content

Introduction

Microsoft gives Office developers a way to extend and improve its Office applications using several special technologies. Office 2000 started IDTExensibility2 for creating COM add-ins. With Office 2002 (XP) Microsoft published the Smart Tag technology to introduce context sensitivity into its applications and the Excel RTD Server technology to replace archaic DDE with a modern solution. Office 2003 enhanced applications' object models and supports these technologies by other applications. As a result, developers have several powerful and effective approaches to embedding their applied code into Microsoft Office applications. However, Office developers want to achieve much more.

This document describes the ADX Extensions™ for Outlook and what it provides to extent the technologies supported by Microsoft.

Welcome to the ADX Extensions for Outlook

Welcome to the ADX Extensions for Outlook. **The ADX Extensions™ for Outlook** (or **ADX X for Outlook**) is a plug-in for Add-in Express™ designed to provide unique features for creating a commercial class UI for your Microsoft Outlook add-ins. Using ADX X for Outlook you can easily create sophisticated user interfaces found in today's most recognizable commercial Outlook add-ins. Now Outlook developers use Add-in Express and the ADX Extensions for Outlook to create protected Outlook-based solutions with feature-rich Outlook add-ins.

The ADX Extensions for Outlook is built on our own exclusive Add-in Express™ technology and based on true RAD approaches inherited from Microsoft shared solutions which provides a flexible and the fastest way to program stable and powerful Outlook-based solutions. The flexibility of the ADX X for Outlook object model lets you address every level of the object hierarchy; the ADX X for Outlook event model lets you code to the precise action.

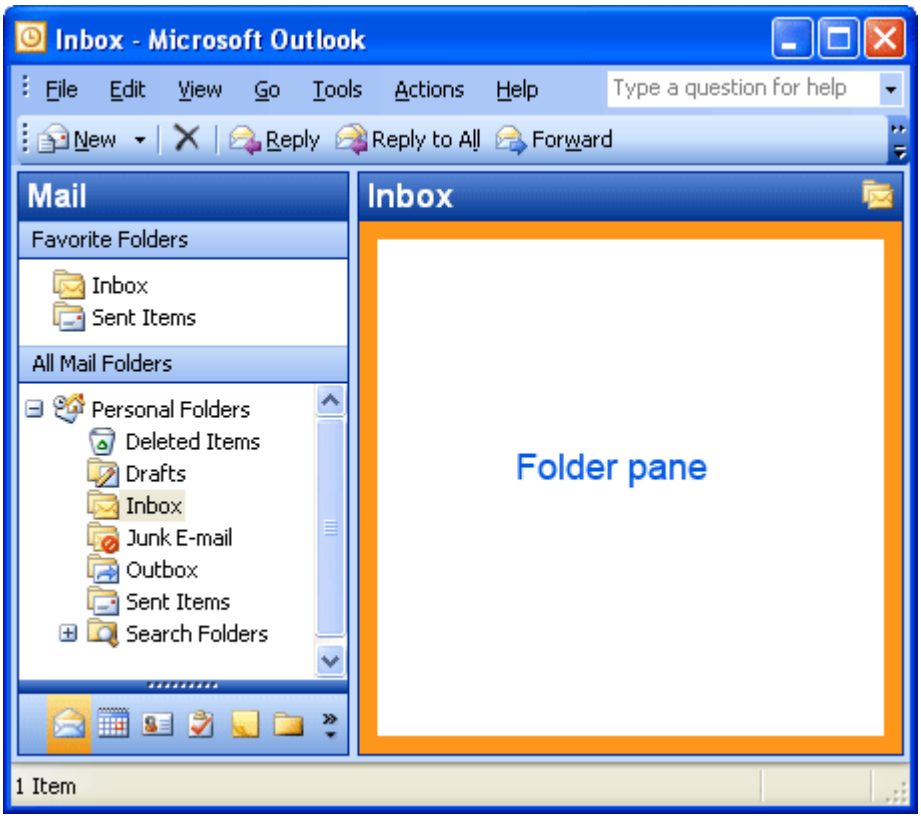
The ADX Extensions for Outlook was architected from the ground up to overstep the limits of the existing technology and to deliver tomorrow’s solutions today. We gathered all our experience in Outlook, .NET and VCL development and, in fact, released a unique product.

What are panes and sub-panes

The ADX Extensions for Outlook was designed and developed to allow you to integrate your forms with two main Outlook windows: the Explorer and Inspector windows. ADX X for Outlook may help you embed your forms into Outlook views and Outlook forms. With the ADX Extensions for Outlook you can completely replace any Outlook view with your forms, add your forms to any Outlook views and to any Outlook forms.

Folder panes

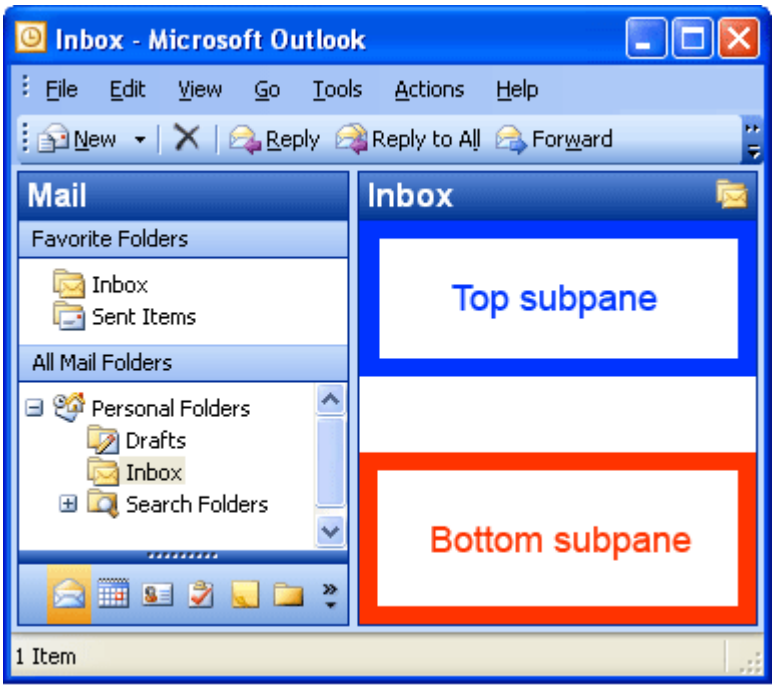
The first feature provided by ADX X for Outlook is folder panes. This feature allows you to replace the content of the folder pane with your own form. In this case ADX X for Outlook embeds your form into a folder web-view created specially for this purpose. The following picture shows what a folder pane is.

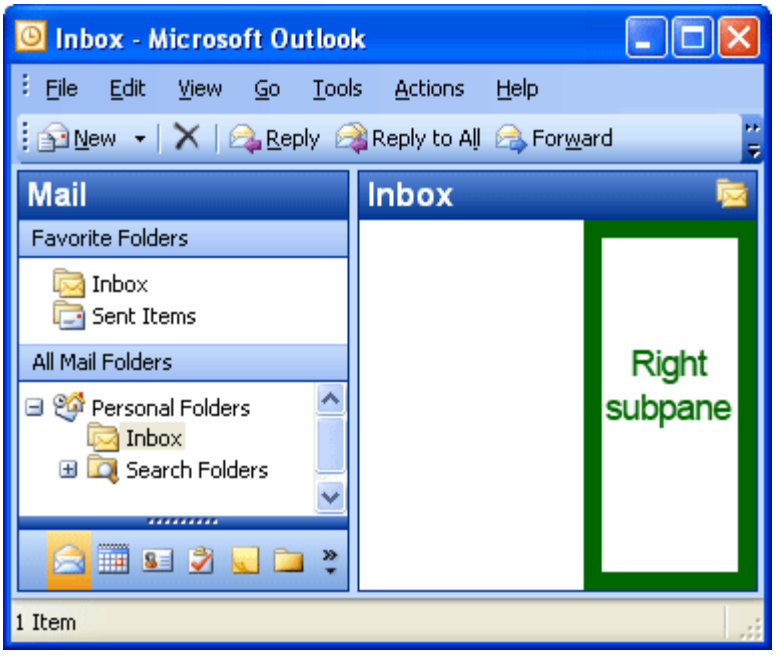


For example, you can use folder panes to integrate your non-Outlook data as data from your SQL Servers, CRM, CMS or DMS. You can do it by creating an empty folder, binding it to your form and interacting with your data using this form. Another approach is to use folder panes to create feature-rich Outlook views that contain more effective controls such as tree views, grids, tab controls, etc.

Folder sub-panes

In addition to folder panes, the ADX Extensions for Outlook allows you to embed your forms into folder sub-panes. Folder sub-panes may help you make your form context-sensitive to the Outlook selection. E.g., your form may show the sales volume for the selected contact, or the pay schedule for all selected tasks, etc. Now in the view of the UI design, only one form into one sub-pane (top, bottom or right) is supported.





Form sub-panes

Also, the ADX Extensions for Outlook allows you to embed your custom forms into the Outlook Inspector windows such as e-mail forms, contact forms, task forms, etc. Form sub-panes can help you customize any built-in and custom Outlook forms, and use any controls (including grids, tree views, etc.) on Outlook forms. Please note, in the view of the UI design, the ADX Extensions for Outlook supports embedding only one form into one sub-pane of the Outlook Inspector window (bottom, left or right).

Supported folders and forms

For folder panes and folder sub-panes the ADX Extensions for Outlook supports all types of folders including E-mail, Contact, Appointment, Task, Journal, Note, Post, and Distribution List. However, there is one exception: folder panes cannot be created for the *Outbox* folder.

For form sub-panes the ADX Extensions for Outlook supports all built-in Outlook forms including Recipient, Appointment, MeetingRequest, MeetingCancellation, MeetingResponseNegative, MeetingResponsePositive, MeetingResponseTentative, Contact, Journal, Mail, Post, Task, TaskRequest, TaskRequestUpdate, TaskRequestAccept, TaskRequestDecline, DistributionList. In addition to built-in forms, ADX X for Outlook can be used to embed sub-panes into all custom forms defined by custom form class names.

Be Aware

The examples and their descriptions given here are very simple and brief. Don't feel intimidated. To use the ADX Extensions for Outlook is quite easy. This tool allows you to start quickly, to avoid any difficulties and pitfalls when embedding your forms into Outlook windows. However, you should take into account that we deliberately avoid any descriptions of Outlook objects, their properties, methods and events. This documentation implies that you have some experience in using Add-in Express for developing Outlook add-ins as well as some experience with the Outlook object model.

System requirements

Now the ADX Extensions for Outlook is available in two editions, .NET and VCL. The .NET Edition of the ADX Extensions for Outlook (ADX.NET X for Outlook) supports Visual Basic .NET 2003 and 2005, Visual C# 2003 and 2005, and RemObjects Chrome 1.5.3 and higher on .NET Framework 1.1 and 2.0. The VCL edition supports all versions of Borland Delphi starting from version 5. The complete system requirements are listed below.

The ADX Extensions for Outlook (ADX X for Outlook) is compatible with all Outlook versions starting from Outlook 2000 and requires Add-in Express installed.

Required Add-in Express versions

ADX.NET X for Outlook	ADX.VCL X for Outlook
▪ Add-in Express .NET version 2.6 or higher 2.x	▪ Add-in Express VCL version 2.7 and higher 2.x

Supported Outlook versions

ADX.NET X for Outlook	ADX.VCL X for Outlook
▪ Outlook 2000 with / without updates ▪ Outlook 2002 (XP) with / without updates ▪ Outlook 2003 with / without updates	▪ Outlook 2000 with / without updates ▪ Outlook 2002 (XP) with / without updates ▪ Outlook 2003 with / without updates

Supported programming languages

ADX.NET X for Outlook	ADX.VCL X for Outlook
▪ Visual Basic .NET 2005 ▪ Visual Basic .NET 2003 ▪ Visual C# .NET 2005	▪ Delphi 5 with update pack 1 ▪ Delphi 6 with update pack 2 ▪ Delphi 7 with update pack 1

▪ Visual C# .NET 2003 ▪ RemObjects Chrome 1.5	▪ Delphi 2005 with update pack 3 ▪ Delphi 2006 with update pack 2

Supported IDE versions

ADX.NET X for Outlook	ADX.VCL X for Outlook
▪ Visual Studio .NET 2005, Team System (all editions) ▪ Visual Studio .NET 2005, Professional ▪ Visual Studio .NET 2005, Standard ▪ Visual Basic .NET 2005, Standard ▪ Visual C# .NET 2005, Standard ▪ Visual Basic .NET 2005, Express ▪ Visual C# .NET 2005, Express ▪ Visual Studio .NET 2003, Enterprise (all editions) ▪ Visual Studio .NET 2003, Professional ▪ Visual Studio .NET 2003, Standard ▪ Visual Basic .NET 2003, Standard ▪ Visual C# .NET 2003, Standard	▪ Delphi 5, Professional and other high-level edition ▪ Delphi 6, Professional and other high-level edition ▪ Delphi 7, Professional and other high-level edition ▪ Delphi 2005, Professional and other high-level edition ▪ Delphi 2006, Professional and other high-level edition

Note

- Evaluation or trial versions of RemObjects Chrome and Borland Delphi are not supported.

Getting support

If you own a copy of the ADX Extensions for Outlook, you are entitled to certain benefits regarding support services offered by the Add-in Express Team.

Register on website

Visit our website and create a member profile. It is important that your most current information is entered into your member profile. This will ensure that our support service team can deliver support correspondence.

Note

- You can use the direct link to create a member profile.

Getting Help

You can obtain technical support using our website at www.add-in-express.com. Every registered user can submit support issues via a special web-form. In addition, on our website there is a customer community actively supported by the Add-in Express Team, the HOWTOs section with "how to" examples, reference add-ins (ADX Toys), Premium Zone and much more.

Important

- Please consult the support service options of your subscription before submitting a support issue.

Getting started on .NET

The ADX Extensions for Outlook is a visual tool. It is based on the Windows API and comprised of over twenty internal classes. But all the internal classes were designed to provide only two public components. This makes ADX X for Outlook easy-to-use in accordance with the true RAD paradigm. So, the ADX Extensions for Outlook provides a very comfortable way to enhance the GUI of your Outlook add-ins with minimal service coding. You write your applied code only.

This section shows how you can quickly get started with the ADX Extensions for Outlook in Visual Studio .NET and describes what the ADX Extensions for Outlook adds to your add-ins based on Add-in Express.

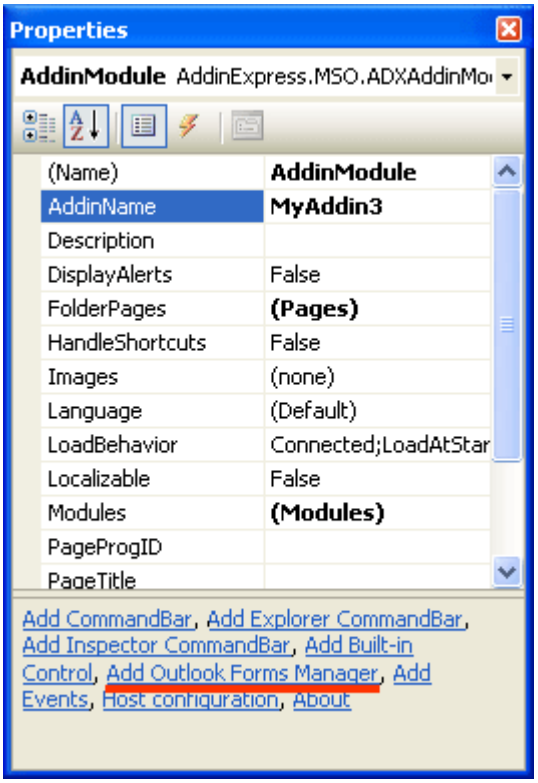
What is added to Add-in Express

The ADX Extensions .NET for Outlook is a plug-in for Add-in Express .NET that adds its own command, wizard and components, namely:

- A **new command**, “Add Outlook Forms Manager”, is added to the add-in module. The command includes the ADX Extensions for Outlook in the current add-in and adds a special component, the Outlook Forms Manager, to the add-in module.
- A **new wizard**, “ADX Outlook Form”, is added to the “Add New Item” dialog of the add-in project. The wizard adds a new Outlook form to the current add-in project.
- Two new components, the **Outlook Forms Manager** and **Outlook Form**, are published by ADX X for Outlook. They implement the ADX X Extensions for Outlook functionality.

New command

The ADX Extensions for Outlook adds a new command to the add-in module command set. It is the “**Add Outlook Forms Manager**” command.



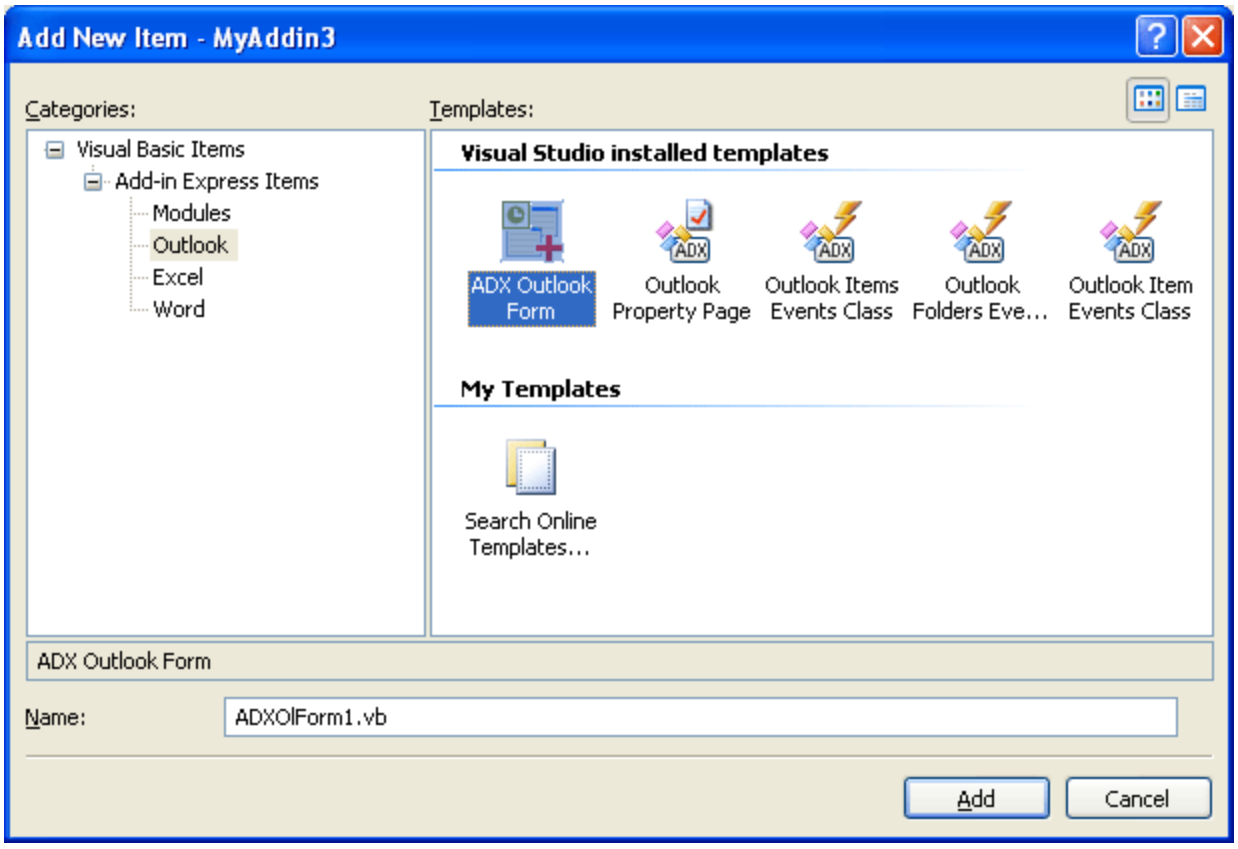
The command enables the ADX Extensions for Outlook for the current add-in solution, includes all references necessary to support ADX X for Outlook by the current add-in setup project and the shim, and adds a special component, **ADXOlFormsManager** (the Outlook Forms Manager), to the add-in module.

Note

- You should run the command before adding embedded forms.

New wizard

The ADX Extensions for Outlook provides a special form class, ADXOlForm, that implements a form embedded into the Outlook windows. You can add a new form based on ADXOlForm via a special wizard, "**ADX Outlook Form"**, available through the *Add New Item* dialog of the add-in project.



Note

- You can run the wizard after running the “Add Outlook Forms Manager” command.

Outlook Forms Manager

The ADX Extensions for Outlook publishes two main classes used for embedding your forms into Outlook. The most important component of ADX X for Outlook is Outlook Forms Manager (ADXOlFormsManager). Outlook Forms Manager centralizes and controls your forms and binds them to Outlook folders.

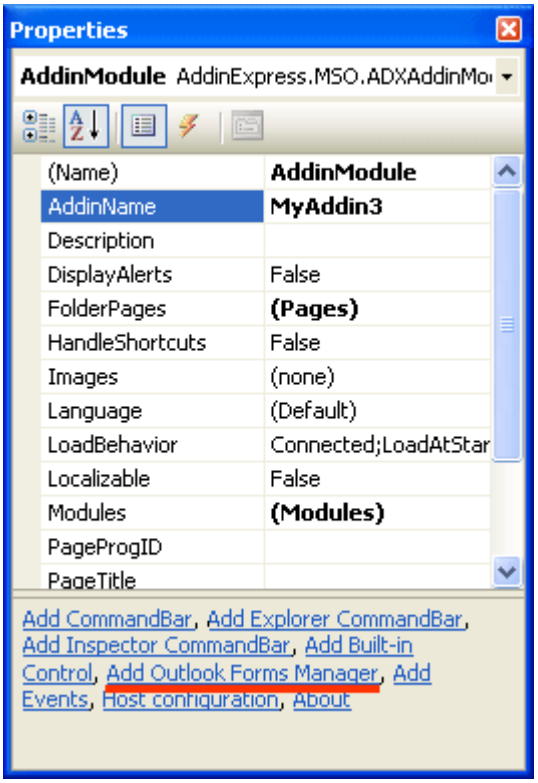
Forms embedded into Outlook

The ADX Extensions for Outlook implements a special form class that can be embedded into Outlook windows. The class is called **ADXOlForm** and is a descendant of Windows.Forms.Form. In order to be embedded into Outlook windows, all your Outlook forms should be descendants of ADXOlForm. You can add a new embedded form to your add-in project via the wizard described above. ADXOlForm publishes several Outlook-specific properties and events that can be used to access Outlook objects from an embedded form.

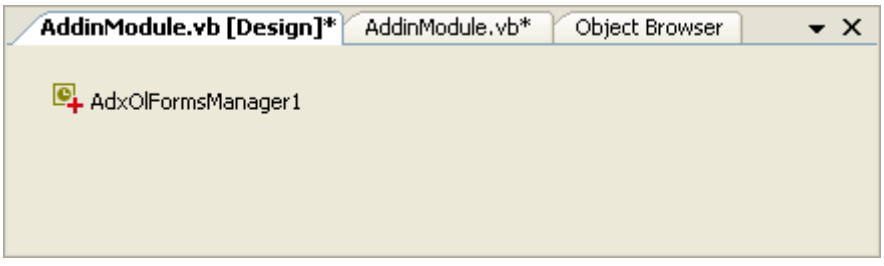
Your first Folder Sub-pane

1. Add the Outlook Forms Manager

The ADX Extensions for Outlook adds to the add-in module commands set a special command, "**Add Outlook Forms Manager**", available on the command area of the Properties window or by right-clicking the add-in module.

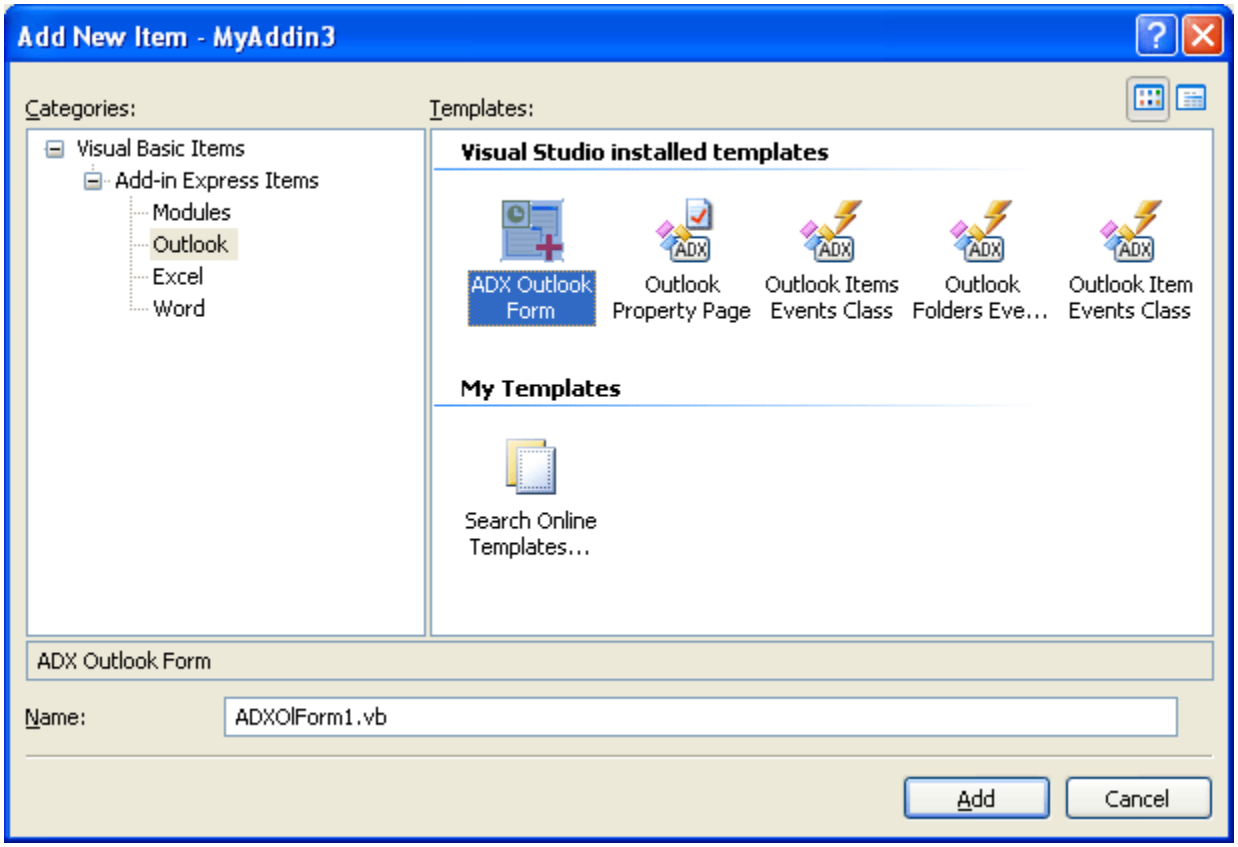


To include the ADX Extensions for Outlook functionality in your Outlook add-in project, select the add-in module and run the “Add Outlook Forms Manager” command. The command adds the *Outlook Forms Manager* component to the add-in module.



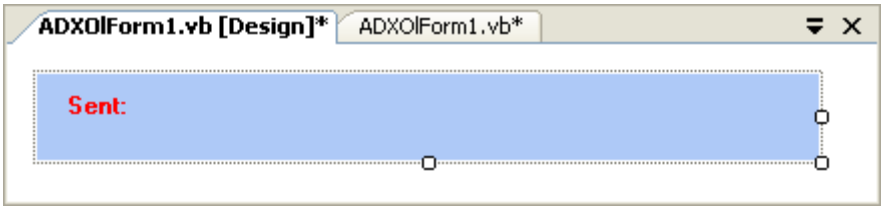
2. Add a new embedded form

Then, run the "**ADX Outlook Form**" wizard from the *Add New Item* dialog of the add-in project. The wizard adds a new form to the add-in project.



3. Customize the form

On your form, you can use any .NET controls such as calendars, edit boxes, grids, list views, etc. In this example we added a label to show when the selected message was sent.



4. Access to Outlook object

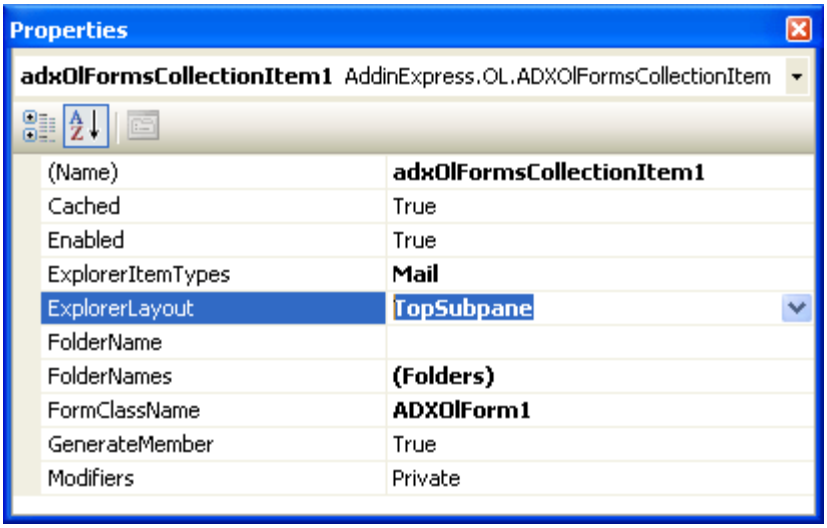
ADXOlForm provides access to all Outlook objects and events. For example, you can handle the *ADXSelectionChange* event to synchronize your form with selection changes in the current Explorer window:

```
C#
  private void ADXOlForm1_ADXSelectionChange()
   {
      Outlook._Application OlApp = this.OutlookAppObj as Outlook._Application;
      Outlook.Selection Selection = OlApp.ActiveExplorer().Selection;
      if (Selection.Count != 0)
      {
         Outlook.MailItem item = Selection[1] as Outlook.MailItem;
         SentLabel.Text = "Sent: " + item.SentOn;
      }
   }

VB.NET
   Private Sub ADXOlForm1_ADXSelectionChange() Handles Me.ADXSelectionChange
      Dim OlApp As Outlook._Application = CType(OutlookAppObj, Outlook._Application)
      Dim Selection As Outlook.Selection = OlApp.ActiveExplorer().Selection
      Dim Item As Outlook.MailItem

      If Selection.Count <> 0 Then
         Item = CType(Selection(1), Outlook.MailItem)
         SentLabel.Text = "Sent: " + Item.SentOn
      End If
   End Sub
```

5. Binding the form to Outlook folders

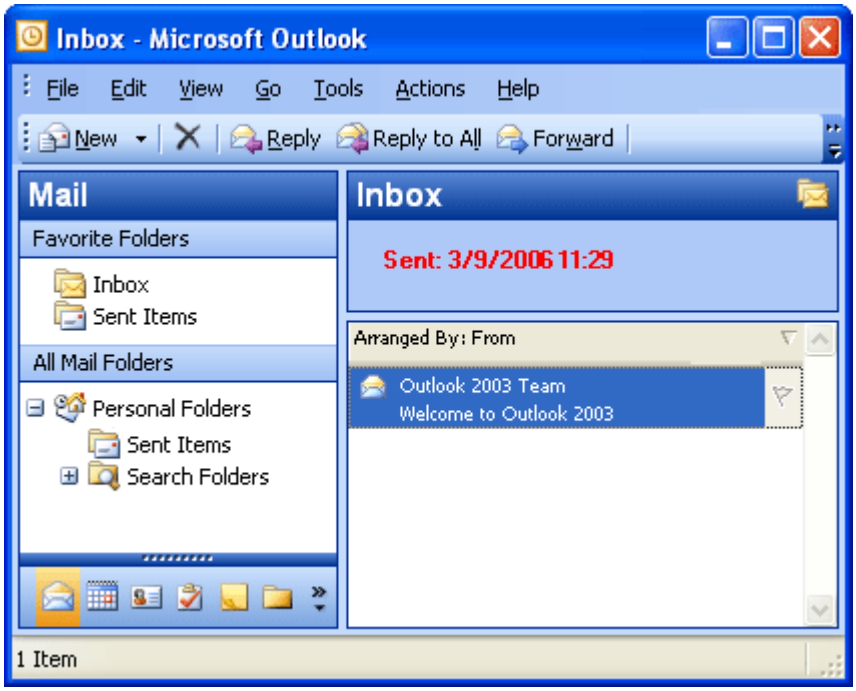
To specify the folders which your form is displayed for, you should add a new item to the *Items* collection of the Outlook Forms Manager, select your form class in the *FormClassName* property and specify the Outlook folder via three special properties, *FolderName*, *FoldersNames* and *ExplorerItemTypes*, that are common for all Outlook-related classes provided by Add-in Express. Then, with the *ExplorerLayout* property you specify the

pane or sub-pane which the form is placed on: *WebViewPane* that replaces the content of the Explorer window with a folder pane, or *RightSubpane*, *TopSubpane* or *BottomSubpane* that create corresponding folder sub-panes.



6. Run your add-in

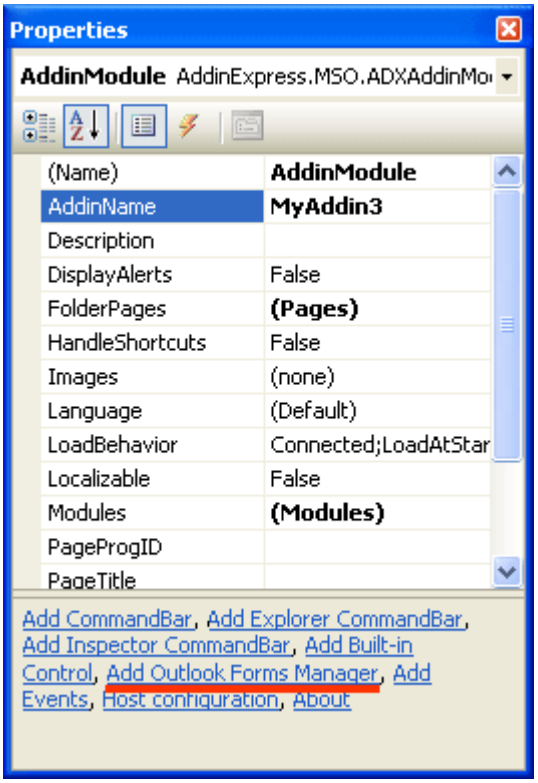
Finally, you rebuild the add-in project, run Outlook and find your form embedded.



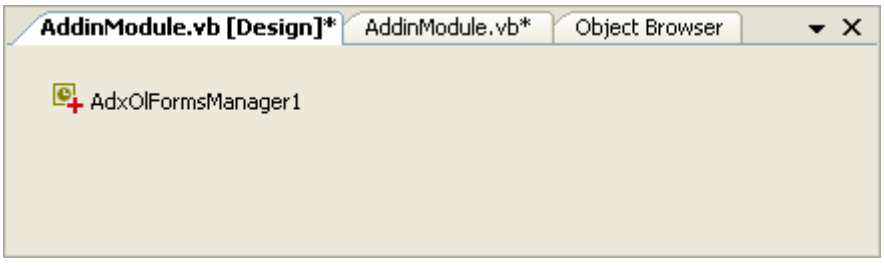
Your first Form Sub-pane

1. Add the Outlook Forms Manager

The ADX Extensions for Outlook adds to the add-in module commands set a special command, "**Add Outlook Forms Manager**", available on the command area of the Properties window or by right-clicking the add-in module.

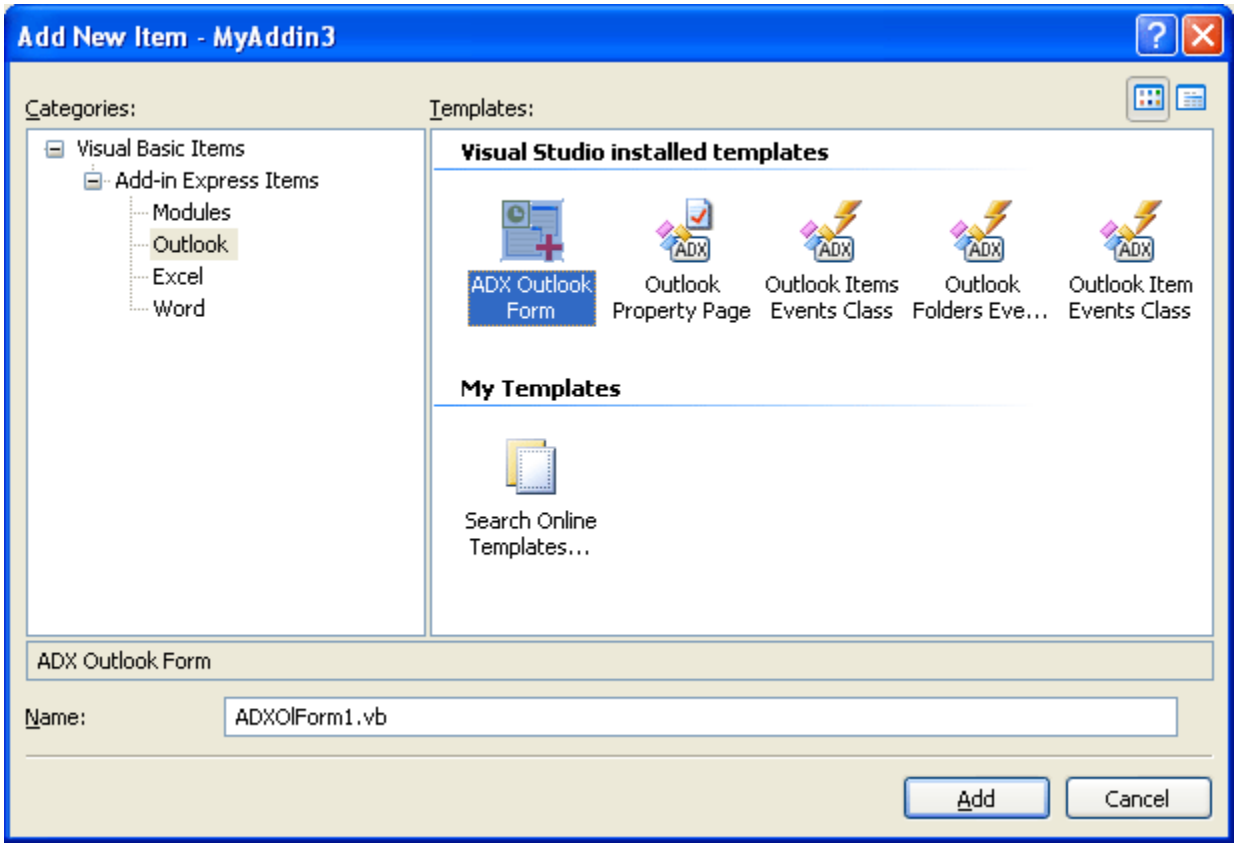


To include the ADX Extensions for Outlook functionality in your Outlook add-in project, select the add-in module and run the “Add Outlook Forms Manager” command. The command adds the *Outlook Forms Manager* component to the add-in module.



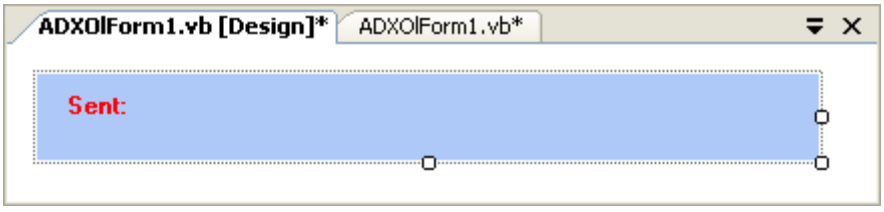
2. Add a new embedded form

Run the "**ADX Outlook Form**" wizard from the Add New Item dialog of the add-in project. The wizard adds a new form to the add-in project.



3. Customize the form

On your form, you can use any .NET controls such as calendars, edit boxes, grids, list views, etc. In this example we added a label to show when the selected message was sent.



4. Access to Outlook object

ADXOlForm provides access to the base Outlook objects and events. For example, you can handle the *ADXBeforeShow* event to initialize your form for the current Inspector window:

C#

```
private void ADXOlForm1_ADXBeforeFormShow()
{
    Outlook.Inspector Inspector = (Outlook.Inspector) InspectorObj;
    Outlook.MailItem Item = (Outlook.MailItem)Inspector.CurrentItem;
    SentLabel.Text = Item.Subject;
}
```

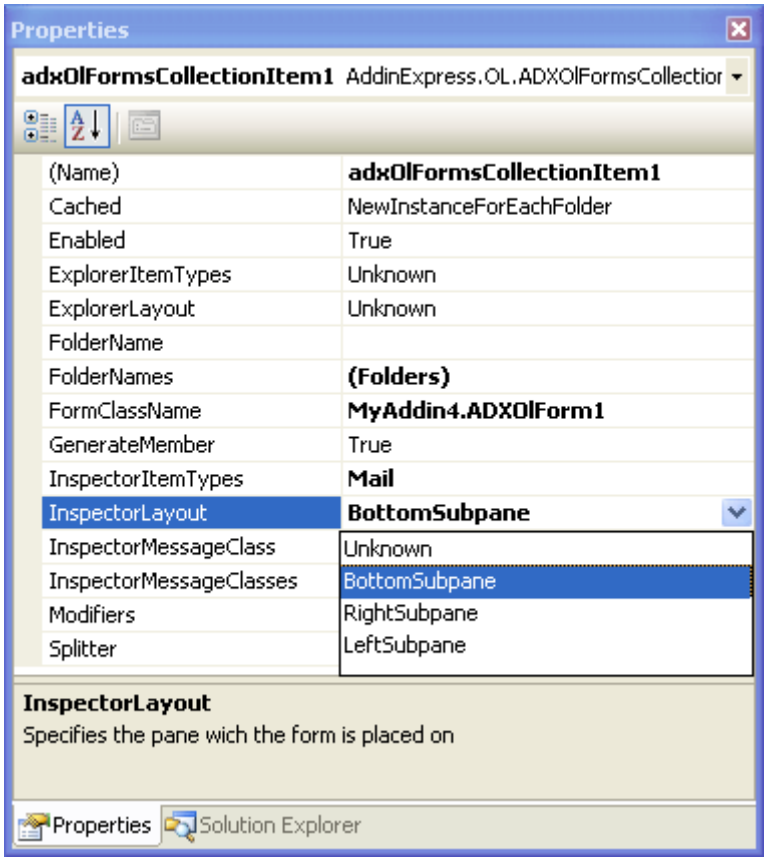
VB.NET

```
Private Sub ADXOlForm1_ADXBeforeShow() Handles Me.ADXBeforeShow
   Dim Inspector As Outlook.Inspector = CType(InspectorObj, Outlook.Inspector)
   Dim Item As Outlook.MailItem = CType(Inspector.CurrentItem, Outlook.MailItem)
   SentLabel.Text = Item.Subject
End Sub
```

5. Binding the form to Outlook forms

To specify the Inspector window which your form is displayed for, you should add a new item to the *Items* collection of the Outlook Forms Manager, select your form class in the *FormClassName* property and specify an Outlook form via three special properties, *InspectorItemTypes*, *InspectorMessageClass* and *InspectorMessageClasses*.

Then, with the *InspectorLayout* property you specify the sub-pane which the form is placed on. InspectorLayout allows you to create left, right or bottom form sub-pane.



6. Run your add-in

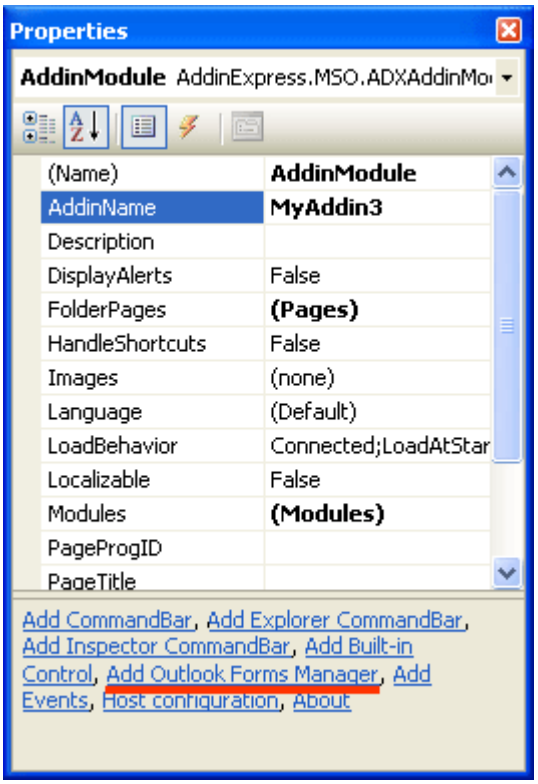
Finally, you rebuild the add-in project, run Outlook and find your form embedded.

Insight of ADX X for Outlook

At first glance, the ADX Extensions for Outlook seems very easy-to-use. It is true if you embed one sub-pane and bind it to one folder. However, you may face some difficulties if your project contains several embedded forms, if your forms are dynamically bound to folders, if two or more forms are bound to one folder, etc.

Outlook Forms Manager

As described above, the ADX Extensions for Outlook adds a new command, "**Add Outlook Forms Manager**", to the add-in module command set,. The command enables the ADX Extensions for Outlook for the current add-in solution, includes all references necessary to support ADX X for Outlook by the current add-in setup project and the add-in shim, and adds a special component, **ADXOlFormsManager** to the add-in module.



Note

- Please note that you should run this command before adding embedded forms.

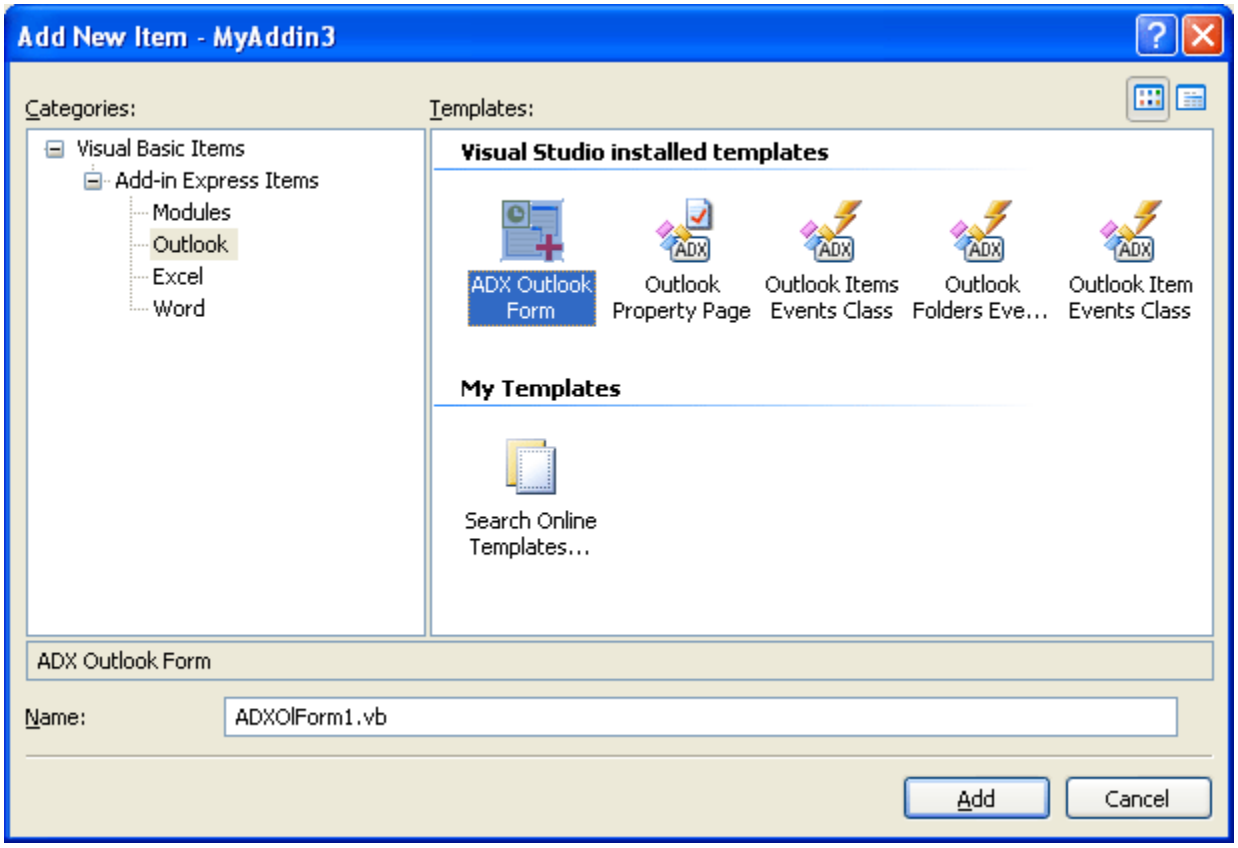
The ADX Extensions for Outlook publishes two components used for embedding your forms into Outlook. The most important component of ADX X for Outlook is the Outlook Forms Manager, an instance of the ADXOlFormsManager class added by the command described above.

The Outlook Forms Manager centralizes and controls your forms and binds them to Outlook folders. The table below shows some properties and events of the component. ADXOlFormsManager is a member of the **AddinExpress.OL.2005** (2003) namespace.

Property, event or method	Description
▪ **Items** collection	An item of the collection binds one specified form to one or several Outlook windows. The form is specified by the FormClassName property of the item, the windows are specified by special properties such as FolderName, FoldersNames and ExplorerItemTypes.
▪ **AddinModule** property	Returns a reference to the add-in module that contains the Outlook Forms Manager component.
▪ **CurrentForm** property	Returns an instance of your form contained in the active Explorer window.
▪ **OutlookAppObj** property	Returns an instance of the Outlook.Application object that hosts your add-in.
▪ **ADXBeforeFolderSwitch** event	Occurs before the Outlook Explorer window goes to a new folder, either as a result of a user action or through the program code. You can use this event to terminate all active processes in the current form embedded into the **current folder view**.
▪ **ADXBeforeFolderSwitchEx** event	Occurs before the Outlook Explorer window goes to a new folder, either as a result of a user action or through the program code. Allows you to temporary disable the creation of your sub-pane for a folder.

ADX Outlook Form

The ADX Extensions for Outlook provides a special form class that can be used for embedding forms into Outlook windows. You can add a new descendant of this class via a special wizard, **ADX Outlook Form**, available through the Add New Item dialog of the add-in project. Please note that you can run the wizard after running the "Add Outlook Forms Manager" command.



The **ADXOlForm** class is a descendant of Windows.Forms.Form. ADXOlForm publishes several Outlook-specific properties and events that can be used to access Outlook objects from an embedded form. The table below describes some properties and events of ADXOlForm. ADXOlForm is a member of the **AddinExpress.OL.2005** (2003) namespace.

Property, event or method	Description
▪ **OutlookAppObj** property	Returns a reference to the instance of Outlook.Application that hosts the add-in.

▪ **ExplorerObj** property	Returns a reference to the instance of the Outlook.Explorer object which your form was embedded into. Returns NULL for form sub-panes.
▪ **InspectorObj** property	Returns a reference to the instance of the Outlook.Inspector object which your form was embedded into. Returns NULL for folder panes and folder sub-panes.
▪ **FolderObj** property	Returns a reference to the instance of Outlook.MAPIFolder in the context of which the form was created. Returns NULL for form sub-panes.
▪ **FolderItemsObj** property	Returns a reference to the Items collection of the FolderObj property. Returns NULL for form sub-panes.
▪ **ADXBeforeFormShow** event	Occurs each time before the form is shown.
▪ **ADXAfterFormShow** event	Occurs each time after the form is shown.
▪ **ADXBeforeInspectorSubpaneClose** event	Occurs each time before its owner is closed.
▪ **ADXSelectionChange** event	Retranslates the SelectionChange event of Outlook for folder panes and folder sub-panes.
▪ **FormsManager** property	Returns a reference to the Outlook Forms Manager of the add-in.
▪ The **Item** property	Returns the item of the Items collection of the Outlook Forms Manager that owns the form.

Binding embedded forms to Outlook windows

To embed your form into Outlook windows you should add a new item to the Items collection of the Outlook Forms Manager and bind it to your form's class and the Outlook Explorer or / and Inspector window via special properties of the added item. All item properties of the Outlook Forms Manager’s Items collection are described in the table below.

Property, event or method	Description
▪ **Cached** property	Defines a caching strategy for your form. Applies to folder sub-panes and folder panes only. Cached strategies are described below.
▪ **Enabled** property	Enables / disable embedding your form.
▪ **ExplorerItemTypes** property	Specifies the types of Outlook folders which your form is embedded into. Applies to folder panes and folder sub-panes.
▪ **ExplorerLayout** property	Specifies the sub-pane that contains your form. Use WebViewPane to create a folder pane, and RightSubpane, TopSubpane or BottomSubpane to create a folder sub-pane. Applies to folder panes and folder sub-panes.
▪ **FolderName, FolderNames** properties	Specify names of the folders which your forms is embedded for. Applies to folder panes and folder sub-panes.
▪ **FormClassName** property	Specifies your form.
▪ **InspectorItemTypes** property	Specifies the types of the built-in Outlook Inspector forms which your form is embedded into. Applies to form sub-panes.
▪ **InspectorLayout** property	Specifies the sub-pane that contains your form. Use LeftSubpane, RightSubpane or BottomSubpane to create a sub-pane of the corresponding form. Applies to form sub-panes.
▪ **InspectorMessageClass, InspectorMessageClasses** property	Specify the message classes of the custom Outlook forms that contain your form. The properties correspond to the MessageClass property in the Outlook Object Model and

	the MAPI property PR_MESSAGE_CLASS. These properties allow you to embed your form into custom Outlook forms specified by their message class. Applies to form sub-panes.
▪ **Splitter** property	Specifies if your form contains a splitter. If you set this property to False, you disable your form resizing.

Note

- Please note that with one item of the Item collection you can embed your form into the Outlook Explorer and Inspector windows simultaneously.

Above we described a very simple example that shows how to bind one form to several folders. Remember that your form is bound to Outlook folders by an item of the Items collection of the Outlook Forms Manager. In addition, ADX X for Outlook allows you to bind one form to several Outlook folders through several items of this collection. For example:

```
VB.NET

  ' Item #1 binds one form to all mail folders
  Me.AdxOlFormsManager1.Items.Add(Me.AdxOlFormsCollectionItem1)
  Me.AdxOlFormsCollectionItem1.ExplorerItemTypes = _
      AddinExpress.OL.ADXOlExplorerItemTypes.olMailItem
  Me.AdxOlFormsCollectionItem1.FormClassName = "ADXOlForm1"

  ' Item #2 binds the same form to the "Personal Folders\Special" folder
  Me.AdxOlFormsManager1.Items.Add(Me.AdxOlFormsCollectionItem1)
  Me.AdxOlFormsCollectionItem1.FolderName = "Personal Folders\Special"
  Me.AdxOlFormsCollectionItem1.FormClassName = "ADXOlForm1"
```

What can you need it for? Using different items binding the same form to Outlook folders you can create logical sets to control the form's behavior, to change bound folders, etc. For example, you can disable or enable your forms separately for all mail folders or for the "Personal Folders\Special" folder. Also, you can change a form's

layout to place your form on the top sub-pane for mail folders and on the bottom sub-pane for the "Personal Folders\Special" folder. Another purpose is to replace the form class name. It gives you a possibility to dynamically and flexibly bind different forms to different folders.

How can you bind your forms to Outlook folders? First of all, you can do it via the Items collection designer of the Outlook Forms Manager, as shown above. If you need to do it by code (on-the-fly), you should handle the **ADXBeforeFolderSwitch** event of the Outlook Forms Manager and create new items here, delete existing items, enable/disable them, bind to other folders, etc. In general, only in the handler of ADXBeforeFolderSwitch you can change the Items collection of the Outlook Forms Manager.

Cached forms

Outlook creates a new instance of its any windows embedded into the Explorer and Inspector windows whenever the user switches between folders. In contrast to this behavior, the ADX Extensions for Outlook can create one instance of the embedded form when the user visits a folder for the first time and disposes this instance when the user closes Outlook. So, we say that ADX X for Outlook caches embedded forms, which allows your add-in to considerably expedite switching between Outlook folders.

The ADX Extensions for Outlook provides two very important built-in caching strategies that you should be aware of. You can enable or disable caching via the Cached property of the item that binds your form to Outlook folders.

NewInstanceForEachFolder

By default, ADX X for Outlook creates one instance of your form for each folder visited by the user in the current Explorer window. This strategy is called NewInstanceForEachFolder. For example, if you bind your form to two folders, two instances will be created if the user visits both folders bound to your form in one Explorer window. If the user opens a second Explorer window and visits two folders in each Explorer window, four instances of your form will be created.

This caching strategy has been developed to give you a possibility to *create unique content for each folder visited by the user*. Please take it into account when developing embedded forms and their initialization and finalization code. Also, you may need to have common data for all instances of your embedded forms and to show the data on each form instance. Then you can use the add-in module accessible from your form via the AddinModule property.

Note

- By default, all forms are cached by the NewInstanceForEachFolder strategy. You can disable or change this via the Cached property of the corresponding collection item of the Outlook Forms Manager.
- If you disable caching for your form, a new instance of your form will be created whenever the user visits each folder bound to your form.
- Any run-time changes of the corresponding item dispose all cached instances.

OneInstanceForAllFolders

Another built-in caching strategy is OneInstanceForAllFolders. It creates one instance of your forms for all folders bound to your forms and shows this instance for all folders in one Explorer window. For example, if you bind one form to all mail folders, one instance will be created if the user visits any number of mail folders in one Explorer window. If the user opens a second Explorer window and visits any mail folders in two Explorer windows, two instances of your form will be created (for each Explorer).

Why are instances of your form created for each Explorer window? In general, MS Windows cannot show one instance (window) on two windows at the same time. So, your forms should have different instances for each Explorer window.

This caching strategy was developed to give you a possibility to *create one form for all folders visited by the user*. Please take it into account when developing embedded forms and their initialization and finalization code.

Note

- Please note that caching strategies is subject to change.

Please note that caching strategies do not apply to form sub-panes. A new instance of your embedded form is created for each form sub-pane when a new Inspector is opened.

Getting started on VCL

The ADX Extensions for Outlook is a visual tool. It is based on the Windows API and comprised of over twenty internal classes. But all the internal classes were designed to provide only two public components. This makes ADX X for Outlook easy-to-use in accordance with the true RAD paradigm. So, the ADX Extensions for Outlook provides a very comfortable way to enhance the GUI of your Outlook add-ins with minimal service coding. You write your applied code only.

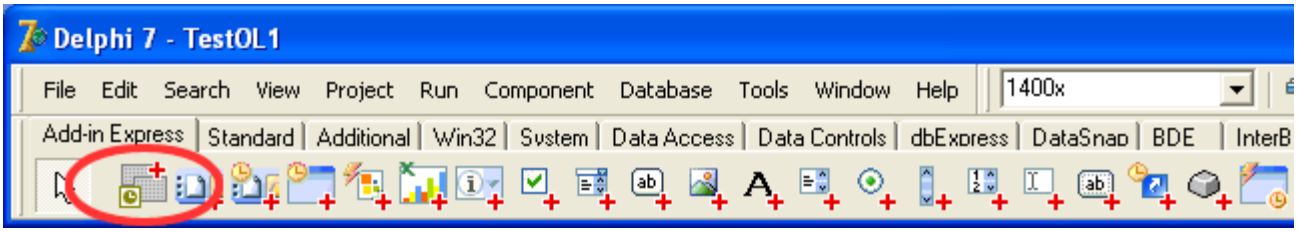
This section shows how you can quickly get started with the ADX Extensions for Outlook in Borland Delphi for Win32 and describes what the ADX Extensions for Outlook adds to your add-ins based on Add-in Express VCL.

What is added to Add-in Express VCL

The ADX Extensions VCL for Outlook is a plug-in for Add-in Express VCL. It adds a new item to the "*New Items*" dialog and a new component, *Outlook Forms Manager*, to the Add-in Express tab on the Component Palette.

Outlook Forms Manager

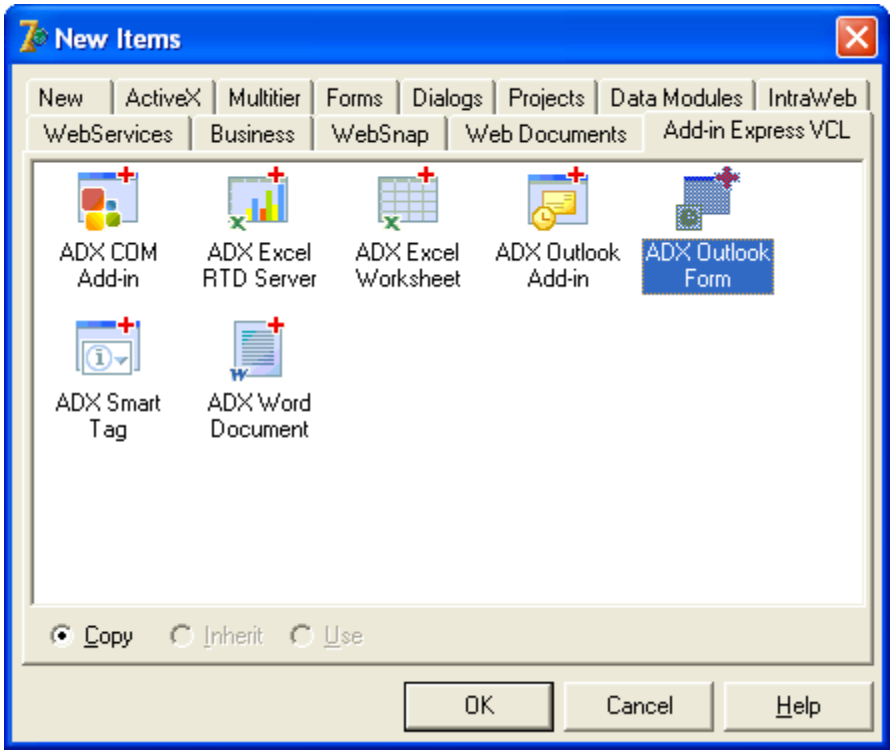
The ADX Extensions for Outlook publishes two components used for embedding your forms into Outlook. One of them is Outlook Forms Manager (*TadxOlFormsManager*) placed on the Add-in Express tab of the Component Palette. Outlook Forms Manager concentrates, centralizes and controls your forms and binds them to Outlook windows. It should be added to your add-in module before creating embedded forms.



New Items dialog box

The ADX Extensions for Outlook provides a special form class that implements a form embedded into the Outlook windows. You can add a new form to your add-in project via a special wizard, **ADX Outlook Form**, available on the Add-in Express tab of the New Items dialog.

The class is called **TadxOlForm** and is a descendant of TForm. In order to be embedded into Outlook windows, all your Outlook forms should be descendants of TadxOlForm. TadxOlForm publishes several Outlook-specific properties and events that can be used to access Outlook objects from an embedded form.



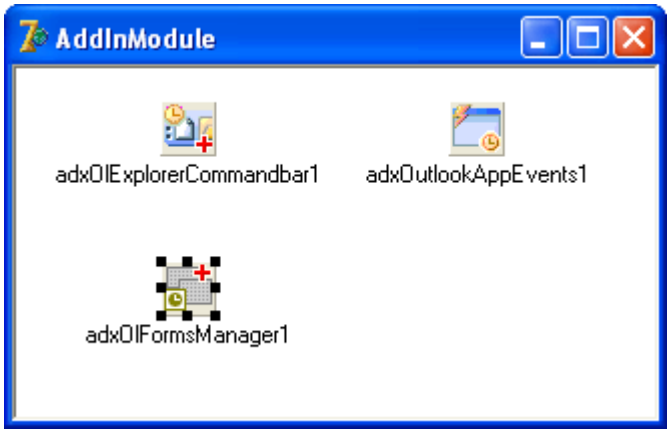
Note

- You can run the wizard after adding the Outlook Forms Manager component to your add-in module.

Your first Folder Sub-pane

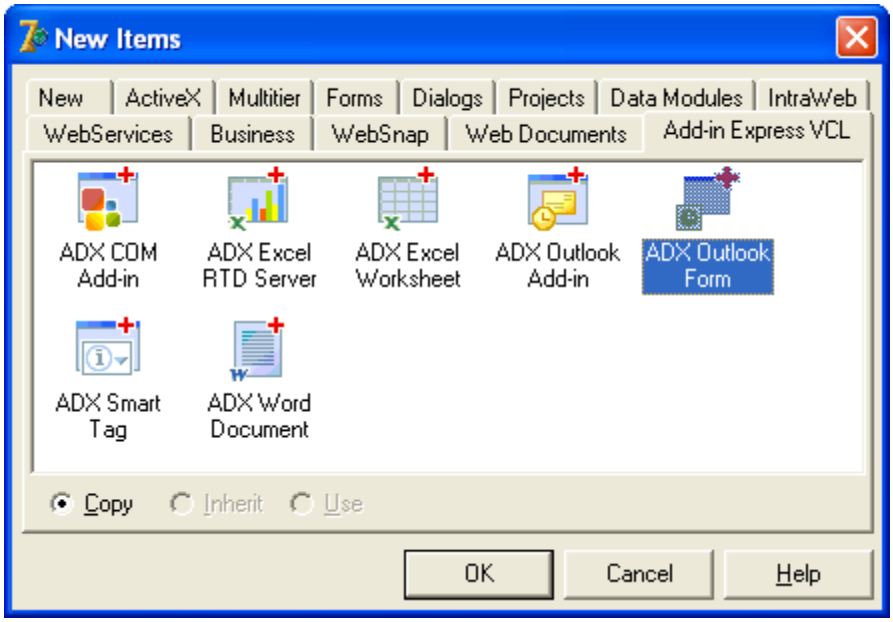
1. Add the Outlook Forms Manager

To include the ADX Extensions for Outlook functionality in your Outlook add-in project you should add the adxOlFormsManager component to your add-in module.



2. Add a new embedded form

Then, run the "**ADX Outlook Form**" wizard from the New Items dialog. The wizard adds a new form to the add-in project.



3. Customize the form

On your form, you can use any controls such as calendars, edit boxes, grids, list views, etc. In this example we added a label to show when the selected message was sent.

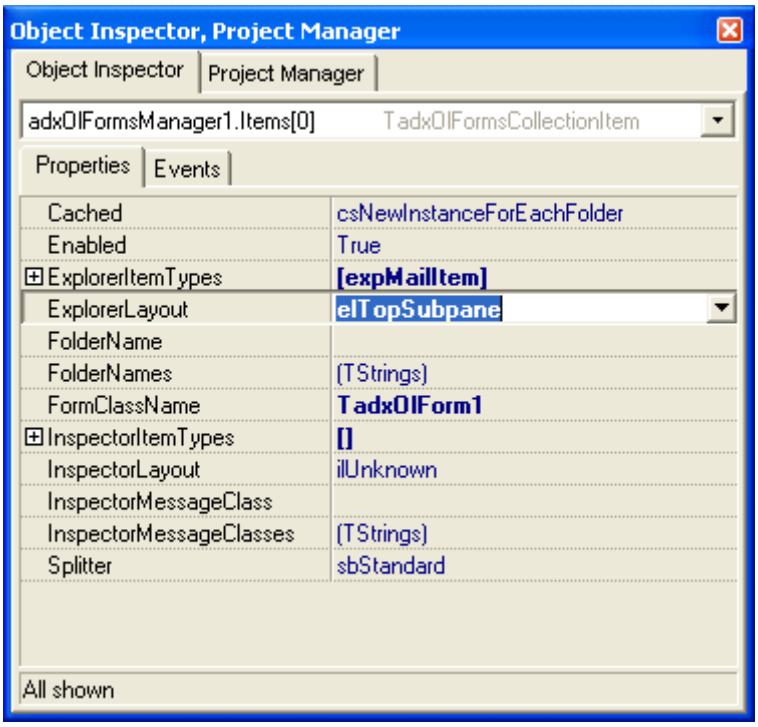
4. Access to Outlook object

ADXOlForm provides access to the base Outlook objects and events. For example, you can handle the *ADXSelectionChange* event of the form to synchronize your form with selection changes in the current Explorer window:

```
procedure TadxOlForm1.adxOlFormADXSelectionChange(Sender: TObject);
var
  Selection: OLEVAriant;
  MailItem: Outlook2000._MailItem;
begin
  Selection := Self.ExplorerObj.Selection;
  if Selection.Count > 0 then begin
    MailItem := Selection.Item[1] as Outlook2000._MailItem;
    SentLabel.Caption := 'Sent: ' + DateTimeToStr(MailItem.SentOn);
  end;
end;
```

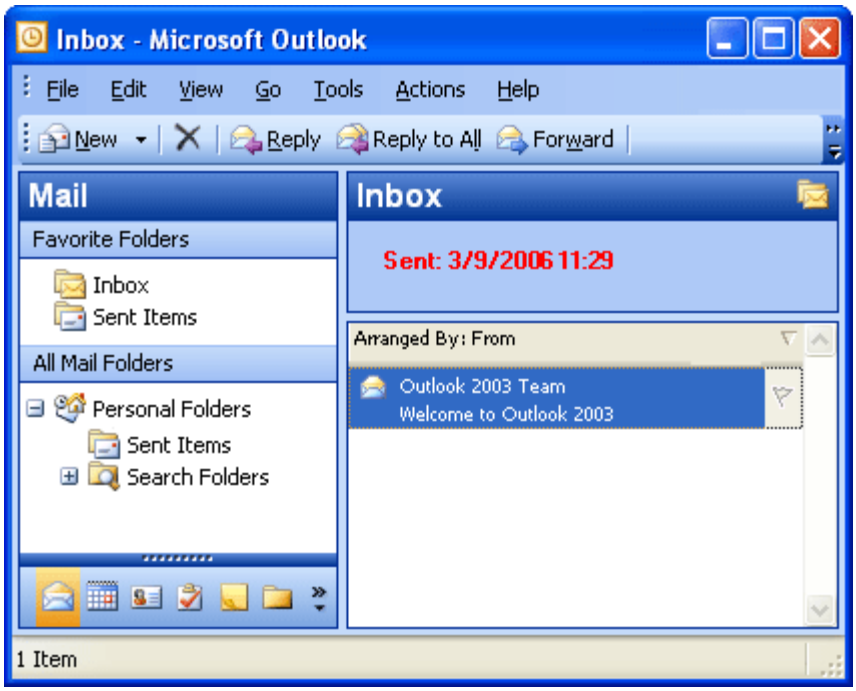
5. Binding the form to Outlook folders

To specify the folders which your form is displayed for, you should add a new item to the *Items* collection of the Outlook Forms Manager, select your form class in the *FormClassName* property and specify the Outlook folder via three special properties, *FolderName*, *FolderNames* and *ExplorerItemTypes*, that are common for all Outlook-related components provided by Add-in Express. Then, with the *ExplorerLayout* property you specify the pane or sub-pane which the form is placed on: *elWebViewPane* that replaces the content of the Explorer window with a folder pane, or *elRightSubpane*, *elTopSubpane* or *elBottomSubpane* that create corresponding folder sub-panes.



6. Run your add-in

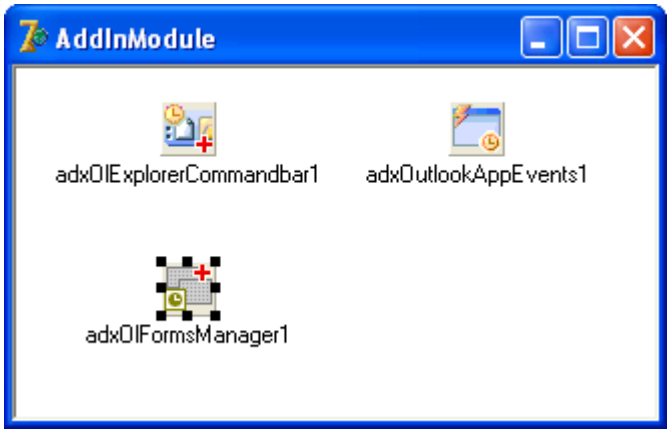
Finally, you rebuild the add-in project, run Outlook and find your form embedded.



Your first Form Sub-pane

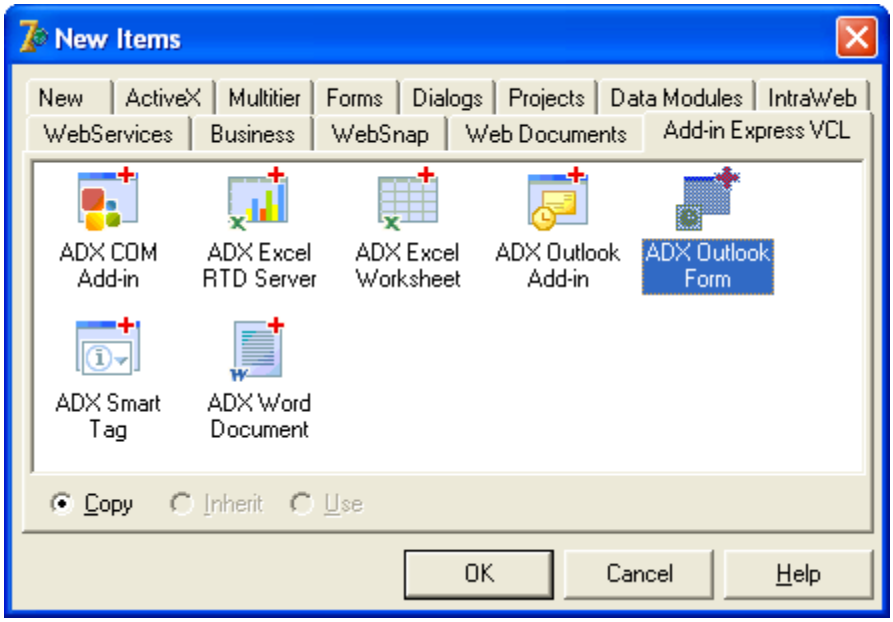
1. Add the Outlook Forms Manager

To include the ADX Extensions for Outlook functionality in your Outlook add-in project, you should add the adxOlFormsManager component to your add-in module.



2. Add a new embedded form

Then, run the "**ADX Outlook Form**" wizard from the New Items dialog. The wizard adds a new form to the add-in project.



3. Customize the form

On your form, you can use any controls such as calendars, edit boxes, grids, list views, etc. In this example we added a label to show when the selected message was sent.

4. Access to Outlook object

ADXOlForm provides access to the base Outlook objects and events. For example, you can handle the ADXBeforeShow event to initialize your form for the current Inspector window:

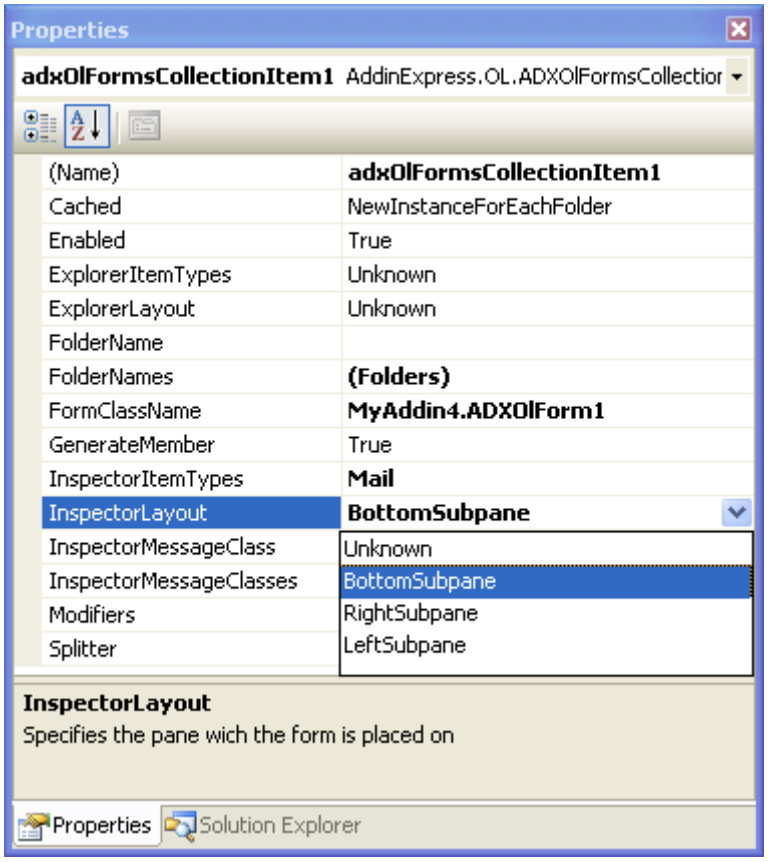
ADXOlForm provides access to the base Outlook objects and events. For example, you can handle the *ADXBeforeShow* event of the form to initialize your form for the current Inspector window:

```
procedure TadxOlForm1.adxOlFormADXBeforeShow (Sender: TObject);
var
  MailItem: OLEVariant;
begin
  MailItem := Self.InspectorObj.CurrentItem;
  SentLabel.Caption := 'Sent: ' + DateTimeToStr(MailItem.SentOn);
end;
```

5. Binding the form to Outlook folders

To specify the Inspector window which your form is displayed for, you should add a new item to the *Items* collection of the Outlook Forms Manager, select your form class in the *FormClassName* property and specify an Outlook form via three special properties, *InspectorItemTypes*, *InspectorMessageClass* and *InspectorMessageClasses*.

Then, with the *InspectorLayout* property you specify the sub-pane which the form is placed on. InspectorLayout allows you to create left, right or bottom form sub-pane.



6. Run your add-in

Finally, you rebuild the add-in project, run Outlook and find your form embedded.

Insight of ADX X for Outlook

At first glance, the ADX Extensions for Outlook seems very easy-to-use. It is true if you embed one sub-pane and bind it to one folder. However, you may face some difficulties if your project contains several embedded forms, if your forms are dynamically bound to folders, if two or more forms are bound to one folder, etc.

Outlook Forms Manager

The ADX Extensions for Outlook publishes two components used for embedding your forms into Outlook. The most important component of ADX X for Outlook is the Outlook Forms Manager, an instance of the TadxOlFormsManager class added by the command described above.

The Outlook Forms Manager centralizes and controls your forms and binds them to Outlook folders. The table below shows some properties and events of the component.

Property, event or method	Description
▪ **Items** collection	An item of the collection binds one specified form to one or several Outlook windows. The form is specified by the FormClassName property of the item, the folders are specified by special properties such as FolderName, FoldersNames and ExplorerItemTypes.
▪ **AddinModule** property	Returns a reference to the add-in module that contains the Outlook Forms Manager component.
▪ **CurrentForm** property	Returns an instance of your form contained in the active Explorer window.
▪ **OutlookAppObj** property	Returns an instance of the Outlook.Application object that hosts your add-in.
▪ **ADXBeforeFolderSwitch** event	Occurs before the Outlook Explorer window goes to a new folder, either as a result of a user action or through the program code. You can use this event to terminate all active processes in the current form embedded into the **current folder view**.
▪ **ADXBeforeFolderSwitchEx** event	Occurs before the Outlook Explorer window goes to a new

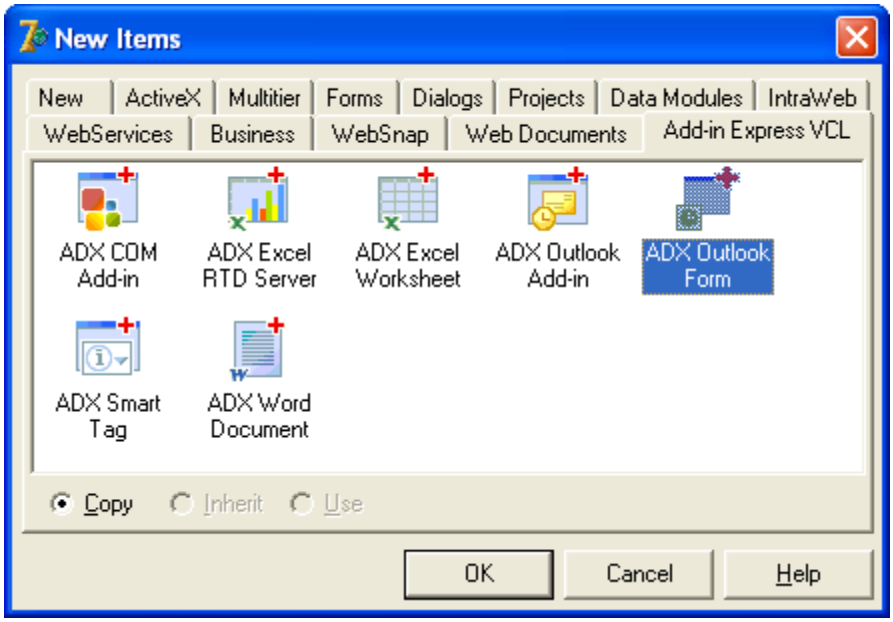
	folder, either as a result of a user action or through the program code. Allows you to temporary disable the creation of your sub-pane for a folder.

ADX Outlook Form

The ADX Extensions for Outlook provides a special form class that can be used for embedding forms into Outlook windows. You can add a new descendant of this class via a special wizard, **ADX Outlook Form**, available through the New Items dialog of the add-in project.

Note

- Please note that you can run the wizard after running the “Add Outlook Forms Manager” command.



The **ADXOlForm** class is a descendant of TForm. ADXOlForm published several Outlook-specific properties and events that can be used to access Outlook objects from an embedded form. The table below describes some properties and events of ADXOlForm.

Property, event or method	Description
▪ **OutlookAppObj** property	Returns a reference to the instance of Outlook.Application that hosts the add-in.
▪ **ExplorerObj** property	Returns a reference to the instance of the Outlook.Explorer object which your form was embedded into. Returns nil for form sub-panes.
▪ **InspectorObj** property	Returns a reference to the instance of the Outlook.Inspector object which your form was embedded into. Returns nil for folder panes and folder sub-panes.
▪ **FolderObj** property	Returns a reference to the instance of Outlook.MAPIFolder in the context of which the form was created. Returns nil for form sub-panes.
▪ **FolderItemsObj** property	Returns a reference to the Items collection of the FolderObj property. Returns nil for form sub-panes.
▪ **ADXBeforeFormShow** event	Occurs each time before the form is shown.
▪ **ADXAfterFormShow** event	Occurs each time after the form is shown.
▪ **ADXBeforeInspectorSubpaneClose** event	Occurs each time before its owner is closed.
▪ **ADXSelectionChange** event	Retranslates the SelectionChange event of Outlook for folder panes and folder sub-panes.
▪ **FormsManager** property	Returns a reference to the Outlook Forms Manager of the add-in.
▪ The **Item** property	Returns the item of the Items collection of the Outlook Forms Manager that owns the form.

Binding embedded forms to Outlook windows

To embed your form into Outlook windows you should add a new item to the Items collection of the Outlook Forms Manager and bind it to your form's class and the Outlook Explorer or / and Inspector window via special

properties of the added item. All item properties of the Outlook Forms Manager's Items collection are described in the table below.

Property, event or method	Description
▪ **Cached** property	Defines a caching strategy for your form. Applies to folder sub-panes and folder panes only. Cached strategies are described below.
▪ **Enabled** property	Enables / disable embedding your form.
▪ **ExplorerItemTypes** property	Specifies the types of Outlook folders which your form is embedded into. Applies to folder panes and folder sub-panes.
▪ **ExplorerLayout** property	Specifies the sub-pane that contains your form. Use WebViewPane to create a folder pane, and RightSubpane, TopSubpane or BottomSubpane to create a folder sub-pane. Applies to folder panes and folder sub-panes.
▪ **FolderName, FolderNames** properties	Specify names of the folders which your forms is embedded for. Applies to folder panes and folder sub-panes.
▪ **FormClassName** property	Specifies your form.
▪ **InspectorItemTypes** property	Specifies the types of the built-in Outlook Inspector forms which your form is embedded into. Applies to form sub-panes.
▪ **InspectorLayout** property	Specifies the sub-pane that contains your form. Use LeftSubpane, RightSubpane or BottomSubpane to create a sub-pane of the corresponding form. Applies to form sub-panes.
▪ **InspectorMessageClass, InspectorMessageClasses** property	Specify the message classes of the custom Outlook forms that contain your form. The properties correspond to the MessageClass property in the Outlook Object Model and

	the MAPI property PR_MESSAGE_CLASS. These properties allow you to embed your form into custom Outlook forms specified by their message class. Applies to form sub-panes.
▪ **Splitter** property	Specifies if your form contains a splitter. If you set this property to False, you disable your form resizing.

Note

- Please note that with one item of the Item collection you can embed your form into the Outlook Explorer and Inspector windows simultaneously.

Binding techniques

Above we described a very simple example that shows how to bind one form to several folders. Remember that your form is bound to Outlook folders by an item of the Items collection of the Outlook Forms Manager. In addition, ADX X for Outlook allows you to bind one form to several Outlook folders through several items of this collection. For example:

What can you need it for? Using different items binding the same form to Outlook folders you can create logical sets to control the form's behavior, to change bound folders, etc. For example, you can disable or enable your forms separately for all mail folders or for the "Personal Folders\Special" folder. Also, you can change a form's layout to place your form on the top sub-pane for mail folders and on the bottom sub-pane for the "Personal Folders\Special" folder. Another purpose is to replace the form class name. It gives you a possibility to dynamically and flexibly bind different forms to different folders.

How can you bind your forms to Outlook folders? First of all, you can do it via the Items collection designer of the Outlook Forms Manager, as shown above. If you need to do it by code (on-the-fly), you should handle the **ADXBeforeFolderSwitch** event of the Outlook Forms Manager and create new items here, delete existing items, enable/disable them, bind to other folders, etc. In general, only in the handler of ADXBeforeFolderSwitch you can change the Items collection of the Outlook Forms Manager.

Cached forms

Outlook creates a new instance of its any windows embedded into the Explorer and Inspector windows whenever the user switches between folders. In contrast to this behavior, the ADX Extensions for Outlook can create one instance of the embedded form when the user visits a folder for the first time and disposes this instance when the user closes Outlook. So, we say that ADX X for Outlook caches embedded forms, which allows your add-in to considerably expedite switching between Outlook folders.

The ADX Extensions for Outlook provides two very important built-in caching strategies that you should be aware of. You can enable or disable caching via the Cached property of the item that binds your form to Outlook folders.

NewInstanceForEachFolder

By default, ADX X for Outlook creates one instance of your form for each folder visited by the user in the current Explorer window. This strategy is called NewInstanceForEachFolder. For example, if you bind your form to two folders, two instances will be created if the user visits both folders bound to your form in one Explorer window. If the user opens a second Explorer window and visits two folders in each Explorer window, four instances of your form will be created.

This caching strategy has been developed to give you a possibility to *create unique content for each folder visited by the user*. Please take it into account when developing embedded forms and their initialization and finalization code. Also, you may need to have common data for all instances of your embedded forms and to show the data on each form instance. Then you can use the add-in module accessible from your form via the AddinModule property.

Note

- By default, all forms are cached by the NewInstanceForEachFolder strategy. You can disable or change this via the Cached property of the corresponding collection item of the Outlook Forms Manager.
- If you disable caching for your form, a new instance of your form will be created whenever the user visits each folder bound to your form.
- Any run-time changes of the corresponding item dispose all cached instances.

OneInstanceForAllFolders

Another built-in caching strategy is OneInstanceForAllFolders. It creates one instance of your forms for all folders bound to your forms and shows this instance for all folders in one Explorer window. For example, if you

bind one form to all mail folders, one instance will be created if the user visits any number of mail folders in one Explorer window. If the user opens a second Explorer window and visits any mail folders in two Explorer windows, two instances of your form will be created (for each Explorer).

Why are instances of your form created for each Explorer window? In general, MS Windows cannot show one instance (window) on two windows at the same time. So, your forms should have different instances for each Explorer window.

This caching strategy was developed to give you a possibility to *create one form for all folders visited by the user*. Please take it into account when developing embedded forms and their initialization and finalization code.

Note

- Please note that caching strategies is subject to change.

Please note that caching strategies do not apply to form sub-panes. A new instance of your embedded form is created for each form sub-pane when a new Inspector is opened.