Official Guide

Add-in Express™ VCL

Getting Started and Class Reference

Add-in Express, VCL Edition

Add-in Express™ VCL

Document version **2.5**
Revised at **28-Jul-06**
Product version **2.x**

Content

Introduction 13

What's This 14

Technical Support 14

Add-in Express Web Site 14

Internet E-mail 14

Technical Support via Instant Messengers 15

System Requirements 15

Supported Delphi Versions 15

Host Applications for COM Add-ins 15
Excel Versions for Automation Add-ins 16
Excel Versions for Real-Time Data Servers 16
Host Applications for Smart Tags 16
About This Document 16
Getting Started 17
Your First Add-in 18
Step #1 – Run a COM add-in specific wizard 18
Step #2 – Specify the add-in project properties 19
Step #3 – Review the add-in project 19
Step #4 – Customize the add-in module 21
Step #5 – Add a new command bar 22
Step #6 – Add a new button 23
Step #7 – Register and run the add-in 25
Several Notes 26
Terminology 26
How to get access to host apps 26
Command bars changing 27
Debugging add-ins 27
What are the OfficeTag property and the Tag property 27
Pop-ups 27
Edit and combo boxes and the OnChange event 28
Attention 28
Outlook Add-ins 29
Step #1 – Run the Outlook specific wizard 29
Step #2 – Add option pages 30
Step #3 – Customize option pages 31
Step #4 – Add new command bars 32
Step #5 – Register and run the add-in 33
Handling Built-in Controls 33
Step #1 – Run a COM add-in specific wizard 34
Step #2 – Customize the add-in module 35
Step #3 – Add the adxBuiltInControl component 36
Step #4 – Set up the BuiltInID property 36
Step #5 – enabled or disable the standard action 37
Step #6 – Handle the OnAction event 38

Smart Tag Libraries 39
Step #1 – Run the Smart Tag specific wizard 39
Step #2 – Review the smart tag library project 39
Step #3 – Customize the smart tag module 42
Step #4 – Add a new smart tag component 42
Step #5 – Add a new smart tag actions 43
Step #6 – Register and run the smart tag library 45
Excel Real-Time Data Servers 46
Step #1 – Run the RTD Server specific wizard 47
Step #2 – Name your RTD server 48
Step #2 – Add a new topic 49
Step #3 – Register and run an add-in 50
Excel Automation Add-ins 51
Step #1 – Run the COM add-in specific wizard 51
Step #2 – Customize the add-in 52
Step #4 – Add a new function to the type library 53
Step #5 – Register and run the add-in 54
Class Reference 55
Add-in Express Units 56
Constants 57
adxLCID 57
adxVersion 57
adxOfficeAppNames 57
Types 57
TadxControlType 57
TadxHostAppSet, TadxOfficeHostApp 58
TadxMsoBarPosition 58
TadxMsoBarProtection 58
TadxMsoBarType 58
TadxMsoButtonState 58
TadxMsoButtonStyle 59
TadxMsoComboStyle 59
TadxMsoCommandBarButtonHyperlinkType 59
TadxOLItemClasses, TadxOLItemClass 59
TadxOLItemTypes, TadxOLItemType 60
TadxOLItemTypeAction 60

TadxStartMode 60
TadxKindType 60
Classes 62
TadxAddIn 62
COMAddInModule Property 62
Factory Property 62
TadxBuiltInControl 63
CommandBar Property 63
DefaultInterface Property 63
DisableStandardAction Property 64
BuiltInID Property 64
OfficeTag Property 64
Owner Property 64
OnAction Event 64
TadxCOMAddInModule 66
AccessApp Property 68
AddInInstance Property 68
AddInName Property 68
COMAddInClassFactory Property 68
COMAddInClassInstance Property 69
CommandBars Property 69
CommandBarsCount Property 69
Description Property 69
DisplayAlerts Property 69
ExcelApp Property 70
FrontPageApp Property 70
HostApp Property 70
HostType Property 70
LoadBehavior Property 71
MapPointApp Property 71
OutlookApp Property 71
PowerPointApp Property 72
ProjectApp Property 72
RegistryKey Property 72
Registry Property 72
StartMode Property 73

SupportedApps Property 73
VisioApp Property 73
WordApp Property 74
XLAutomationAddIn Property 74
AddMapPointCommand Method....74
CommandBarByName Method....76
CommandBarIndexOf Method....76
FindCommandBar Method....76
FindControl Method 76
NameSpaceOptionsPagesAdd Method....77
OptionsPagesAdd Method....77
OnAddInBeginShutdown Event 78
OnAddInFinalize Event....78
OnAddInInitialize Event 78
OnAddInStartupComplete Event 78
OnAfterAddinRegister Event....79
OnBeforeAddinUnregister Event 79
OnError Event....79
OnOLXXX Events 80
TadxCommandBar 81
AdaptiveMenu Property 82
BuiltIn Property 82
CommandBarLeft Property 82
CommandBarName, CommandBarNameLocal Properties....82
CommandBarTop Property....83
Context Property....83
Controls Property....83
DefaultInterface Property....84
Enabled Property 84
Height Property....84
Index Property....84
Owner Property....84
Parent Property....85
Position Property....85
Protection Property 85
SupportedApps Property 85

RowIndex Property 86
Temporary Property 86
Type_ Property 86
Visible Property 87
Width Property 87
ControlByCaption Method 87
ControlByOfficeTag Method 87
ControlByTag Method 88
TadxCommandBarButton 89
BuiltInFace Property 89
DefaultInterface Property 90
DisableStandardAction Property 90
FaceID Property 90
Glyph Property 90
GlyphTransparentColor Property 91
HyperlinkType Property 91
ShortcutText Property 91
State Property 91
Style Property 91
OnClick Event 92
TadxCommandBarComboBox 93
DropdownLines Property 93
DropdownWidth Property 94
Items Property 94
List Property 94
ListCount Property 94
ListHeaderCount Property 94
ListIndex Property 95
Style Property 95
Text Property 95
OnChange Event 96
TadxCommandBarControl 97
AfterID Property 98
AsButton Property 98
AsComboBox Property 99
AsDropdownList Property 99

AsEdit Property 99
AsPopup Property 99
Before Property 100
BeforeID Property 100
BeginGroup Property 100
BuiltIn Property 100
Caption Property 101
CommandBar Property 101
DefaultInterface Property 101
DescriptionText Property 101
Enabled Property 101
Height Property 102
HelpContextID Property 102
HelpFile Property 102
IsPriorityDropped Property 102
Left Property 103
OfficeID Property 103
OfficeIndex Property 103
OfficeTag Property 103
OLEUsage Property 104
OlExplorerItemTypes Property 104
OlInspectorItemTypes Property 104
OlItemTypeAction Property 104
OnAction Property 105
Parameter Property 105
Parent Property 106
Priority Property 106
Tag Property 106
Temporary Property 106
TooltipText Property 106
Top Property 107
Type_ Property 107
Visible Property 108
Width Property 108
TadxCommandBarControls 109
Items Property 109

Add Method....109
DeleteControl Method....110
ItemByTag Method....110
TadxCommandBarDropdownList....111
DropdownLines Property....111
DropdownWidth Property....112
Items Property....112
List Property....112
ListCount Property....112
ListHeaderCount Property....113
ListIndex Property....113
Style Property....113
Text Property....113
OnChange Event....114
TadxCommandBarEdit....115
Style Property....115
Text Property....115
OnChange Event....115
TadxCommandBarPopup....117
DefaultInterface Property....117
Controls Property....117
OfficeID Property....118
ControlByCaption Method....118
ControlByOfficeTag Method....118
ControlByTag Method....118
TadxCustomCommandBarComboBox....120
DefaultInterface Property....120
TadxFactory....121
FilePath Property....121
RegistryKey Property....121
TadxOlExplorerCommandbar....122
FolderName Property....122
FolderNames Property....122
ItemTypes Property....123
TadxOlInspectorCommandbar....124
FolderName Property....124

FolderNames Property....124
ItemTypes Property....125
TadxRTDFactory....126
FilePath Property....126
TadxRTDServer....127
Factory Property....127
RTDServerModule Property....127
TadxRTDTopic....128
DefaultValue Property....129
Enabled Property....129
Text Property....129
TopicID Property....130
String01..String28 Properties....130
Strings Property....130
UseStoredValue Property....130
OnDisconnectTopic Event....131
OnRefreshData Event....131
TadxXLRTDServerModule....132
Interval Property....132
UpdateTopics Method....132
OnError Event....133
OnRTDFinalize Event....133
OnRTDInitialize Event....133
OnConnectData Event....133
TadxRecognizerFactory....134
FilePath Property....134
TadxActionFactory....135
TadxRecognizerObject....136
Factory Property....136
TadxActionObject....137
Factory Property....137
TadxSmartTagAction....138
Caption Property....138
Captions Property....138
Name Property....139
Tag Property....139

OnClick Event 139
TadxSmartTagActions 141
Items Property 141
TadxSmartTag 142
Actions Property 142
Caption Property 143
Captions Property 143
DownloadURL Property 143
Kind Property 143
RecognizedMask Property 144
RecognizedWords Property 144
OnActionNeeded Event 144
OnPropertyPage Event 144
OnRecognize Event 145
TadxSmartTagModule 147
NamespaceURI Property 147
SmartTagDesc Property 148
SmartTagDescs Property 148
SmartTagName Property 148
SmartTagNames Property 148
OnActionInitialize Event 148
OnActionFinalize Event 149
OnInitialize Event 149
OnFinalize Event 150
OnError Event 150
End User License Agreement ..... Error! Bookmark not defined.

Introduction

Microsoft gives us a way to extend and improve her applications using several special technologies. Office 2000 started IDTExensibility2 for creating COM add-ins. With Office XP Microsoft published the smart tag technology to provide context sensitivity into her applications and the RTD Server technology to replace archaic DDE by a modern solution. As a result, we have several powerful and effective approaches to embed our applied code to 10 Microsoft Office applications.

What's This

In the beginning Add-in Express was developed as a VCL library to simplify creating COM add-ins. It hided all straits of low-level programming while implementing the IDTExtensibility2 interface and provided GUI management including the creation of command bars and command bar controls. At that time Add-in Express supported COM add-ins for the main Office applications, namely Outlook, Word, Excel, PowerPoint and Access. Later we also supported FrontPage and Project by our customers' requests. After Office XP had been published we added smart tags and Excel real-time data servers to give our customers these additional technologies.

At last, we redesigned our Add-in Express to provide the RAD way for creating COM add-ins, smart tags and RTD servers, added one more technology, Excel Automation add-ins, and supported MapPoint, Visio and Publisher. Finally, we ported Add-in Express to .NET. With all this we got Add-in Express 2, the Add-in Express that is described here, in this document.

Now Add-in Express is a VCL and .NET library that will help you to develop COM add-ins, smart tags, Excel RTD servers and Excel Automation add-ins. It provides the shortest way to enhance Microsoft end-user applications and allows you to concentrate your efforts on the applied code.

Technical Support

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

Add-in Express Web Site

The Add-in Express web sire at www.add-in-express.com provides a wealth of information and software downloads for Add-in Express developers, including:

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

Internet E-mail

For technical support through Internet, e-mail us at:

- vcl@add-in-express.com - Add-in Express VCL Edition staff
- net@add-in-express.com - Add-in Express .NET Edition stuff
- support@add-in-express.com - General questions.

Technical Support via Instant Messengers

If you are a subscriber of our Premium Support Service and you need help immediately you can request technical support via an instant messenger, e.g. Windows/MSN Messenger or ICQ. Please ask us at http://www.add-in-express.com/premium-area/contact-us.php.

System Requirements

Add-in Express supports all applications from MS Office family. You can use Add-in Express on Borland Delphi 5 and higher with "Microsoft Office 2000 Sample Automation Wrapper Components" installed. All supported applications and Delphi versions are listed below.

Supported Delphi Versions

- Delphi 5 Architect with Update Pack 1
- Delphi 5 Enterprise with Update Pack 1
- Delphi 5 Professional with Update Pack 1
- Delphi 6 Architect with Update Pack 2
- Delphi 6 Enterprise with Update Pack 2
- Delphi 6 Professional with Update Pack 2
- Delphi 7 Architect with Update Pack 1
- Delphi 7 Enterprise with Update Pack 1
- Delphi 7 Professional with Update Pack 1
- Delphi 2005 for VCL Architect with Update 1
- Delphi 2005 for VCL Enterprise with Update 1
- Delphi 2005 for VCL Professional with Update 1

NOTE. You should have Microsoft Office 2000 Sample Automation Server Wrapper Components installed.

Host Applications for COM Add-ins

- Microsoft Excel 2000 and higher
- Microsoft Outlook 2000 and higher

- Microsoft Word 2000 and higher
- Microsoft FrontPage 2000 and higher
- Microsoft PowerPoint 2000 and higher
- Microsoft Access 2000 and higher
- Microsoft Project 2000 and higher
- Microsoft MapPoint 2002 and higher
- Microsoft Visio 2002 and higher
- Microsoft Publisher 2002 and higher

Excel Versions for Automation Add-ins

- Microsoft Excel 2002 and higher

Excel Versions for Real-Time Data Servers

- Microsoft Excel 2002 and higher

Host Applications for Smart Tags

- Microsoft Excel 2002 and higher
- Microsoft Word 2002 and higher
- Microsoft PowerPoint 2003 and higher

About This Document

Here, in the Getting Started part, we describe Add-in Express step-by-step each project supported by our product and give you the complete instruction how to develop COM add-ins, Outlook add-ins, smart tags, Excel RTD servers and Excel Automation add-ins. Also, we provide the complete guide of Add-in Express types and classes, the Add-in Express Class Reference part.

To aid you in learning how to use Add-in Express, we included several examples in the installation package. All the examples that you see in this tutorial are collected in sample projects that can be found in the QDemo subfolder of the Add-in Express installation folder.

Getting Started

This chapter describes how to use it for developing COM add-ins, RTD servers and smart tags and. Also you can find here several comments about using Add-in Express.

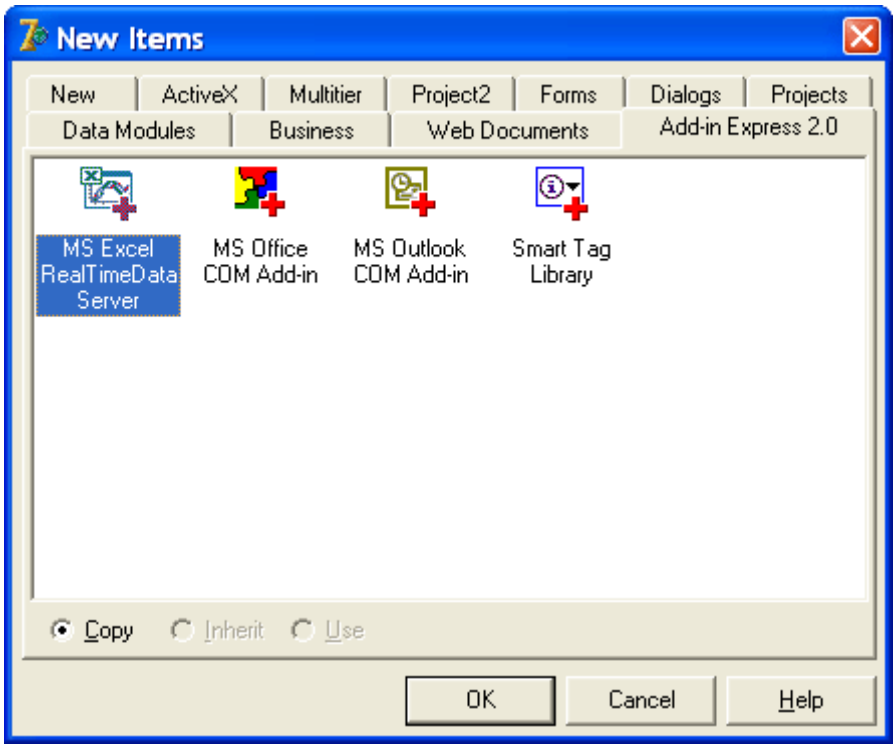
Your First Add-in

COM development in Borland Delphi is not a visual process. But Add-in Express allows you to avoid this and makes add-in development easier. How can this be achieved? With ease. Let's go!

Step #1 – Run a COM add-in specific wizard

Add-in Express adds to Borland Delphi IDE several add-in specific wizards. All these wizards are available on the "Add-in Express 2.0" tab of the "New Item" window.

To create a new COM add-in project close the projects you have opened, choose "File | New | Others", select the "Add-in Express 2.0" tab and double-click the "MS Office COM Add-in" icon. This will start the "MS Office COM Add-in" wizard.

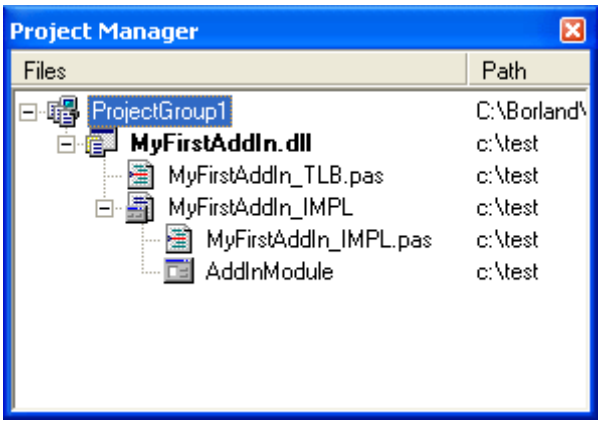


Step #2 – Specify the add-in project properties

With the wizard you specify the name of the add-in project, the destination folder, and the name of the add-in coclass, and click “Next” and “Finish”.

Step #3 – Review the add-in project

When the wizard is finished, the add-in project will be created and opened in Delphi IDE.



The add-in project includes the following modules:

- The project source files (ProjectName.*);
- The type library files: binary (ProjectName.tlb) and Object Pascal unit (ProjectName_TLB.pas);
- The add-in module (ProjectName_IMPL.pas and ProjectName_IMPL.dfm).

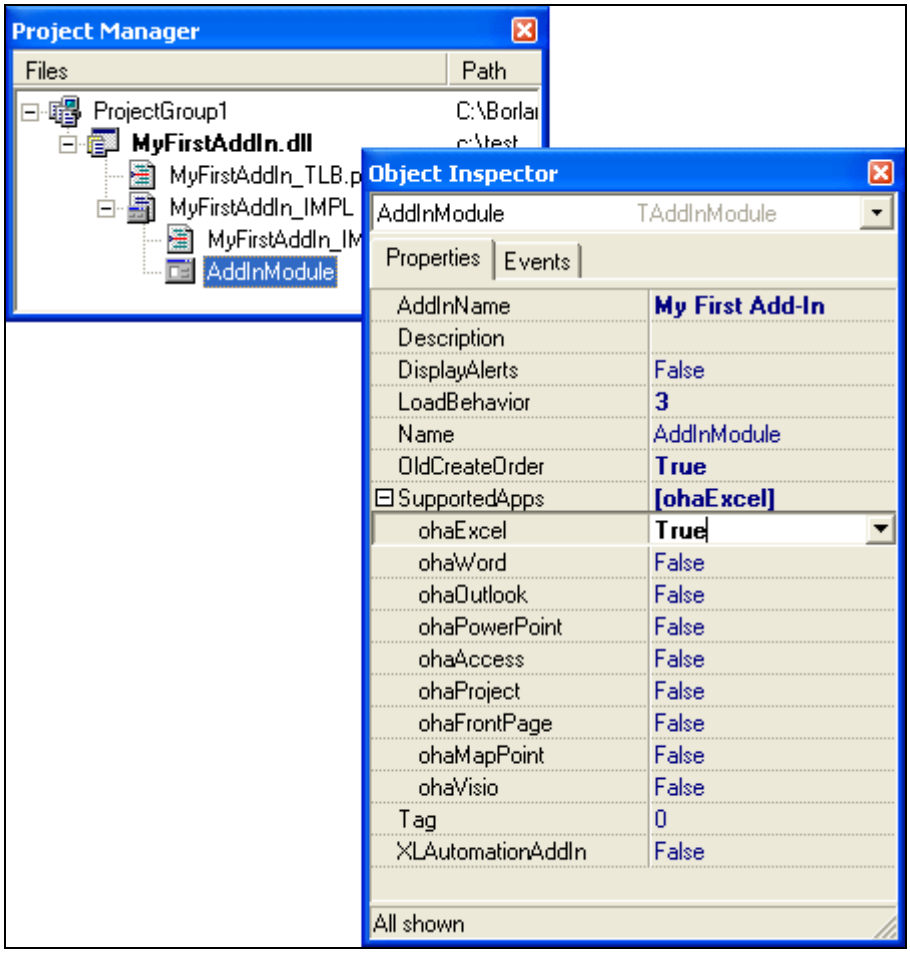
Among all the modules described above only the add-in module is of importance for you. In this example (as well as in your future projects) this unit contains the following code:

```
unit MyFirstAddIn_IMPL;

interface

uses
  SysUtils, ComObj, ComServ, ActiveX, Variants, adxAddIn,
  MyFirstAddIn_TLB, StdVcl;

type
  TMyAddIn = class(TadxAddin, IMyAddIn)
  end;

  TAddInModule = class(TadxCOMAddInModule)
  private
  protected
  public
  end;

implementation

{$R *.dfm}

initialization
  TadxFactory.Create(ComServer, TMyAddIn, CLASS_MyAddIn, TAddInModule);
end.
```

The add-in module contains two classes: the “interfaced” class (TMyAddin in this case) and the add-in module class (TAddInModule). The “interfaced” class is a descendant of the TadxAddIn class that implements the add-in specific interface required by the add-in architecture (IDTExtensibility2 for COM add-ins). The TadxAddIn class being complete, most often you needn’t add any methods to this class. The add-in module class implements the add-in functionality to be released. The add-in module is an analogue to Data Module, but unlike Data Module, the add-in module allows you to set all the necessary properties of your add-in, to handle its events, to create toolbars and controls. This is exactly what we are going to do now.

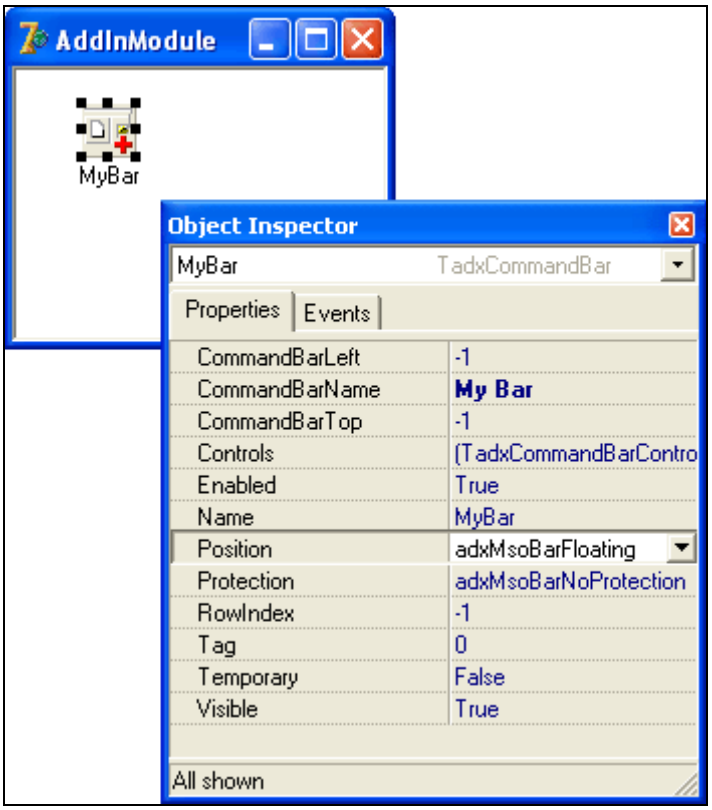
Step #4 – Customize the add-in module

In the Project Manager window, select the add-in module, activate Object Inspector, name your add-in via the AddInName property (this name will appear in the COM add-ins dialog box of the host applications), and select the host applications you need in the SupportedApps property.



Step #5 – Add a new command bar

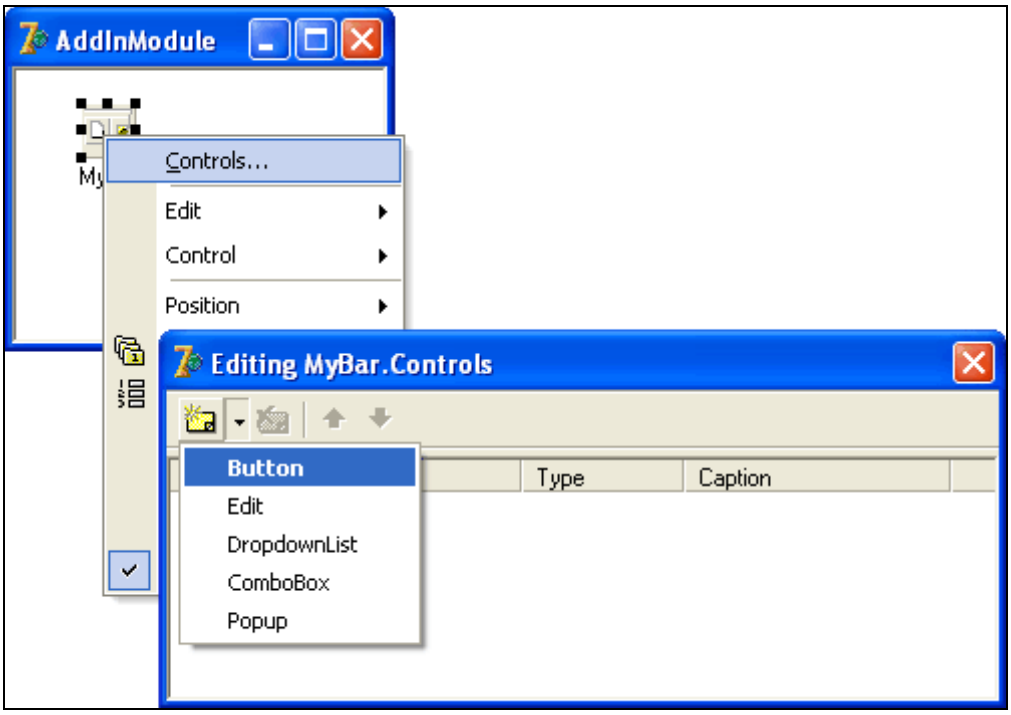
To add a new command bar, select the Add-in Express tab on the Component Palette and add adxCommandBar to the add-in module. In Object Inspector, name the toolbar via the CommandBarName property and select its position via the Position property.

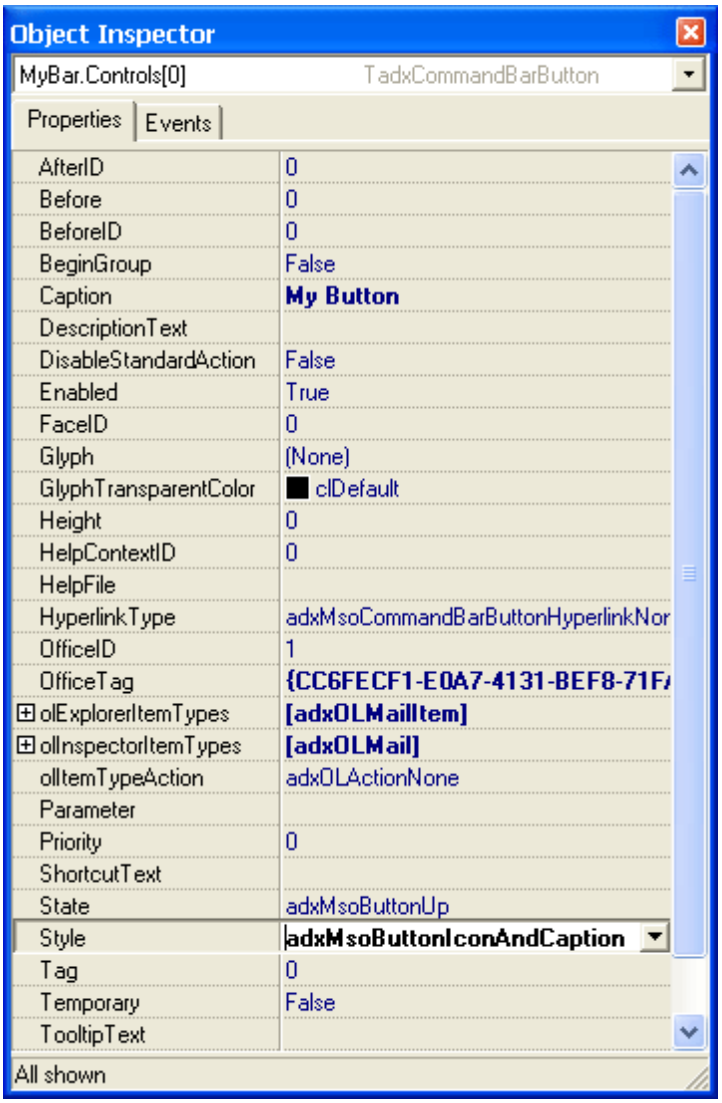


Step #6 – Add a new button

To add a new button, right-click the toolbar component and select the Controls item on the popup menu. In the Editing window, select the Button item on the Add New popup button.

In the Editing window, select the added button, activate Object Inspector, name the button via the Caption property, add the button glyph, and set the adxMsoButtonIconAndCaption style via the Style property.



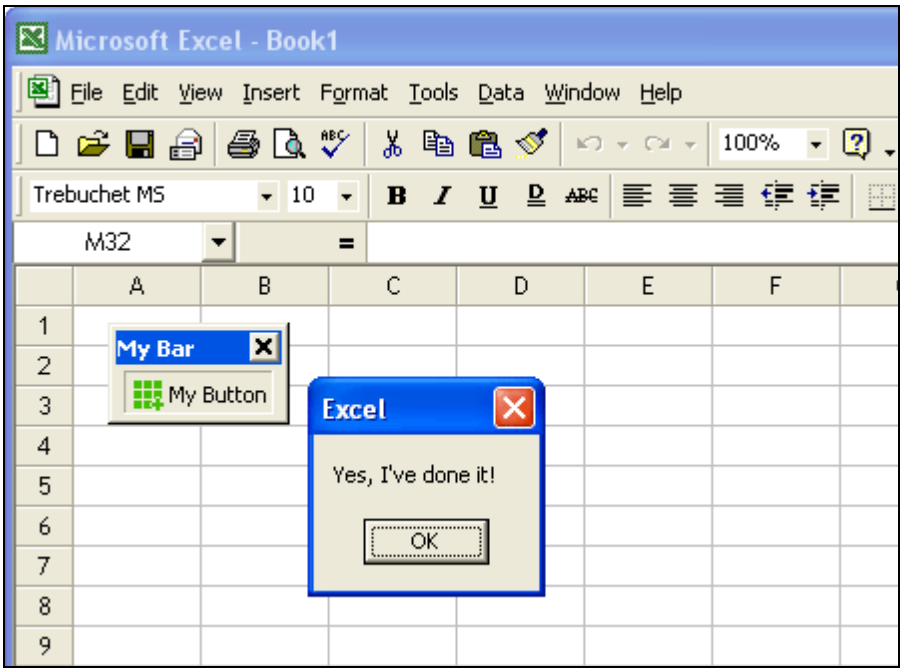


To handle the added button click, select the Events tab on Object Inspector, double click the OnClick event, and enter the code you need. For example:

```
procedure TAddInModule.MyBarControls0Click(Sender: TObject);
begin
  ShowMessage('Yes, I'#39've done it!');
end;
```

Step #7 – Register and run the add-in

Save the add-in project, compile it, close all applications that you have selected as add-in host applications, and register the add-in via “Run | Register ActiveX Server”. Run one of the selected host applications, find your toolbar and click the button.



Several Notes

You might have an impression that creating add-ins is a very simple task. Please don't get too enthusiastic. Yes, Add-in Express makes embedding your code to Office applications very simple, but you should write your applied code yourself, and I guess it would be something more exquisite than a single call of ShowMessage.

Here we describe some important issues you will encounter when developing your add-ins.

Terminology

In this document, on our site (add-in-express.com), and in all our txt files we use the terminology suggested by Microsoft for all toolbars, their controls, and for all interfaces of Office Type Library. For example:

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

Add-in Express uses interfaces from Office Type Library. We cannot describe them here. Please refer to the VBAOFF9.CHM (Office 2000) file and to Office Type Library.

How to get access to host apps

The previous example is intended for demonstration purposes only. We created it to let you get acquainted with the general principles of Add-in Express. In your future add-ins you may want to get access to the host applications objects. Add-in Express allows access to them through the XxxApp property of the add-in module. For example, you can access an active Excel list by calling ExcelApp.ActiveSheet or you can access an active explorer by calling OutlookApp.ActiveExplorer.

Unfortunately, we can't describe here in more or less detailed way how to work with all interfaces of the MS Office applications. The reason is simple - there are too many of them (you can count yourself: Excel, Word, Outlook, PowerPoint, Access, Project, FrontPage, MapPoint, Visio), while we have limited work force. So we recommend you to use the VBA online help of Office applications. And don't forget about early binding and imported type libraries that are used by Add-in Express.

Note. Our Primary subscribers can get help from us about any host application object models.

Command bars changing

Unlike its previous version, Add-in Express 2 tracks changes you have made in all controls and command bars at design-time and updates them automatically. With Add-in Express 1 you had to unregister/register your add-in each time you changed control properties (Caption, for example). Now you needn't do it.

But there may occur the situation when your changes are not shown. For example, you change a picture on your button, but the changes are not reflected on the host command bar. In this case, firstly, let us know and secondly, use the old "unregister/register" way.

Debugging add-ins

Unfortunately, the Delphi debugger is not able to debug DLLs when the path to these files is too long. To debug your add-ins, just indicate the root of one of the logical drives as the destination path (the Output Directory parameter) in the Project Options window. And remember to indicate the host application in "Run | Parameters".

What are the OfficeTag property and the Tag property

Here we address people who are already acquainted with the first version of Add-in Express.

Add-in Express identifies all its controls (command bar controls) through the use of the OfficeTag property (the Tag property of the CommandBarControl interface). The value of this property is generated automatically and in most cases you don't need to change it. But if you need to change it, don't hesitate.

In version 2 all real Delphi amateurs can again enjoy the Tag property, which now has the same value as the other components.

Pop-ups

According to the Microsoft's terminology, the term "popup" can be used for several controls: popup menu, popup button, and submenu. With Add-in Express you can crate your own popup as an element of your controls command bar collection and add to it any control via the Controls property.

But pop-ups have a feature that is very annoying: if an edit box or a combo box is added to a popup, their events are fired very oddly. Don't regard this bug as that of ADX. It seems to be intended by MS.

Edit and combo boxes and the OnChange event

The OnChange event appears only when the value was changed and the focus was shifted. This is also not our bug but a MS guys' "trick".

Attention

If your questions are not answered here, please see the HOWTOs pages on www.add-in-express.com. We are constantly extending these pages. This document also contains Add-in Express Class Reference, where all Add-in Express classes are described in details.

Outlook Add-ins

COM add-ins for Outlook provide two of the capabilities developers request most often: extending menus and toolbars at the application level; extending property pages for folders and the main Options page. That is why Add-in Express provides a special wizard and several Outlook specific classes that allow using these features.

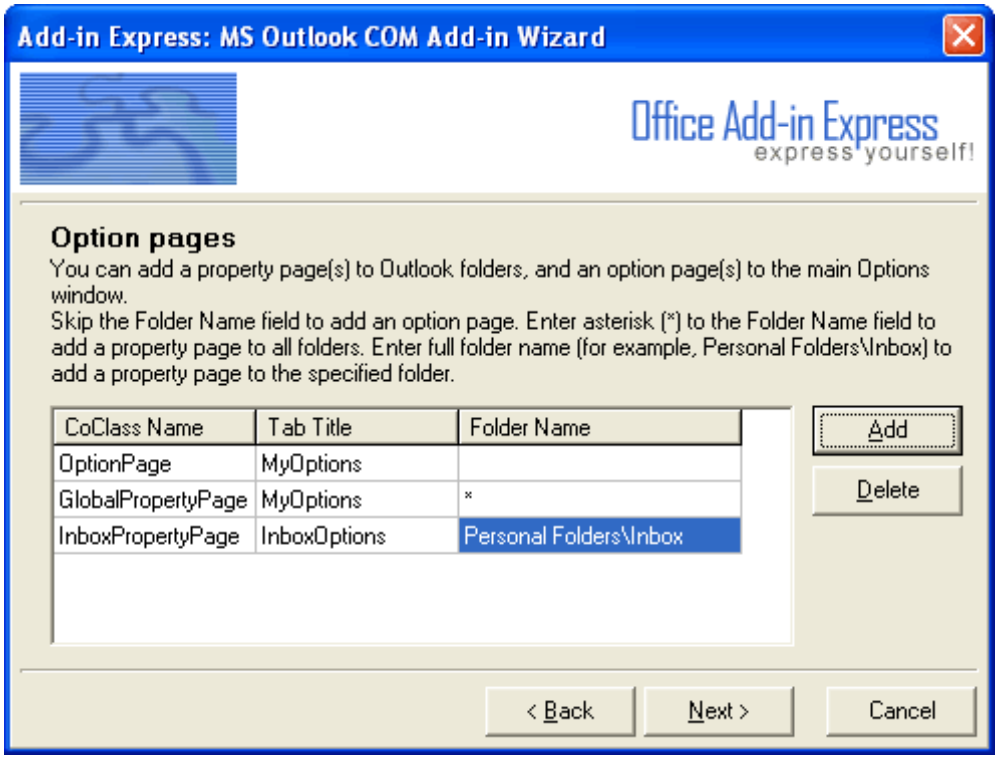
Note. These components only add several Outlook specific features to the main components of Add-in Express. If you don't need option pages and your add-in adds only one toolbar to Outlook Explorer, you may skip the examples described below.

Step #1 – Run the Outlook specific wizard

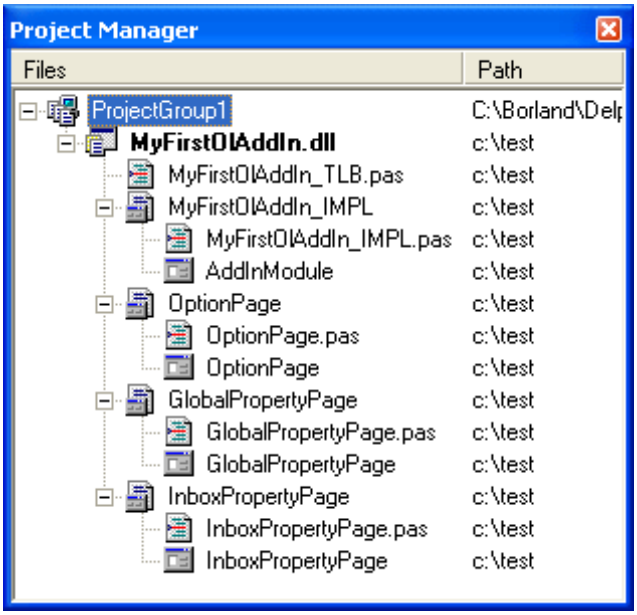
To create an Outlook COM add-in project close all opened projects, choose "File | New | Others" ("File | New" for Delphi 5), select the "Add-in Express 2.0" tab in the New Items window and double-click the "MS Outlook COM add-in" icon. This will start the "MS Outlook COM add-in" wizard. Enter your project name, the add-in coclass name and the destination folder for the add-in project, and click Next.

Step #2 – Add option pages

Outlook allows adding custom option pages to the Options dialog box ("Tools | Options") or to the Properties dialog box of any folder. Add-in Express simplifies the creation of option pages. You can add a new option page(s) through the "MS Outlook COM add-in" wizard.



The wizard generates the add-in project, adds the defined option pages as active forms and opens the project in Delphi IDE.



Step #3 – Customize option pages

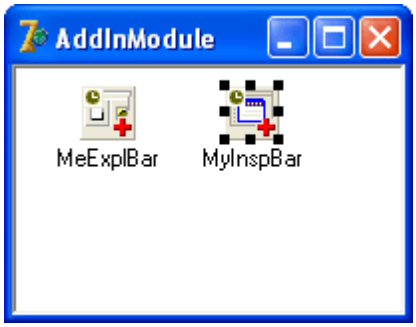
To the option page, add the controls you need and handle their events. The property page form and the option page form are a special edition of Delphi Active Form and implement all the logic needed to support Outlook option pages. By default, an option page contains two controls only: a label and an edit box.

The edit box shows how to catch the OnChange event and how to update the corresponding option page. For example:

```
procedure TMyPropPage.Edit1Change(Sender: TObject);
begin
  GetPropertyPageSite;
  // TODO - put your code here
  UpdatePropertyPageSite;
end;
```

Step #4 – Add new command bars

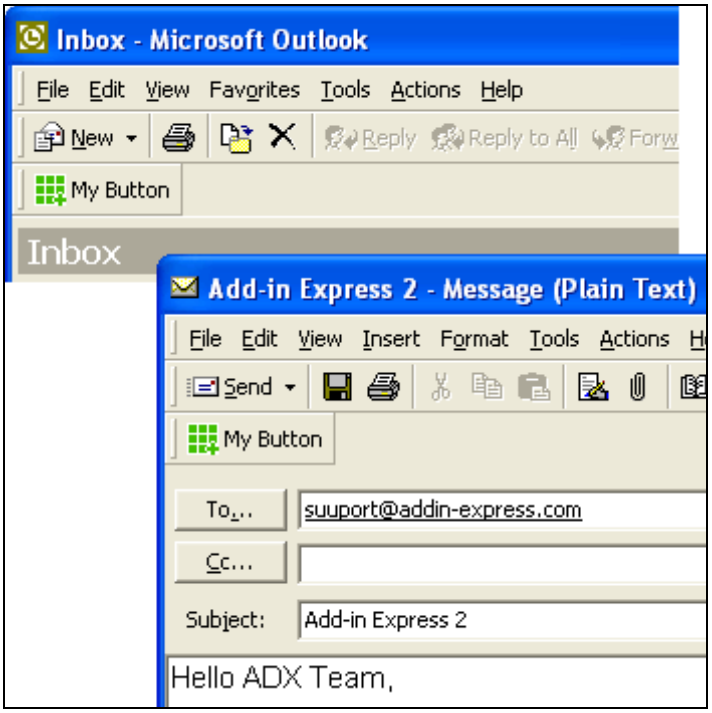
Add-in Express provides two Outlook specific command bars: adxOlExplorerCommandBar and adxOlInspectorCommandBar.



The first adds a command bar to the Outlook Explorer windows and solves a lot of problems with the Outlook command bars system. The second adds a command bar to the Outlook Inspector window. Both can be tuned on for specified folders or item types via the Folder property and the ItemTypes property. To add a new command bar, just add to the add-in module one of the components described above, customize it, and add command bar controls.

Step #5 – Register and run the add-in

Save the add-in project, compile it, close Outlook, and register the add-in via “Run | Register ActiveX Server”. Run Outlook and find your option page and command bar.



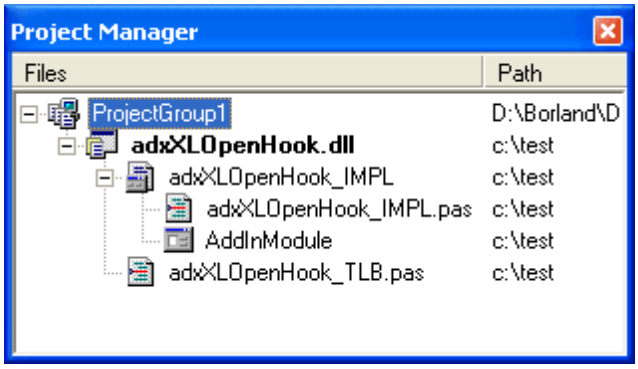
Handling Built-in Controls

Add-in Express allows you to “hook” all built-in controls of the host application including menu items, buttons, edit boxes, etc. For example, we will hook the Open button of Excel.

Step #1 – Run a COM add-in specific wizard

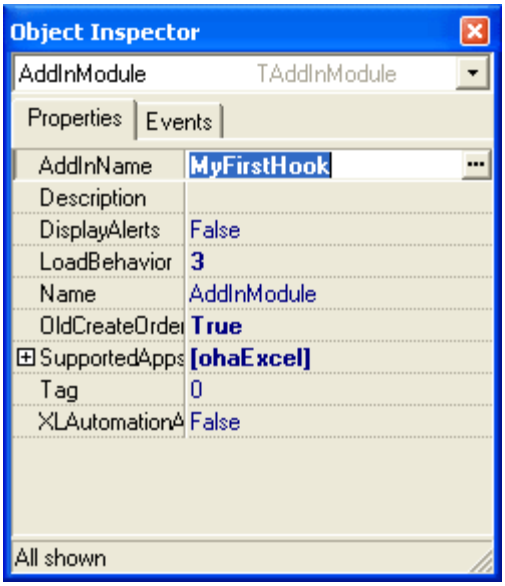
In Delphi IDE, close all opened projects, select the “File | New | Others…” item on the main menu, and run the “MS Office COM add-in” wizard on the “Add-in Express 2.0” tab of the “New Item” dialog box. Enter the name of the project, the add-in coclass name and the destination folder for the project, and click Next and then click Finish. The wizard generates a new project and opens it in Delphi IDE.

When the wizard is finished, the add-in project will be created and opened in Delphi IDE.



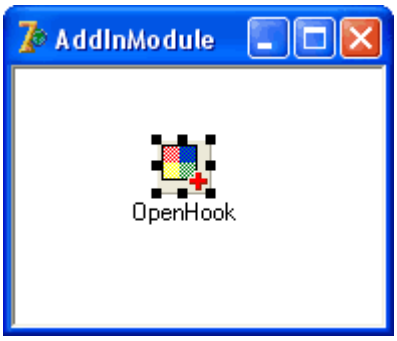
Step #2 – Customize the add-in module

In the Project Manager window, select the add-in module, activate Object Inspector, name your add-in via the AddInName property (this name will appear in the COM add-ins dialog box of the host applications), select Excel as a host application, and set True to the XLAutomationAddIn property.



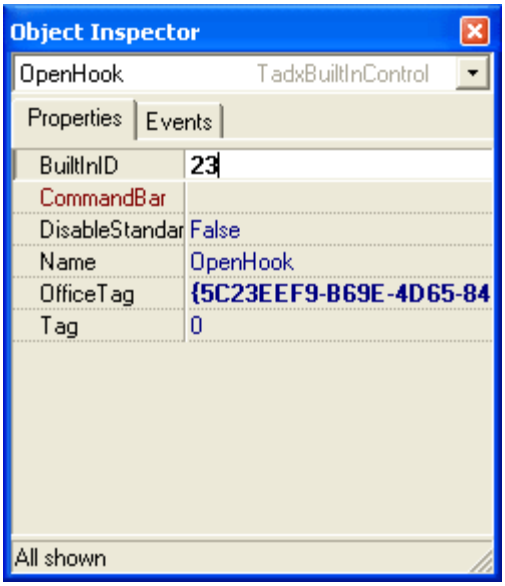
Step #3 – Add the adxBuiltInControl component

Add the adxBuiltInControl component to the add-in module.



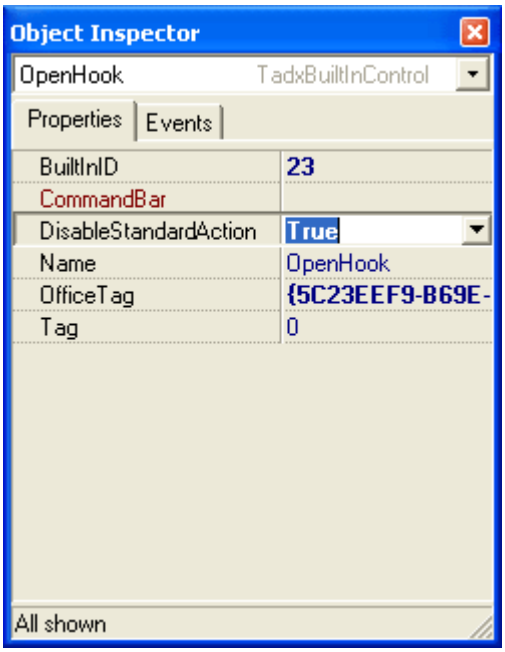
Step #4 – Set up the BuiltInID property

Set the standard ID of the control you need. To get this ID you can use our Standard Controls Scanner (see the Download section on the Add-in Express website).



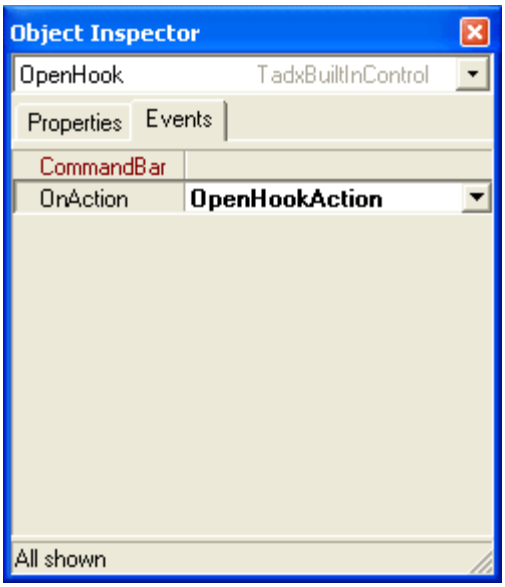
Step #5 – Enabled or disable the standard action

Disable the standard action of the control via the DisableStandardAction property.



Step #6 – Handle the OnAction event

Handle the OnAction event:




```
procedure TAddInModule.OpenHookAction(
  Sender: TObject);
begin
  ShowMessage('I''ve done this!');
end;
```

Smart Tag Libraries

Smart Tags are a technology introduced in Office XP and 2003 that provides Office users with more interactivity for the content of their Office documents. A smart tag is an element of text in an Office document that is recognized as having custom actions associated with it. An example of one of these special elements of text might be an e-mail name that is typed into a Word document or an Excel workbook. If the e-mail name is recognized as a smart tag, the user is presented with one or more actions to perform on the specified text. Possible actions associated with an e-mail name are to look up additional contact information, or send a new e-mail message to that contact.

Add-in Express allows you to create a new smart tag via a couple of clicks. But Add-in Express provides a different way including terminology. A smart tag solution based on Add-in Express implements one smart tag recognizer, but we call this solution the smart tag library. The point is that Add-in Express architecture allows you to create several smart tags within one solution and one recognizer. That is why we regard smart tag solutions based on ADX as smart tag libraries, not one smart tag.

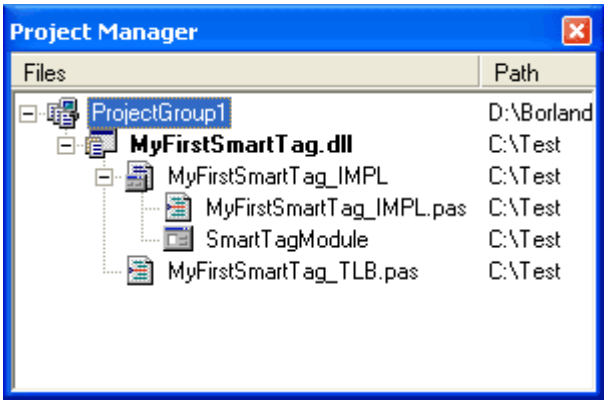
Step #1 – Run the Smart Tag specific wizard

In Delphi IDE, close all opened projects, select the “File | New | Others…” item on the main menu, and run the “Smart Tag Library” wizard on the “Add-in Express 2.0” tab of the “New Item” dialog box.

Enter the name of the project, the coclass name and the destination folder for the project, click Next and then click Finish. The wizard generates a new smart tag project and opens it in Delphi IDE.

Step #2 – Review the smart tag library project

When the wizard is finished, the smart tag project will be created and opened in Delphi IDE. The smart tag project includes the following modules: the project source files (ProjectName.*), the type library files: binary (ProjectName.tlb) and Object Pascal unit (ProjectName_TLB.pas), the smart tag module (ProjectName_IMPL.pas and ProjectName_IMPL.dfm).



Among all the modules described above only the smart tag module is of importance for you. In this example (as well as in your future projects) this unit contains the following code:

```
unit MyFirstSmartTag_IMPL;

interface

uses
  SysUtils, ComObj, ComServ, ActiveX, Variants, adxSmartTag,
  MyFirstSmartTag_TLB, StdVcl;

type

  TFirstSmartTagRecognizer=class(TadxRecognizerObject, IFirstSmartTagRecognizer)
  protected
  end;

  TFirstSmartTagAction = class(TadxActionObject, IFirstSmartTagAction)
  protected
  end;

  TSmartTagModule = class(TadxSmartTagModule)
  private
  protected
  public
  end;
```

```
implementation

{$R *.dfm}

initialization

  TadxRecognizerFactory.Create(ComServer,
    TFirstSmartTagRecognizer,
    CLASS_FirstSmartTagRecognizer,
    TSmartTagModule);

  TadxActionFactory.Create(ComServer,
    TFirstSmartTagAction,
    CLASS_FirstSmartTagAction,
    TSmartTagModule);

end.
```

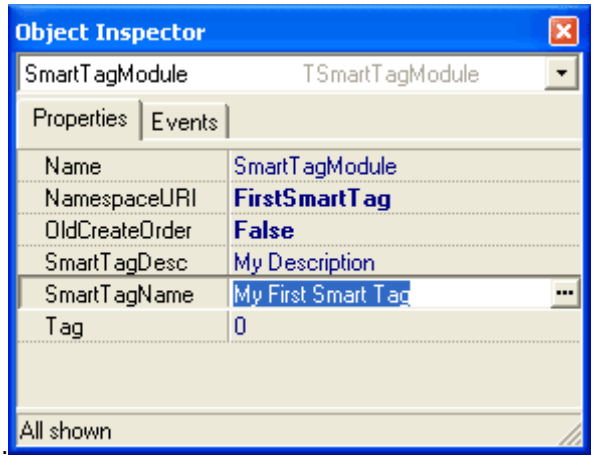
The smart tag module contains three classes:

- The “interfaced” classes (TFirstSmartTagRecognizer and TFirstSmartTagAction);
- The smart tag module class (TSmartTagModule).

The “interfaced” classes are descendants of the TadxRecognizerObject class and the TadxActionObject class that implement the smart tag specific interfaces required by the smart tag architecture: ISmartTagRecognizer, ISmartTagRecognizer2, ISmartTagAction and ISmartTagAction2. These classes being complete, most often you needn’t add or override any methods to them. The smart tag module class implements the smart tag functionality to be released. The smart tag module is an analogue of Data Module, but unlike Data Module, the smart tag module allows you to set all the necessary properties of your smart tags. This is exactly what we are going to do now.

Step #3 – Customize the smart tag module

In the Project Manager window, select the smart tag module, activate Object Inspector, name your smart tag via the SmartTagName property (this name appears in the Smart Tags tab on the host application AutoCorrect Options dialog box), and enter the description of the smart tag via the SmartTagDesc property. These properties depend on Locale.

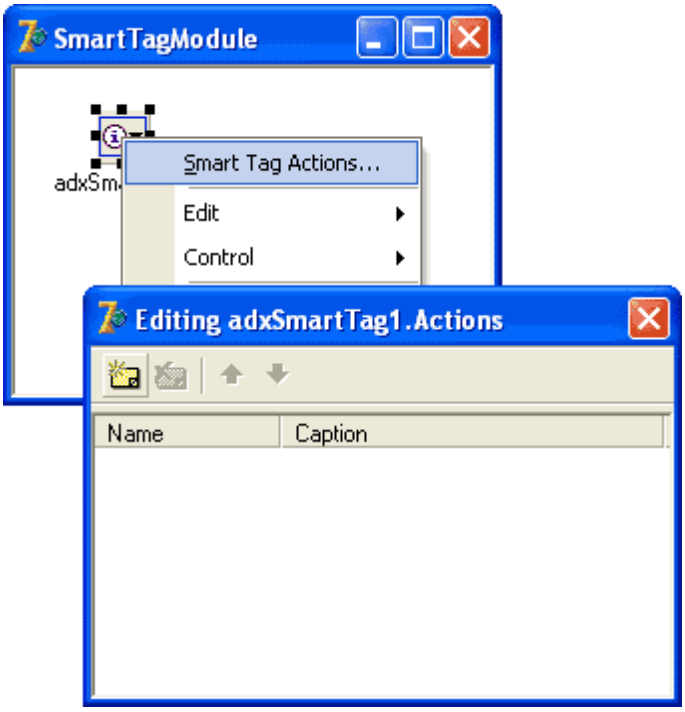


Step #4 – Add a new smart tag component

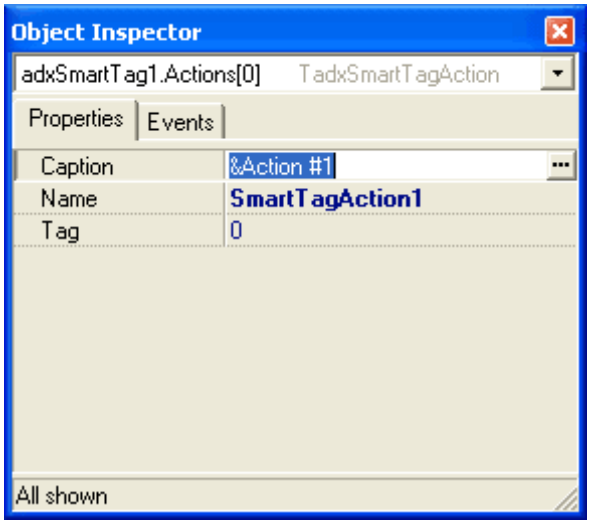
To add a new smart tag, select the Add-in Express tab on the Component Palette and add adxSmartTag to the smart tag module. In Object Inspector, specify the smart tag caption via the Caption property and specify the phrase(s) (via RecognizedWords property) that will be recognized by your smart tag. The smart tag caption will be shown as a caption of the smart tag context menu (pop-up).

Step #5 – Add a new smart tag actions

To add a new smart tag action, right-click the added smart tag component, select the Smart Tag Actions item on the pop-up menu, and in the Editing window click the Add New button.



In the Editing window, select the added action, activate Object Inspector, and specify the action caption via the Caption property. The Caption property depends on Locale. The value of the Caption property will be shown as an item of the smart tag context menu (pop-up).

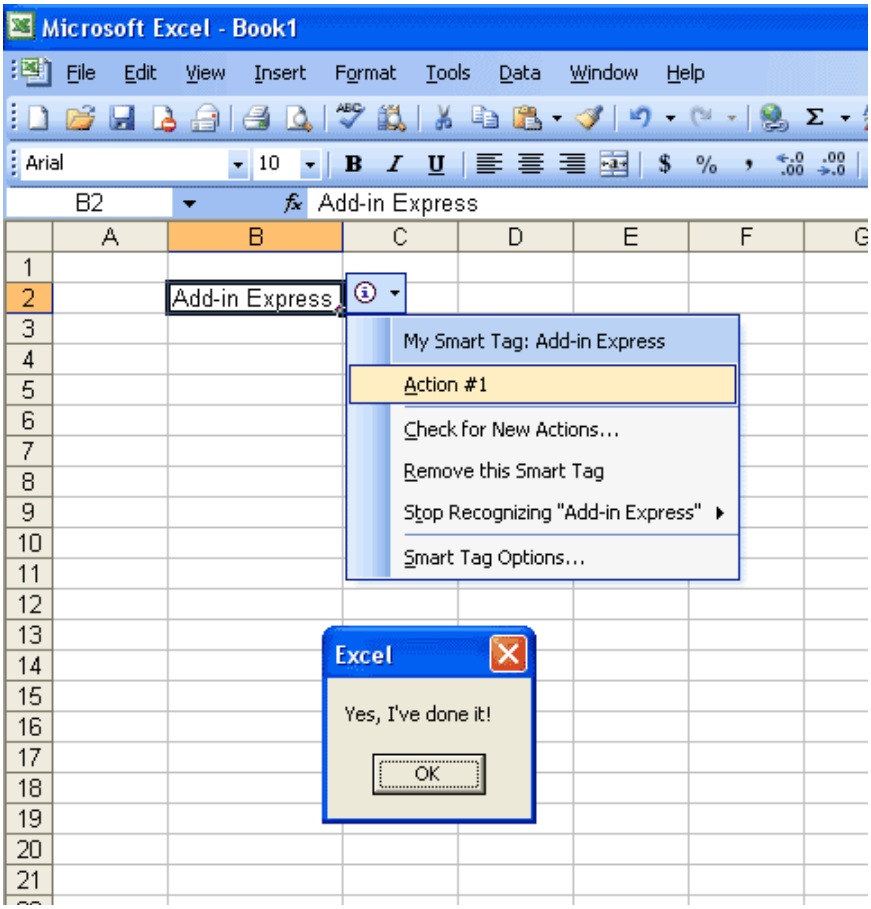


To handle the click event of this menu item, select the Events tab on Object Inspector, double click the OnClick event, and enter the code you need. For example:

```
procedure TSmartTagModule.adxSmartTag1Actions0Click(Sender: TObject;
  const AppName: WideString; const Target: IDispatch;
  const Text, Xml: WideString; LocaleID: Integer);
begin
  ShowMessage('Yes, I'#39've done it!');
end;
```

Step #6 – Register and run the smart tag library

Save the project, compile it, and register the smart tag library via “Run | Register ActiveX Server”. Run MS Excel or MS Word, type one of the smart tag phrases you specified, and click the menu item.



Excel Real-Time Data Servers

Microsoft Excel 2002 and higher provides a new way to view and update data in real time. This real-time data (RTD) feature is great for working with constantly-changing data such as stock quotes, currency exchange rates, inventory levels, price quotes, weather information, sports scores, and so on.

Before we begin, you should be familiar with the notion of a real-time data source, an RTD server, and a topic.

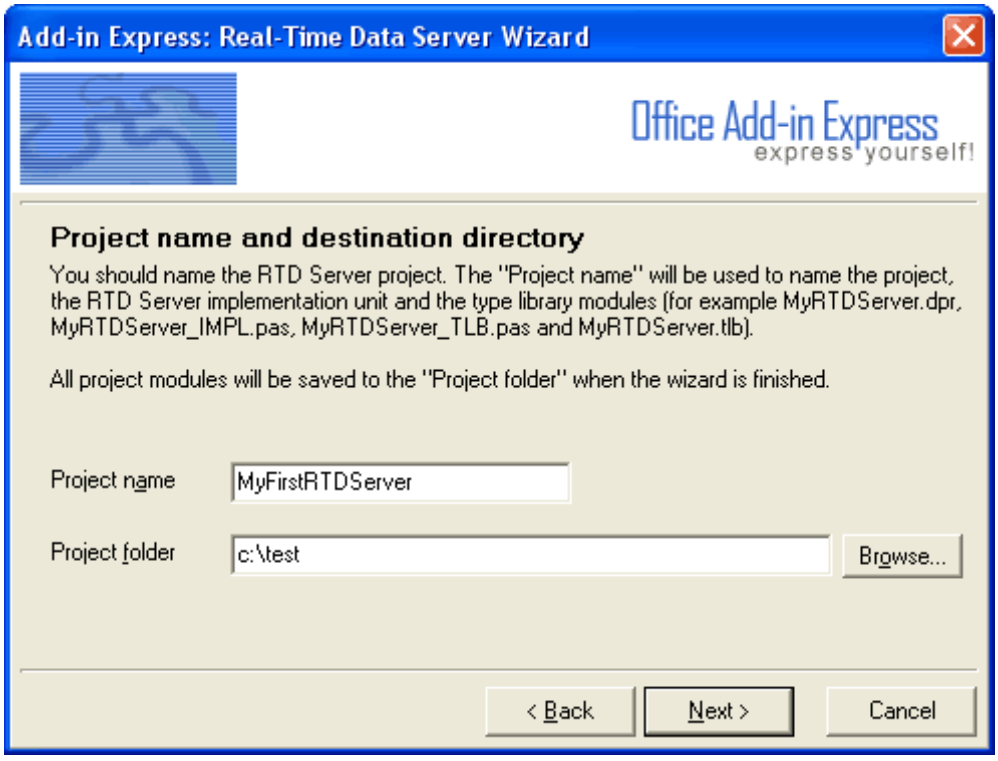
- A real-time data source can be any source of data that can be accessed programmatically.
- An RTD server is a Component Object Model (COM) Automation server that implements the IRtdServer interface. Excel uses the RTD server to communicate with a real-time data source.
- A topic is a string (or a set of strings) that uniquely identifies a piece of data that resides in a real-time data source. The RTD server passes the topic to the real-time data source and receives the value of the topic from the real-time data source; the RTD server then passes the value of the topic to Excel for display. For example, the RTD server passes the topic "NewTopic" to the real-time data source, and the RTD server receives the topic's value of "72.12" from the real-time data source. The RTD server then passes the topic's value to Excel for display.

Are RTD servers available on Add-in Express? Yes, let’s go.

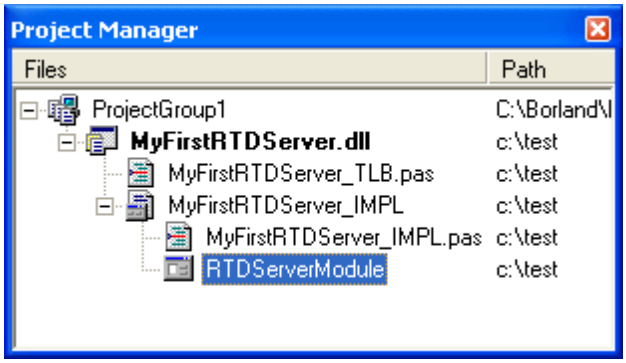
Step #1 – Run the RTD Server specific wizard

In Delphi IDE, close all opened projects, select the “File | New | Others…” item on the main menu, and run the “MS Excel Real-Time Data Server” wizard on the “Add-in Express 2.0” tab of the “New Item” dialog box.

Enter the name of the add-in project and the destination folder for it, and click Next. Then specify the type of server, name the coclass of your server, click Next and Finish.

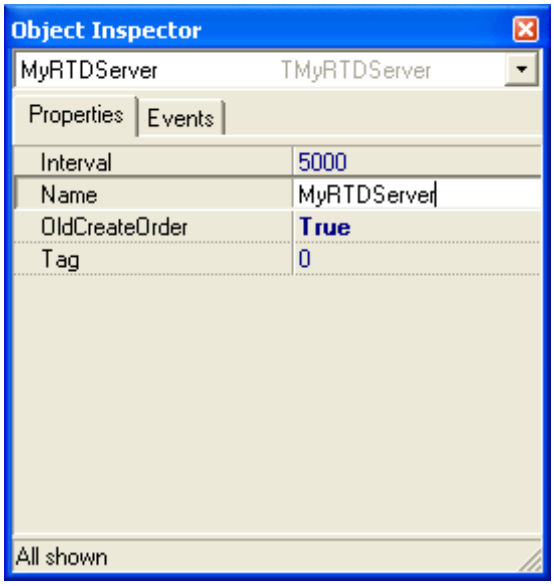


The wizard generates a new project and opens it in Delphi IDE.



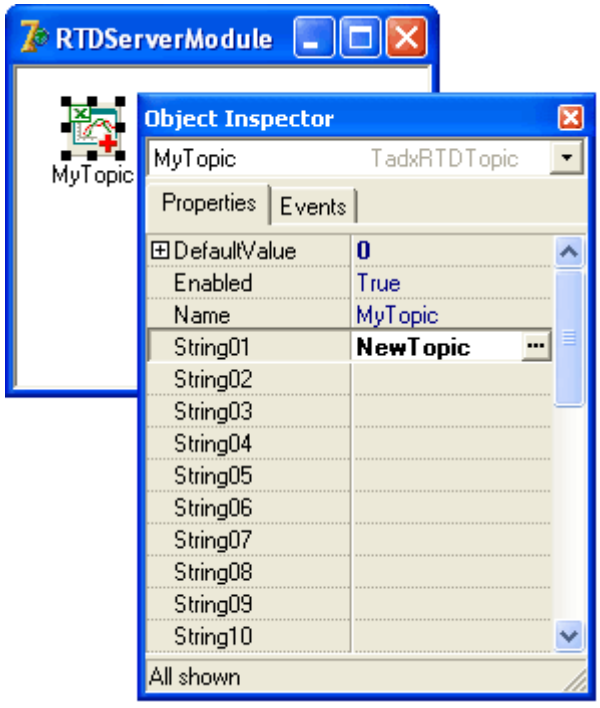
Step #2 – Name your RTD server

Name your RTD server via the Name property of the RTDServer module.



Step #2 – Add a new topic

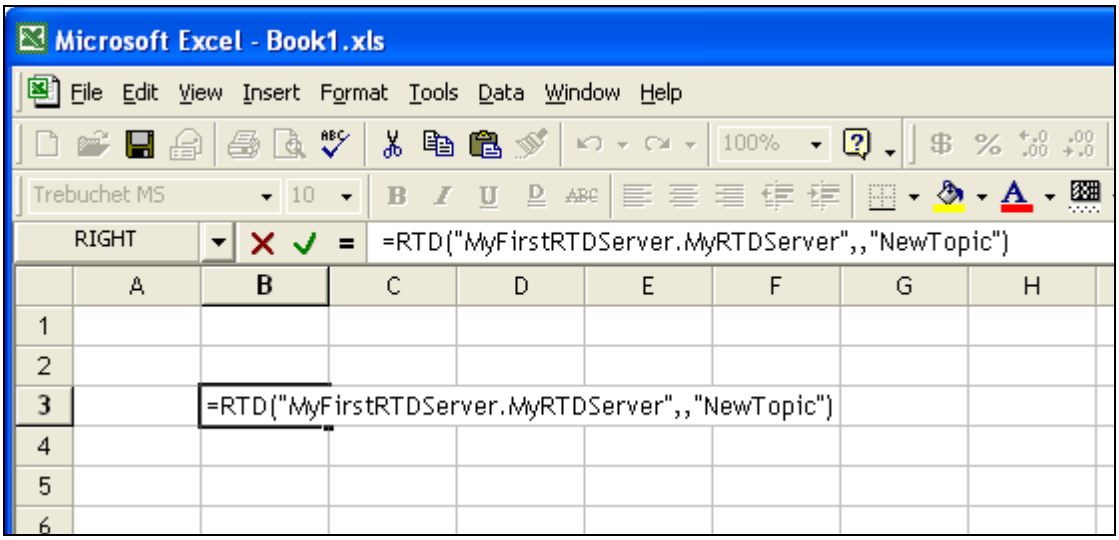
To add a new topic, select the Add-in Express tab on Component Palette and add adxRTDTopic to the RTDServer module. In Object Inspector, specify a topic via the StringXX property and handle the OnRefreshData event.




```
function TRTDServerModule.MyTopicRefreshData(Sender: TObject): OleVariant;
begin
  Result := RandomRange(-100, 100);
end;
```

Step #3 – Register and run an add-in

Save the add-in project, compile it, close Excel, and register the add-in via “Run | Register ActiveX Server”. Run Excel and enter the RTD function to a cell.



The syntax for the Excel RTD worksheet function is:

=RTD(ProgID, Server, String1, String2, ... String28)

The ProgID parameter is a required string value representing the programmatic ID (ProgID) of the RTD server. The Server parameter is a required string value representing the name of the computer on your intranet (using Distributed COM) on which the RTD server is running. If the RTD server is running on the local computer, leave this parameter blank or use two quotation marks ("").

Note. When you use the RTD method of the WorksheetFunction object, you cannot leave the Server parameter blank; you must use two quotation marks to represent the local computer.

The String1 through String28 parameters represent topics to be sent to the RTD server. Only the String1 parameter is required; the String2 through String28 parameters are optional. There is a limit of 28 parameters, and in most cases, only the String1 parameter is used. The actual values for the String1 through String28 parameters depend on the requirements of the real-time data server.

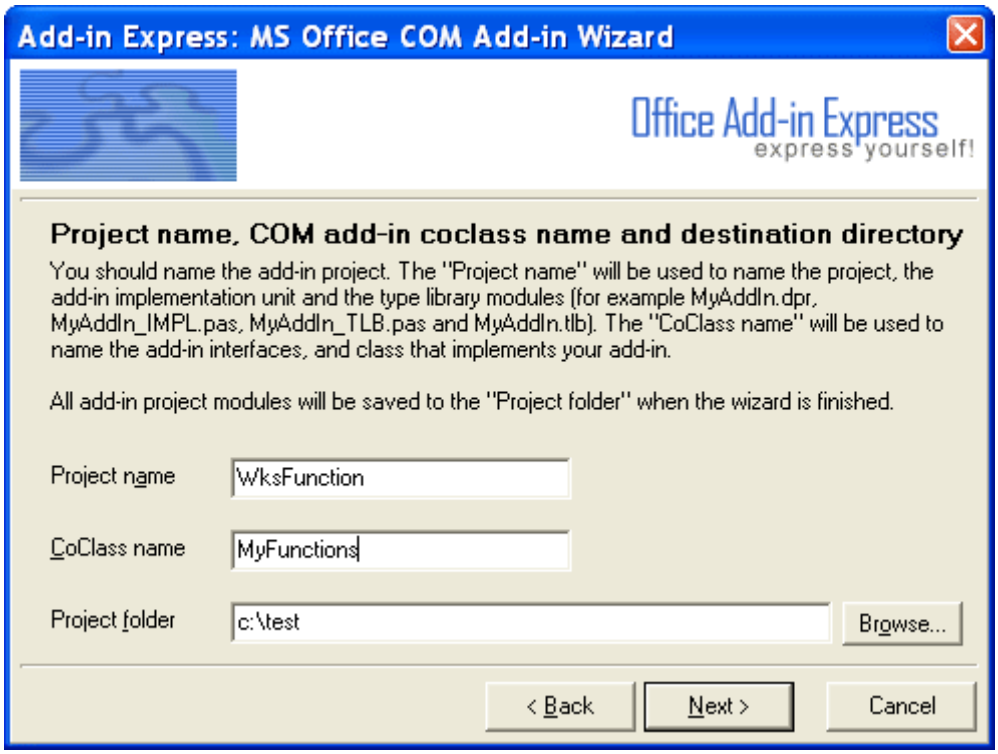
Excel Automation Add-ins

In Excel 2002 and later, COM add-ins may be used to develop new worksheets functions. There is no difference from COM add-ins but in a special registration method. Add-in Express simplifies the creation of Automation Add-ins. Note. Automation add-ins are supported by Excel 2002 and higher.

Step #1 – Run the COM add-in specific wizard

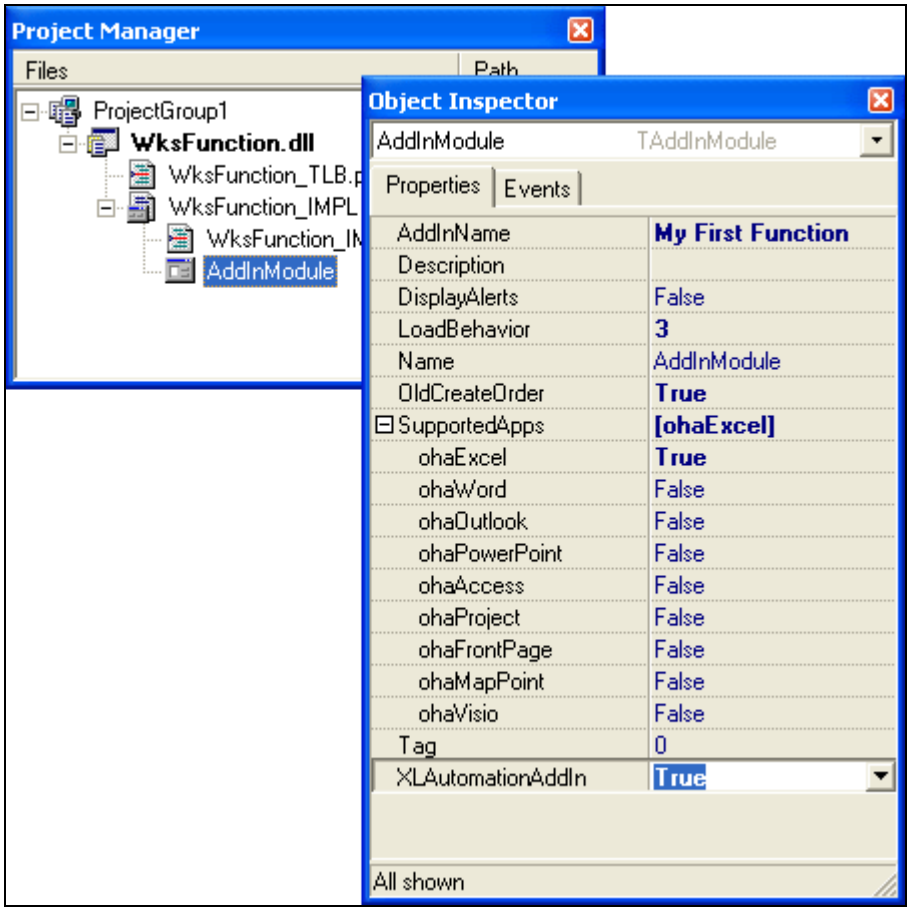
In Delphi IDE, close all opened projects, select the “File | New | Others…” item on the main menu, and run the “MS Office COM add-in” wizard on the “Add-in Express 2.0” tab of the “New Item” dialog box.

Enter the name of the project, the add-in coclass name and the destination folder for the project, and click Next and then click Finish. The wizard generates a new project and opens it in Delphi IDE.



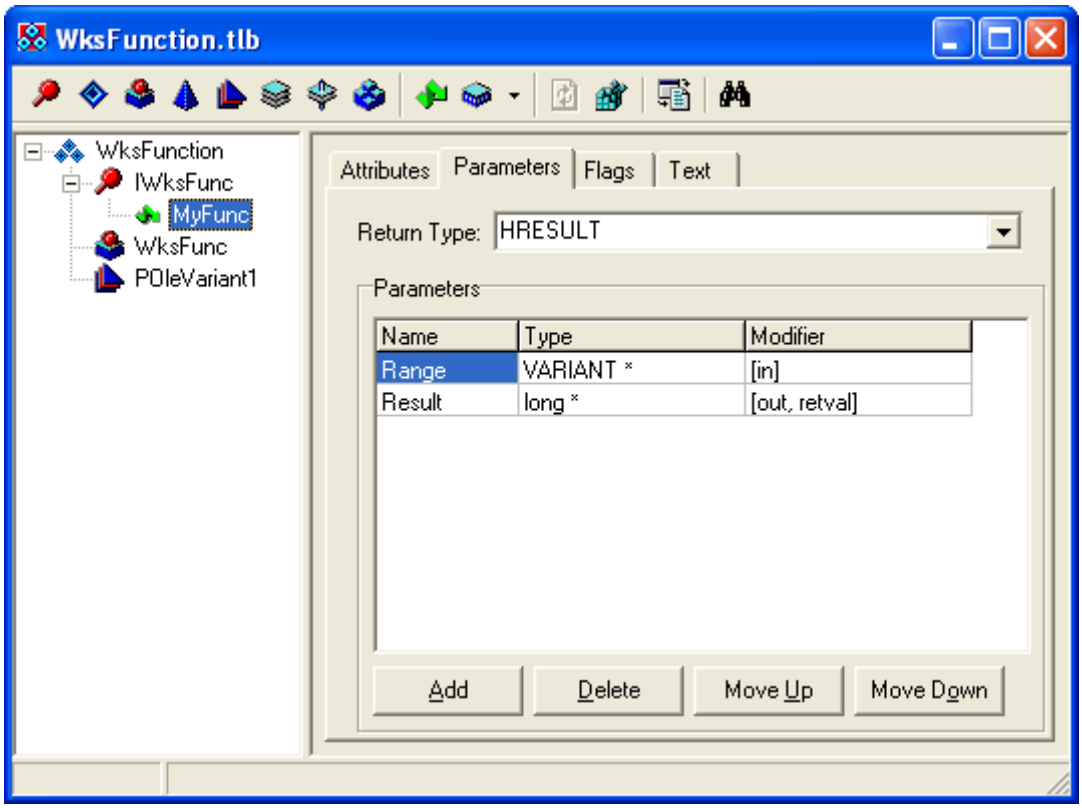
Step #2 – Customize the add-in

In the Project Manager window, select the add-in module, activate Object Inspector, name your add-in via the AddInName property (this name will appear in the COM add-ins dialog box of the host applications), select Excel as a host application, and set True to the XLAutomationAddIn property.



Step #4 – Add a new function to the type library

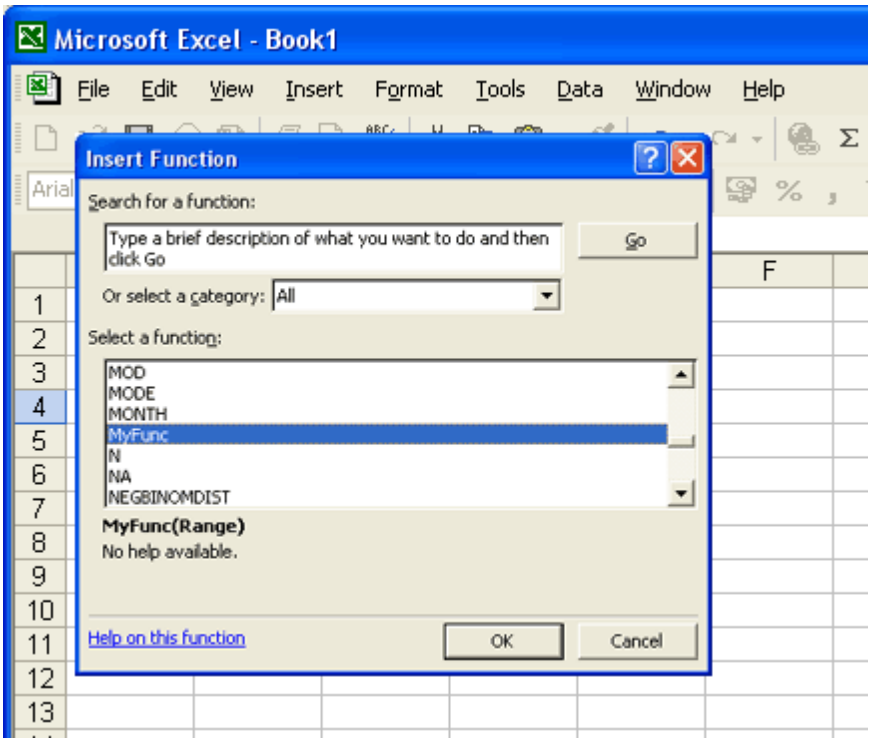
In the Type Library Editor, add a new method (or a new read only property) to the add-in interface, click the Refresh button, go to the add-in implementation unit and implement the added method.




```
function TWksFunc.MyFunc(var Range: OleVariant): Integer;
begin
  Result := 0;
  case VarType(Range) of
    varDispatch: Result := Range.Cells[1, 1].Value div 2;
    varSmallint, varInteger, varSingle,
    varDouble, varCurrency, varShortInt, varByte,
    varWord, varLongWord, varInt64: Result := Range div 2;
  end;
end;
```

Step #5 – Register and run the add-in

Save the add-in project, compile it, close Excel, and register the add-in via “Run | Register ActiveX Server”. Run Excel and enter the new function to a cell.



Class Reference

This chapter describes briefly the contents of the main units that make up Add-in Express. First you will find the overview of all Add-in Express units, and then the main Add-in Express types, constants and classes are described in more details. Note! All constants, types and classes of Add-in Express 1.x (with the “axp” prefix) are preserved for compatibility and their description you can find in “Add-in Express 1 Developer’s Guide” (see the Download section on the Add-in Express website).

Add-in Express Units

Unit	Contents
adxAddIn	The main Add-in Express unit. All types and classes needed for COM add-ins development. Described below.
adxAddInDesignerObjects	The imported type library of COM add-ins they are based on.
adxOffice	The MS Office 2000 imported type library that is used only for ADX classes' implementation.
adxRTDServ	Implements Excel Real-Time Data Servers. Described below.
adxSmartTag	Implements Office Smart Tags.
adxSmartTagTLB	The Smart Tag XP/2003 imported type library.
FrontPage2000, FrontPageEditor2000	The FrontPage 2000 imported type library.
MapPoint2002	The MapPoint 2002 imported type library.
Project2000	The MS Project 2000 imported type library.
Publisher2003	The Publisher 2003 imported type library.
Visio2002	The Visio 2002 imported type library.

Constants

adxLCID

Default locale identifier. Unit: adxAddIn

```
const
  adxLCID: integer = LOCALE_USER_DEFAULT;
```

Can be used when calling host application interface methods.

adxVersion

Returns the current version of Add-in Express. Unit: adxAddIn

```
const
  adxVersion: string = '2.0.0.144 VCL';
```

Returns as a string the current version of Add-in Express including its build.

adxOfficeAppNames

Array of the application names supported by Add-in Express. Unit: adxAddIn

```
const
  adxOfficeAppNames:array [TadxOfficeHostApp] of string =
    ('Excel', 'Word', 'Outlook', 'PowerPoint', 'Access', 'MS Project',
    'FrontPage', 'MapPoint', 'Visio');
```

Types

TadxControlType

The types of command bar controls supported by Add-in Express. Unit: adxAddIn

```
type
  TadxControlType = (adxButton, adxEdit, adxDropdown, adxComboBox, adxPopup);
```

TadxHostAppSet, TadxOfficeHostApp

TadxHostAppSet is a set of the applications supported by Add-in Express. Unit: adxAddIn

```
type
  TadxOfficeHostApp = (ohaExcel, ohaWord, ohaOutlook, ohaPowerPoint, ohaAccess,
    ohaProject, ohaFrontPage, ohaMapPoint, ohaVisio, ohaPublisher);
  TadxHostAppSet = set of TadxOfficeHostApp;
```

TadxMsoBarPosition

The types position where a command bar can be shown. Unit: adxAddIn

```
type
  TadxMsoBarPosition = (adxMsoBarLeft, adxMsoBarTop, adxMsoBarRight,
    adxMsoBarBottom, adxMsoBarFloating, adxMsoBarPopup, adxMsoBarMenuBar);
```

TadxMsoBarProtection

The types of protection of the add-in command bar. Unit: adxAddIn

```
type
  TadxMsoBarProtection = (adxMsoBarNoProtection, adxMsoBarNoCustomize,
    adxMsoBarNoResize, adxMsoBarNoMove, adxMsoBarNoChangeVisible,
    adxMsoBarNoChangeDock, adxMsoBarNoVerticalDock, adxMsoBarNoHorizontalDock);
```

TadxMsoBarType

The types of command bars that are supported by Add-in Express. Unit: adxAddIn

```
type
  TadxMsoBarType = (adxMsoBarTypeNormal, adxMsoBarTypeMenuBar,
    adxMsoBarTypePopup);
```

TadxMsoButtonState

States of the command bar button. Unit: adxAddIn

```
type
  TadxMsoButtonState = (adxMsoButtonUp, adxMsoButtonDown, adxMsoButtonMixed);
```

TadxMsoButtonStyle

The types of command bar style that are supported by Add-in Express. Unit: adxAddIn

```
type
  TadxMsoButtonStyle = (adxMsoButtonAutomatic, adxMsoButtonIcon,
    adxMsoButtonCaption, adxMsoButtonIconAndCaption,
    adxMsoButtonIconAndWrapCaption, adxMsoButtonIconAndCaptionBelow,
    adxMsoButtonWrapCaption, adxMsoButtonIconAndWrapCaptionBelow);
```

TadxMsoComboStyle

The types of combo box style supported by Add-in Express. Unit: adxAddIn

```
type
  TadxMsoComboStyle = (adxMsoComboNormal, adxMsoComboLabel);
```

TadxMsoCommandBarButtonHyperlinkType

The types of command bar hyperlink that are supported by Add-in Express. Unit: adxAddIn

```
type
  TadxMsoCommandBarButtonHyperlinkType = (
    adxMsoCommandBarButtonHyperlinkNone,
    adxMsoCommandBarButtonHyperlinkOpen,
    adxMsoCommandBarButtonHyperlinkInsertPicture);
```

TadxOLItemClasses, TadxOLItemClass

The types of Outlook item classes supported by Add-in Express. Unit: adxAddIn

```
type
  TadxOLItemClass = (adxOLAppointment, adxOLMeetingRequest,
    adxOLMeetingCancellation, adxOLMeetingResponseNegative,
```

```
    adxOLMeetingResponsePositive, adxOLMeetingResponseTentative,
    adxOLContact, adxOLJournal, adxOLMail, adxOLPost, adxOLTask,
    adxOLTaskRequest, adxOLTaskRequestUpdate, adxOLTaskRequestAccept,
    adxOLTaskRequestDecline, adxOLDistributionList);
  TadxOLItemClasses = set of TadxOLItemClass;
```

TadxOLItemTypes, TadxOLItemType

Types of the Outlook item types supported by Add-in Express. Unit: adxAddIn

```
type
  TadxOLItemType = (adxOLMailItem, adxOLAppointmentItem, adxOLContactItem,
    adxOLTaskItem, adxOLJournalItem, adxOLNoteItem, adxOLPostItem,
    adxOLDistributionListItem);
  TadxOLItemTypes = set of TadxOLItemType;
```

TadxOLItemTypeAction

Types of the actions for command bar control.

```
type
  TadxOLItemTypeAction = (adxOLActionNone, adxOLActionShow, adxOLActionEnable);
```

TadxStartMode

The types of values that indicate why the add-in was started. Unit: adxAddIn

```
type
  TadxStartMode = (smFirstStart, smNormal, smUninstall);
```

TadxKindType

Specifies the possible variants of smart tag phrases recognition. Unit: adxSmartTag

```
type
  TadxKindType = (ktList, ktMask, ktCustom);
```

- ktList – specifies that the smart tag phrases are located in the RecognizedWords property.

- ktMask - specifies that the smart tag phrase is recognized by a mask (the RecognizedMask property).
- ktCustom – specifies that the smart tag phrases are recognized dynamically by the OnRecognize event.

Classes

TadxAddIn

TadxAddIn is an interfaced class that implements the IDTExtensibility2 interface required by COM add-ins. Unit: adxAddIn

```
type
  TadxAddin = class(TAutoObject, IDTExtensibility2)
  public
    property COMAddInModule: TadxCOMAddInModule;
    property Factory: TadxFactory;
  end;
```

The TadxAddIn class is a subsidiary class that implements the COM add-in interface. You needn't call its methods and properties.

COMAddInModule Property

Specifies the add-in module object for the TadxAddIn class. Read-only.

```
public
  property COMAddInModule: TadxCOMAddInModule;
```

Factory Property

Specifies the class factory object for the TadxAddIn class. Read-only.

```
public
  property Factory: TadxFactory;
```

Factory is the class factory that is used to instantiate a TadxAddIn.

TadxBuiltInControl

Allows handling events of all built-in controls of the host application. Unit: adxAddIn

```
type
  TadxBuiltInControl = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    property Owner: TadxCOMAddInModule;
    property DefaultInterface: CommandBarControl;
  published
    property BuiltInID: Integer default 1;
    property OfficeTag: WideString;
    property CommandBar: TadxCommandBar;
    property DisableStandardAction: boolean default False;
    property OnAction: TNotifyEvent;
  end;
```

TadxBuildInControl is a wrapper over the Office2000.CommandBarControl interface and allows accessing to its properties and methods. Use TadxBuiltInControl to handle events of the built-in controls of the add-in host application.

CommandBar Property

Returns or sets the command bar that contains this control. Read-write.

```
published
  property CommandBar: TadxCommandBar;
```

DefaultInterface Property

Provides access to the default interface for the component. Read-only.

```
public
  property DefaultInterface: CommandBarControl;
```

Returns a reference to the Office2000.CommandBarControl interface of the control instance.

DisableStandardAction Property

Allows disabling the action of the built-in control. Read-write.

```
published
  property DisableStandardAction: Boolean default False;
```

Set True to disable the action that the built-in control fires. The default value is False.

BuiltInID Property

Returns or sets the ID for a built-in command bar control. Read-write.

```
published
  property BuiltInID: Integer;
```

A control's ID determines the built-in action for that control. The default value is 1.

Note. Use “Built-in Controls Scanner” to get all IDs of all host applications (see the Download section on the Add-in Express website).

OfficeTag Property

A unique value that identifies the control. Read-write.

```
published
  property OfficeTag: WideString;
```

Do not change this property in design-time. It is done automatically.

Owner Property

Returns the Owner of the component. Read-only.

```
public
  property Owner: TadxCOMAddInModule;
```

Call the Owner to obtain a reference to the add-in module that owns this component.

OnAction Event

Occurs when the user clicks a button (a menu item) or changes the value of an edit box (a combo box).

```
published
  property OnAction: TNotifyEvent;
```

TadxCOMAddInModule

TadxCOMAddInModule centralizes handling of Add-in Express components. Unit: adxAddIn

```
type
  TadxCOMAddInModule = class(TDataModule)
  protected
    procedure AddMapPointCommand(const DisplayName, MethodName: WideString);
    procedure OptionsPagesAdd(ASender: TObject;
      const Pages: PropertyPages); virtual;
    procedure NameSpaceOptionsPagesAdd(ASender: TObject;
      const Pages: PropertyPages; const Folder: MAPIFolder); virtual;
  public
    constructor Create(AOwner: TComponent); override;
    constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
    destructor Destroy; override;
    function CommandBarIndexByName(const CommandBarName: WideString):
      TadxCommandBar;
    function CommandBarIndexOf(const CmdBar: TadxCommandBar): Integer;
    function FindCommandBar(const CmdBarName: WideString): CommandBar;
    function FindControl(ACommandBar: CommandBar; AType: MsoControlType;
      ID: Integer): CommandBarControl; overload;
    function FindControl(ACommandBar: CommandBar; AType: MsoControlType;
      Tag: WideString): CommandBarControl; overload;
    function FindControl(AType: MsoControlType; ID: Integer):
      CommandBarControl; overload;
    function FindControl(AType: MsoControlType; Tag: WideString):
       CommandBarControl; overload;
    property COMAddInClassInstance: TadxAddIn;
    property COMAddInClassFactory: TadxFactory;
    property CommandBars[Index: Integer]: TadxCommandBar;
    property CommandBarsCount: Integer;
    property RegistryKey: string;
    property HostApp: OLEVariant;
    property HostType: TadxOfficeHostApp;
    property AddInInstance: COMAddIn;
```

```
    property StartMode: TadxStartMode;
    property ExcelApp: Excel2000.TExcelApplication;
    property WordApp: Word2000.TWordApplication;
    property OutlookApp: Outlook2000.TOutlookApplication;
    property PowerPointApp: MSPPt2000.TPowerPointApplication;
    property AccessApp: Access2000.TAccessApplication;
    property ProjectApp: Project2000.TProjectApplication;
    property FrontPageApp: FrontPage2000.TFrontPageApplication;
    property MapPointApp: MapPoint2002.TMapPointApplication;
    property VisioApp: Visio2002.TVisioApplication;
  published
    property AddInName: string;
    property Description: string;
    property LoadBehavior: Integer default 3;
    property DisplayAlerts: boolean default False;
    property XLAutomationAddIn: boolean default False;
    property SupportedApps: TadxHostAppSet default [ohaExcel];
    property OnError: TadxOnError;
    property OnAddInInitialize: TNotifyEvent;
    property OnAddInStartupComplete: TNotifyEvent;
    property OnAddInBeginShutdown: TNotifyEvent;
    property OnAddInFinalize: TNotifyEvent;
    property OnAfterAddinRegister: TNotifyEvent;
    property OnBeforeAddinUnregister: TNotifyEvent;
  end;
```

TadxCOMAddInModule is a descendant of TDataModule and is used to provide True RAD for add-in development.

Use a TadxCOMAddInModule object in an application to provide a location for centralized handling of Add-in Express components. Typically these are command bar and command bar controls components, such as TadxCommandBar and TadxBuiltInControl. COMAddInModules are not limited to the ADX components; they can also contain other nonvisual components, such as TTimer, TOpenDialog, or TImageList). At design time a TadxCOMAddInModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them.

AccessApp Property

A reference to the Access application. Read-only.

```
public
  property AccessApp: Access2000.TAccessApplication;
```

Allows accessing to Access if it is the host application. Returns an automation object wrapper instance if Access is the host application, else returns nil.

AddInInstance Property

A reference to the COM add-in interface. Read-only.

```
public
    property AddInInstance: COMAddIn;
```

This property returns a reference to the COMAddIn interface that refers to this add-in in the COMAddIns collection of the host application.

AddInName Property

Returns or sets a name of the add-in. Read-write.

```
published
  property AddInName: string;
```

Specifies the name of the add-in that will be shown in the COM Add-ins dialog box of the host application.

COMAddInClassFactory Property

Returns a reference to the class factory. Read-only.

```
public
  property COMAddInClassFactory: TadxFactory;
```

Returns a reference to the instance of the TadxFactory class. Allows accessing to the class factory properties. The reference is kept actual all the time the add-in is running.

COMAddInClassInstance Property

Returns a reference to the COM add-in instance. Read-only.

```
public
  property COMAddInClassInstance: TadxAddIn;
```

Returns a reference to the instance of the TadxAddIn class that implements the COM add-in interface.

CommandBars Property

Enumerates all add-in's command bars. Read-only.

```
public
  property CommandBars[Index: Integer]: TadxCommandBar;
```

Returns by index a reference to an instance of TadxCommandBar contained in the add-in module.

CommandBarsCount Property

Returns the total number of add-in's command bars. Read-only.

```
public
  property CommandBarsCount: Integer;
```

Returns the total number of instances of TadxCommandBar contained in the add-in module.

Description Property

Returns or sets the description of the add-in. Read-write.

```
published
  property Description: string;
```

Returns or sets a descriptive string for the add-in.

DisplayAlerts Property

Enables/disables alerts. Read-write.

```
published
  property DisplayAlerts: boolean default False;
```

When the COM add-in is being registered or unregistered, all instances of the host application must be closed. Set True to the DisplayAllerts property if you want to notify a user that the host applications are not closed.

ExcelApp Property

A reference to the Excel application. Read-only.

```
public
  property ExcelApp: Excel2000.TExcelApplication;
```

Allows accessing to Excel if it is the host application. Returns an automation object wrapper instance if Excel is the host application, else returns nil.

FrontPageApp Property

A reference to the FrontPage application. Read-only.

```
public
  property FrontPageApp: FrontPage2000.TFrontPageApplication;
```

Allows accessing to FrontPage if it is the host application. Returns an automation object wrapper instance if FrontPage is the host application, else returns nil.

HostApp Property

Returns a reference to the host application interface. Read-only.

```
public
  property HostApp: OLEVariant;
```

Returns a reference to the interface (Excel.Application, for example) of the host application, where the add-in is running. Allows accessing interfaces of the host application and their properties and methods. The reference is kept actual all the time when add-in is running.

HostType Property

Returns a host application type. Read-only.

```
public
```

```
  property HostType: TadxOfficeHostApp;
```

Returns the type of the host application, where the add-in is running. Allows identification of the host application. Is useful in developing COM Add-ins that are to work in several host applications. The value is kept actual all the time when the add-in is running.

LoadBehavior Property

Sets or returns when the add-in starts. Read-write.

```
published
  property LoadBehavior: Integer default 3;
```

This value determines how the add-in is loaded by the host application, and is made up of a combination of the following values:

- 0 – Disconnect. The COM Add-In is not loaded.
- 1 – Connected. The COM Add-in is loaded.
- 2 – Bootload. Load on application Startup.
- 8 – DemandLoad. Load only when requested by the user.
- 16 – ConnectFirstTime. Load only once (on next startup).

A typical value specified is 3 (Connected and Bootload).

MapPointApp Property

A reference to the MapPoint application. Read-only.

```
public
  property MapPointApp: MapPoint2002.TMapPointApplication;
```

Allows accessing to MapPoint if it is the host application. Returns an automation object wrapper instance if MapPoint is the host application, else returns nil.

OutlookApp Property

A reference to the Outlook application. Read-only.

```
public
  property OutlookApp: Outlook2000.TOutlookApplication;
```

Allows accessing to Outlook if it is the host application. Returns an automation object wrapper instance if Outlook is the host application, else returns nil.

PowerPointApp Property

A reference to the PowerPoint application. Read-only.

```
public
  property PowerPointApp: MSPPt2000.TPowerPointApplication;
```

Allows accessing to PowerPoint if it is the host application. Returns an automation object wrapper instance if PowerPoint is the host application, else returns nil.

ProjectApp Property

A reference to the MS Project application. Read-only.

```
public
  property ProjectApp: Project2000.TProjectApplication;
```

Allows accessing to MS Project if it is the host application. Returns an automation object wrapper instance if MS Project is the host application, else returns nil.

RegistryKey Property

Returns the registry key of the add-in registration record. Read-only.

```
public
  property RegistryKey: string;
```

The location of the key is “\Software\Microsoft\Office\%HOST%\AddIns\%PROGID%”, where HOST is the name of MS Office application, and PROGID is the ProgID of the class factory.

Registry Property

Provides access to the registry path corresponding to the add-in. Read-only.

```
public
  property Registry: TRegistry;
```

The location of the opened key is “\Software\Microsoft\Office\%HOST%\AddIns\%PROGID%”, where HOST is the name of MS Office application, and PROGID is the ProgID of the class factory.

StartMode Property

Returns a start mode of the add-in. Read-only.

```
public
  property StartMode: TadxStartMode;

type
  TadxStartMode = (smFirstStart, smNormal, smUninstall);
```

Allows determining a start mode of the add-in.

SupportedApps Property

Returns or sets the applications that are supported by the add-in.

```
published
  property SupportedApps: TadxHostAppSet;

type
  TadxOfficeHostApp = (ohaExcel, ohaWord, ohaOutlook, ohaPowerPoint,
    ohaAccess, ohaProject, ohaFrontPage, ohaMapPoint, ohaVisio, ohaPublisher);
  TadxHostAppSet = set of TadxOfficeHostApp;
```

Use this property to support several applications as the host application.

VisioApp Property

A reference to the MS Visio application. Read-only.

```
public
  property VisioApp: Visio2002.TVisioApplication;
```

Allows accessing to MS Visio if it is the host application. Returns an automation object wrapper instance if MS Visio is the host application, else returns nil.

WordApp Property

A reference to the Word application. Read-only.

```
public
  property WordApp: Word2000.TWordApplication;
```

Allows accessing to Word if it is the host application. Returns an automation object wrapper instance if Word is the host application, else returns nil.

XLAutomationAddIn Property

Determines if the add-in is the Excel Automation Add-in. Read-write.

```
published
  property XLAutomationAddIn: boolean default False;
```

AddMapPointCommand Method

Adds an item to the Tools menu of MapPoint.

```
protected
  procedure AddMapPointCommand(const DisplayName, MethodName: WideString);
```

Use this method to add add-in commands to Microsoft MapPoint.

Parameters:

- DisplayName – a name of a new item.
- MethodName - a method name of the COM object that implements the add-in. The method should be added to the add-in COM object via the Type Library Editor.

Example:

```
type
  TMPTestAddIn = class(TadxAddin, IMPTestAddIn)
  private
    procedure ADXMapPointMethod01; safecall;
    procedure ADXMapPointMethod02; safecall;
```

```
  end;

  TAddInModule = class(TadxCOMAddInModule)
    procedure adxCOMAddInModuleAddInInitialize(Sender: TObject);
  private
  protected
  public
  end;

implementation

{$R *.dfm}

{ TMPTestAddIn }

procedure TMPTestAddIn.ADXMapPointMethod01;
begin
  showmessage('ADXMapPointMethod01');
end;

procedure TMPTestAddIn.ADXMapPointMethod02;
begin
  showmessage('ADXMapPointMethod02');
end;

{ TAddInModule }

procedure TAddInModule.adxCOMAddInModuleAddInInitialize(Sender: TObject);
begin
  AddMapPointCommand('ADXMapPointCommand01', 'ADXMapPointMethod01');
  AddMapPointCommand('ADXMapPointCommand02', 'ADXMapPointMethod02');
end;

initialization
  TadxFactory.Create(ComServer, TMPTestAddIn, CLASS_MPTestAddIn, TAddInModule);
```

```
end.
```

CommandBarByName Method

Finds an add-in command bar by its name.

```
public
    function CommandBarIndexByName(const CommandBarName: WideString):
      TadxCommandBar;
```

In the add-in module, finds an instance of TadxCommandBar with the name specified by the CommandBarName parameter and returns a reference to it.

CommandBarIndexOf Method

Returns the index of the add-in command bar.

```
public
    function CommandBarIndexOf(const CmdBar: TadxCommandBar): Integer;
```

Call IndexOf to get the index for the add-in command bar specified as the CmdBar parameter.

FindCommandBar Method

Finds command bars.

```
public
    function FindCommandBar(const CmdBarName: WideString): CommandBar;
```

In the host application, finds a command bar with the name specified by the CmdBarName parameter and returns a reference to the command bar interface. Note. The FindCommandBar method is optimized to fast search. Do not worry if you get an exception within the FindCommandBar method. We use this trick to accelerate this method.

FindControl Method

Finds command bar controls.

```
public
    function FindControl(ACommandBar: CommandBar; AType: MsoControlType;
```

```
      ID: Integer): CommandBarControl; overload;

    function FindControl(ACommandBar: CommandBar; AType: MsoControlType;
      Tag: WideString): CommandBarControl; overload;

    function FindControl(AType: MsoControlType; ID: Integer):
      CommandBarControl; overload;

    function FindControl(AType: MsoControlType; Tag: WideString):
      CommandBarControl; overload;
```

Finds a command bar control with the specified parameters and returns a reference to the command bar control interface.

Parameter:

- ACommandBar – a reference to the command bar interface.
- AType – a command bar control type.
- ID – a command bar control ID.
- Tag – a command bar control tag.

NameSpaceOptionsPagesAdd Method

An Outlook specific method.

```
protected
    procedure NameSpaceOptionsPagesAdd(ASender: TObject;
      const Pages: PropertyPages; const Folder: MAPIFolder); virtual;
```

The Add-in Express wizards generate this method automatically. You do not need to call this method manually.

OptionsPagesAdd Method

An Outlook specific method.

```
protected
    procedure OptionsPagesAdd(ASender: TObject;
      const Pages: PropertyPages); virtual;
```

The Add-in Express wizards generate this method automatically. You do not need to call this method manually.

OnAddInBeginShutdown Event

Occurs just before the host application begins its unloading procedures.

```
published
  property OnAddInBeginShutdown: TNotifyEvent;
```

The OnAddInBeginShutdown event enables the scenarios where the add-in needs to save state information of any window or dialog box before the host application unloads any UI or add-in.

OnAddInFinalize Event

Occurs when the add-in is disconnected from the host application

```
published
  property OnAddInFinalize: TNotifyEvent;
```

If the host application is disconnecting the add-in, the add-in should leave any customizations it has made to the host command bar set so that the customizations are displayed the next time the host is launched. The add-in can then be demand-loaded during the next session of the host since its command bar customizations are already added. The add-in actually is loaded the first time the user clicks on a command bar customization.

OnAddInInitialize Event

Occurs when the add-in is connected to the host application, either through the COM Add-Ins dialog box or another add-in.

```
published
  property OnAddInInitialize: TNotifyEvent;
```

OnAddInStartupComplete Event

Occurs when the startup of the host application is complete.

```
published
  property OnAddInStartupComplete: TNotifyEvent;
```

The OnAddInStartupComplete event enables scenarios where the add-in needs to know when the host application has completed its startup sequences and loaded any files, add-ins, or other objects into memory. For example, when the OnAddInStartupComplete event is fired, another add-in can then check if certain other add-ins were loaded or not during startup and, hence, determine the behavior of the add-in based on this information.

OnAfterAddinRegister Event

Occurs when the add-in has been registered.

```
published
  property OnAfterAddinRegister: TNotifyEvent;
```

The OnAfterAddinRegister event enables custom actions when the add-in needs to prepare something before the first start of the hostapp with the add-in. This event occurs after the add-in adds its key to the host application registry path, and is fired separately for each host application (for shared add-ins).

OnBeforeAddinUnregister Event

Occurs when the add-in is being unregistered.

```
published
  property OnBeforeAddinUnregister: TNotifyEvent;
```

The OnBeforeAddinUnregister event enables custom actions when the add-in needs to remove something before the add-in is unregistered. This event occurs before the add-in removes its key from the host application registry path, and is fired separately for each host application (for shared add-ins).

OnError Event

Occurs when an exception is raised.

```
published
  property OnError: TadxOnError;

type
  TadxOnError = procedure (const E: SysUtils.Exception;
    var Handled: boolean) of object;
```

Allows handling an exception. By default, Add-in Express handles exceptions and shows an error message box. You can handle an exception yourself and avoid error messages via Handled := True.

OnOLXXX Events

These events wrap the corresponding Outlook.Application events to organize a synchronous event pool and avoid errors when event handlers are called by Outlook asynchronously.

Note. You should use these events instead using them directly from Outlook.Application.

TadxCommandBar

A wrapper over the Office2000.CommandBar interface. Unit: adxAddin

```
type
  TadxCommandBar = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function ControlByCaption(const ACaption: WideString):
      TadxCommandBarControl;
    function ControlByTag(const ATag: Integer): TadxCommandBarControl;
    function ControlByOfficeTag(const AOfficeTag: WideString):
      TadxCommandBarControl;
    property CommandBarNameLocal: WideString;
    property BuiltIn: WordBool;
    property Context: WideString;
    property Index: SYSINT;
    property Parent: IDispatch;
    property Type_: TadxMsoBarType;
    property AdaptiveMenu: WordBool;
    property Height: SYSINT;
    property Width: SYSINT;
    property DefaultInterface: CommandBar;
    property Owner: TadxCOMAddInModule;
  published
    property Controls: TadxCommandBarControls;
    property Enabled: WordBool default True;
    property CommandBarLeft: SYSINT default -1;
    property CommandBarName: WideString;
    property Position: TadxMsoBarPosition default adxMsoBarFloating;
    property RowIndex: SYSINT default -1;
    property Protection: TadxMsoBarProtection default adxMsoBarNoProtection;
    property CommandBarTop: SYSINT default -1;
    property Visible: WordBool default True;
    property Temporary: boolean default False;
```

```
  end;
```

TadxCommandBar is a wrapper over the command bar interface and is a container for all command bar controls. Use this component to add a new command bar to the command bars system of the host application or to connect to the existing command bar.

AdaptiveMenu Property

True if a personalized menu is enabled. Read-write.

```
public
  property AdaptiveMenu: WordBool default False;
```

BuiltIn Property

True if the specified command bar is a built-in command bar of the container application. False if it's a custom command bar. Read-only.

```
public
  property BuiltIn: WordBool;
```

CommandBarLeft Property

Sets or returns the horizontal position of the distance (in pixels) of the command bar, from the left edge of the command bar relative to the screen. Read-write.

```
published
property CommandBarLeft: SYSINT default -1;
```

Note. For docked command bars, this property returns or sets the distance from the command bar to the left of the docking area.

CommandBarName, CommandBarNameLocal Properties

Returns or sets a name of the command bar. Read-write.

```
published
  property CommandBarName: WideString;
  property CommandBarNameLocal: WideString;
```

A local name of the built-in command bar is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars — wherever this list is displayed in the container application. For the built-in command bar, the CommandBarName property returns the command bar's U.S. English name. Use the CommandBarNameLocal property to return the localized name.

If you change the value of the CommandBarNameLocal property for a custom command bar, the value of CommandBarName changes as well, and vice versa.

CommandBarTop Property

Sets or returns the distance (in pixels) from the top edge of the command bar, to the top edge of the screen. Read-write.

```
published
  property CommandBarTop: SYSINT default -1;
```

Note. For docked command bars, this property returns or sets the distance from the command bar to the left of the docking area.

Context Property

Returns or sets a string that determines where a command bar will be saved. This string is defined and interpreted by the application. Read-write.

```
public
  property Context: WideString;
```

You can set the Context property only for custom command bars. This property will fail if the application doesn't recognize the context string, or if the application doesn't support changing context strings programmatically.

Controls Property

Controls of the command bar. Read-write.

```
published
  property Controls: TadxCommandBarControls;
```

Controls are a list of the command bar's child controls. These controls are the buttons, the edit and combo boxes, and the pop-ups that appear on the command bar.

DefaultInterface Property

Provides access to the default interface for the command bar. Read-only.

```
public
  property DefaultInterface: CommandBar;
```

Returns a reference to the Office2000.CommandBar interface of the command bar instance.

Enabled Property

True if the specified command bar is enabled. Read-write.

```
published
    property Enabled: WordBool default True;
```

Height Property

Returns or sets the height of the command bar. Read-write.

```
public
  property Height: SYSINT;
```

Command bar automatically resizes itself to accommodate contained controls.

Index Property

Returns an integer representing of the index number for a command bar in the native CommandBars collection of the host application. Read-only.

```
public
  property Index: SYSINT;
```

Owner Property

Indicates the component that is responsible for streaming and freeing this component. Read-only.

```
public
    property Owner: TadxCOMAddInModule;
```

Use Owner to find the owner of the component and to get access to the add-in module.

Parent Property

Returns a reference to the Parent object interface for the specified object. Read-only.

```
public
  property Parent: IDispatch;
```

Position Property

Returns or sets the position of a command bar. Read-write;

```
published
  property Position: TadxMsoBarPosition  default adxMsoBarFloating;

type
  TadxMsoBarPosition = (adxMsoBarLeft, adxMsoBarTop, adxMsoBarRight,
    adxMsoBarBottom, adxMsoBarFloating, adxMsoBarPopup, adxMsoBarMenuBar);
```

Protection Property

Returns or sets the way a command bar is protected from user customization. Read-write.

```
published
  property Protection: TadxMsoBarProtection default adxMsoBarNoProtection;

type
  TadxMsoBarProtection = (adxMsoBarNoProtection, adxMsoBarNoCustomize,
    adxMsoBarNoResize, adxMsoBarNoMove, adxMsoBarNoChangeVisible,
   adxMsoBarNoChangeDock, adxMsoBarNoVerticalDock, adxMsoBarNoHorizontalDock);
```

Using the constant adxMsoBarNoCustomize prevents users from accessing to the Add or Remove Buttons menu (this menu enables users to customize a toolbar).

SupportedApps Property

Returns or sets the applications to which the command bar will be added.

```
published
  property SupportedApps: TadxHostAppSet;
```

```
type
  TadxOfficeHostApp = (ohaExcel, ohaWord, ohaOutlook, ohaPowerPoint,
    ohaAccess, ohaProject, ohaFrontPage, ohaMapPoint, ohaVisio, ohaPublisher);
  TadxHostAppSet = set of TadxOfficeHostApp;
```

You can use this property when you develop an add-in for several host applications but, for example, the command bar should be added to one or two applications.

RowIndex Property

Returns or sets the docking order of a command bar in relation to other command bars in the same docking area. Read-write.

```
published
    property RowIndex: SYSINT default -1;
```

Can be an integer greater or equal than zero.

Temporary Property

True to make the new command bar temporary. Read-write.

```
published
  property Temporary: Boolean default False;
```

Temporary command bars are deleted when the container application is closed.

Type_ Property

Returns the type of command bar. Read-only.

```
public
  property Type_: TadxMsoBarType;

type
  TadxMsoBarType = (adxMsoBarTypeNormal, adxMsoBarTypeMenuBar,
    adxMsoBarTypePopup);
```

Visible Property

True if the specified object is visible. Read-write.

```
published
  property Visible: WordBool;
```

The Enabled property for a command bar must be set to True before the visible property is set to True.

Width Property

Returns or sets the width (in pixels) of the command bar. Read-write.

```
published
  property Width: SYSINT;
```

Command bar automatically resizes itself to accommodate contained controls.

ControlByCaption Method

Searches for a control with the specified Caption property within the Controls collection.

```
public
  function ControlByCaption(const ACaption: WideString): TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the caption specified by the ACaption parameter. Returns nil if the control isn't found.

ControlByOfficeTag Method

Searches for a control with the specified OfficeTag property within the Controls collection.

```
public
  function ControlByOfficeTag(const AOfficeTag: WideString):
    TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the OfficeTag specified by the AOfficeTag parameter. Returns nil if the control isn't found.

ControlByTag Method

Searches for a control with the specified Tag property within the Controls collection.

```
public
  function ControlByTag(const ATag: Integer): TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the Tag specified by the AOfficeTag parameter. Returns nil if the control isn't found.

TadxCommandBarButton

A wrapper over the CommandBarButton interface. Unit: adxAddIn

```
type
  TadxCommandBarButton = class(TadxCommandBarControl)
  public
    constructor Create(Collection: TCollection); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property BuiltInFace: WordBool default True;
    property DefaultInterface: CommandBarButton;
  published
    property DisableStandardAction: Boolean default False;
    property FaceID: SYSINT default 0;
    property ShortcutText: WideString;
    property State: TadxMsoButtonState default adxMsoButtonUp;
    property Style: TadxMsoButtonStyle default adxMsoButtonAutomatic;
    property HyperlinkType: TadxMsoCommandBarButtonHyperlinkType
      default adxMsoCommandBarButtonHyperlinkNone;
    property Glyph: TBitmap;
    property GlyphTransparentColor: TColor default clDefault;
    property OnClick: TNotifyEvent;
  end;
```

TadxCommandBarButton is a wrapper over the command bar button interface. Use this class to add a new command bar button to the command bar of the host application or to handle any built-in command bar button. TadxCommandBarButton is an item of the TadxCommandBar.Controls collection. Add a button via the TadxCommandBar.Controls.Add method.

To connect to the built-in command bar button you should specify its ID via the OfficeID property.

BuiltInFace Property

True if the face of a command bar button control is its original built-in face. Read-write.

```
public
  property BuiltInFace: WordBool;
```

This property can only be set to True, which will reset the face to the built-in face.

DefaultInterface Property

Provides access to the default interface for the command bar button. Read-only.

```
public
  property DefaultInterface: CommandBarButton;
```

Returns a reference to the Office2000.CommandBarButton interface of the command bar button instance.

DisableStandardAction Property

Allows disabling the action of the built-in button. Read-write.

```
published
  property DisableStandardAction: Boolean default False;
```

Set True to disable the action that the built-in button specified by the OfficeID property. The default value is False.

FaceID Property

Returns or sets the ID number for the face of a command bar button control. Read-write.

```
published
    property FaceID: SYSINT default 0;
```

Glyph Property

Specifies the bitmap that appears on the button. Read-write.

```
published
  property Glyph: TBitmap;
```

Use the Open dialog box that appears as an editor in the Object Inspector to choose a bitmap file (with a .BMP extension) to use on the button, or specify a TBitmap object at runtime.

The lower left pixel of the bitmap is reserved for the “transparent” color. Any pixel in the bitmap which matches that lower left pixel will be transparent if the GlyphTransparentColor property is clDefault.

GlyphTransparentColor Property

Determines which color of the bitmap is to be transparent when the bitmap is drawn. Read-write.

```
published
  property GlyphTransparentColor: TColor default clDefault;
```

Use GlyphTransparentColor to determine how to draw the bitmap transparently. If you want to use the color bottom leftmost pixel as transparent, set GlyphTransparentColor to clDefault.

HyperlinkType Property

Sets or returns the type of hyperlink associated with the specified command bar button. Read-write.

```
published
  property HyperlinkType: TadxMsoCommandBarButtonHyperlinkType
    default adxMsoCommandBarButtonHyperlinkNone;
```

ShortcutText Property

Returns or sets the shortcut key text displayed next to a button control when the button appears on a menu, submenu, or shortcut menu. Read-write.

```
published
  property ShortcutText: WideString;
```

State Property

Returns or sets the appearance of a command bar button controls. Read-write.

```
published
  property State: TadxMsoButtonState default adxMsoButtonUp;

type
  TadxMsoButtonState = (adxMsoButtonUp, adxMsoButtonDown, adxMsoButtonMixed);
```

Style Property

Returns or sets the way a command bar button control is displayed. Read-write.

```
published
  property Style: TadxMsoButtonStyle;

type
TadxMsoButtonStyle = (adxMsoButtonAutomatic, adxMsoButtonIcon,
  adxMsoButtonCaption, adxMsoButtonIconAndCaption,
  adxMsoButtonIconAndWrapCaption, adxMsoButtonIconAndCaptionBelow,
  adxMsoButtonWrapCaption, adxMsoButtonIconAndWrapCaptionBelow);
```

OnClick Event

Occurs when the user clicks the button. Read-write.

```
published
  property OnClick: TNotifyEvent;
```

Use the OnClick event handler to respond when the user clicks the button.

TadxCommandBarComboBox

A wrapper over the CommandBarComboBox interface. Unit: adxAddIn

```
type
  TadxCommandBarComboBox = class(TadxCustomCommandBarComboBox)
  public
    property List;
    property ListCount;
    property ListIndex;
  published
    property DropdownLines;
    property DropdownWidth;
    property ListHeaderCount;
    property Style;
    property Text;
    property Items;
    property OnChange: TNotifyEvent;
  end;
```

TadxCommandBarComboBox is a wrapper over the command bar combo box interface. Use this class to add a new command bar combo box to the command bar of the host application or to connect to the existing command bar combo box. TadxCommandBarComboBox is an item of the TadxCommandBar.Controls collection. Add a combo box via the TadxCommandBar.Controls.Add method.

DropdownLines Property

Returns or sets the number of lines in a command bar combo box control. Read-write.

```
published
  property DropdownLines: SYSINT default 0;
```

The combo box control must be a custom control and it must be a drop-down list box or a combo box. If this property is set to 0 (zero), the number of lines in the control will be based on the number of items in the list.

An error occurs if you attempt to set this property for a combo box control that's an edit box (TadxCommandBarEdit) or a built-in combo box control.

DropdownWidth Property

Returns or sets the width (in pixels) of the list for the specified command bar combo box control. Read-write.

```
published
  property DropdownWidth: SYSINT default 0;
```

If this property is set to -1, the width of the list is based on the length of the longest item in the combo box list. If this property is set to -0, the width of the list is based on the width of the control.

An error occurs if you attempt to set this property for a built-in control.

Items Property

Provides access to the list of items (strings) in the list portion of the combo box. Read-write.

```
published
  property Items: TStrings;
```

This is a design-time property. The combo box list will be initialized with contents of the Items property.

List Property

Returns or sets an item in the command bar combo box controls. Read-write.

```
public
  property List [Index: SYSINT]: WideString;
```

ListCount Property

Returns the number of list items in a command bar combo box control. Read-only.

```
public
  property ListCount: SYSINT;
```

ListHeaderCount Property

Returns or sets the number of list items in the command bar combo box control that appears above the separator line. Read-write.

```
published
  property ListHeaderCount: SYSINT default -1;
```

A ListHeaderCount property value of -1 indicates that there's no separator line in the combo box control.

ListIndex Property

Returns or sets the index number of the selected item in the list portion of the command bar combo box controls. Read-only.

```
public
  property ListIndex: SYSINT;
```

If nothing is selected in the list, this property returns zero. Setting the ListIndex property causes the specified control to select the given item and execute the appropriate action in the application.

This property fails when applied to controls other than the TadxCommandBarComboBox or the TadxCommandBarDropdownList controls.

Style Property

Returns or sets the way a command bar combo box control is displayed. Read-write.

```
published
  property Style: TadxMsoComboStyle default adxMsoComboNormal;

type
  TadxMsoComboStyle = (adxMsoComboNormal, adxMsoComboLabel);
```

Text Property

Returns or sets the text in the display or edit portion of the command bar combo box control. Read-write.

```
published
  property Text: WideString;
```

OnChange Event

Occurs when the end user changes the value of a command bar combo box. Read-write.

```
published
  property OnChange: TNotifyEvent;
```

Use the OnChange event handler to respond when the user changes the value of the TadxCommandBarComboBox.

TadxCommandBarControl

A wrapper over the Office2000.CommandBarControl interface. Unit: adxAddIn

```
type
  TadxCommandBarControl = class(TCollectionItem)
  protected
    procedure DefineProperties(Filer: TFiler); override;
  public
    constructor Create(Collection: TCollection); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property AsButton: TadxCommandBarButton;
    property AsEdit: TadxCommandBarEdit;
    property AsDropdownList: TadxCommandBarDropDownList;
    property AsComboBox: TadxCommandBarComboBox;
    property AsPopup: TadxCommandBarPopup;
    property BuiltIn: WordBool;
    property OfficeIndex: SYSINT;
    property InstanceID: Integer;
    property Left: SYSINT;
    property OLEUsage: MsoControlOLEUsage;
    property Parent: CommandBar;
    property Top: SYSINT;
    property Type_: MsoControlType;
    property IsPriorityDropped: WordBool;
    property OnAction: WideString;
    property CommandBar: TadxCommandBar;
    property DefaultInterface: CommandBarControl;
  published
    property BeginGroup: WordBool default False;
    property Caption: WideString;
    property DescriptionText: WideString;
    property Enabled: WordBool default True;
    property Height: SYSINT default 0;
    property HelpContextID: SYSINT default 0;
```

```
    property HelpFile: WideString;
    property Parameter: WideString;
    property Priority: SYSINT default 0;
    property OfficeID: SYSINT default 1;
    property OfficeTag: WideString;
    property TooltipText: WideString;
    property Visible: WordBool default True;
    property Width: SYSINT default 0;
    property Before: Integer default 0;
    property Temporary: boolean default False;
    property Tag: Integer default 0;
  end;
```

TadxCommandBarControl is a wrapper over the command bar control interface. This is a base class to all command bar controls supported by Add-in Express. All command bar controls are items of the TadxCommandBar.Controls collection.

Use instances of this class to add any built-in controls of the add-in host application to your command bar. You are also to specify the ID of the built-in control via the OfficeID property. You cannot handle events of the built-in control through TadxCommandBarControl.

AfterID Property

An ID of the built-in command bar controls. Read-write.

```
published
  property AfterID: Integer;
```

An ID of the command bar control after which the control will be inserted.

AsButton Property

Represents a command bar control as a command bar button. Read-only.

```
public
    property AsButton: TadxCommandBarButton;
```

The TadxCommandbarControl class is an ancestor of all command bar control classes such as command bar buttons, edit boxes, etc. Use the AsButton property to get access to the control as a command bar button. Returns nil, if the control is not a button.

AsComboBox Property

Represents a command bar control as a command bar combo box. Read-only.

```
public
    property AsComboBox: TadxCommandBarComboBox;
```

The TadxCommandbarControl class is an ancestor of all command bar control classes such as command bar buttons, edit boxes, etc. Use the AsComboBox property to get access to the control as a command bar combo box. Returns nil, if the control is not a combo box.

AsDropdownList Property

Represents a command bar control as a command bar dropdown list. Read-only.

```
public
    property AsDropdownList: TadxCommandBarDropDownList;
```

The TadxCommandbarControl class is an ancestor of all command bar control classes such as command bar buttons, edit boxes, etc. Use the AsDropDownList property to get access to the control as a command bar dropdown list. Returns nil, if the control is not a dropdown list.

AsEdit Property

Represents a command bar control as a command bar edit box. Read-only.

```
public
    property AsEdit: TadxCommandBarEdit;
```

The TadxCommandBarControl class is an ancestor of all command bar control classes such as command bar buttons, edit boxes, etc. Use the AsEdit property to get access to the control as a command bar edit box. Returns nil, if the control is not an edit box.

AsPopup Property

Represents a command bar control as a command bar pop-up. Read-only.

```
public
    property AsPopup: TadxCommandBarPopup;
```

The TadxCommandbarControl class is an ancestor of all command bar control classes such as command bar buttons, edit boxes, etc. Use the AsPopup property to get access to the control as a command bar pop-up. Returns nil, if the control is not a pop-up.

Before Property

A number that indicates the position of the new control on the command bar. Read-write.

```
published
  property Before: Integer;
```

A number that indicates the position of the new control on the command bar. The new control will be inserted before the control at this position. If this argument is 0, the control is added at the end of the specified command bar.

BeforeID Property

An ID of the built-in command bar controls. Read-write.

```
published
  property BeforeID: Integer;
```

An ID of the command bar control before which the control will be inserted.

BeginGroup Property

True if the specified command bar control appears at the beginning of a group of controls on the command bar. Read-write.

```
published
  property BeginGroup: WordBool default False;
```

BuiltIn Property

True if the command bar control is a built-in control of the container application. False if it's a custom control, or if it's a built-in control whose OnAction property has been set. Read-only.

```
public
  property BuiltIn: WordBool;
```

Caption Property

Returns or sets the caption text for a command bar control.

```
published
  property Caption: WideString;
```

The Caption property also is used as TooltipText if TooltipText is empty.

CommandBar Property

Returns the owner (TadxCommandBar) of the command bar control. Read-only.

```
public
  property CommandBar: TadxCommandBar;
```

DefaultInterface Property

Returns a reference to the Office2000.CommandBarControl interface. Read-only.

```
public
  property DefaultInterface: CommandBarControl;
```

Use DefaultInterface to access the default interface for this interfaced object.

DescriptionText Property

Returns or sets the description for a command bar control. Read-only.

```
published
  property DescriptionText: WideString;
```

The description is not displayed to the user, but it can be useful for documenting the behavior of the control for other developers.

Enabled Property

Enables or disables the control. Read-write.

```
published
  property Enabled: WordBool;
```

True if the command bar control is enabled.

Height Property

Returns or sets the height of command bar controls. Read-write.

```
published
  property Height: SYSINT;
```

HelpContextID Property

Returns or sets the Help context Id number for the Help topic attached to the command bar control. Read-write.

```
published
  property HelpContextID: SYSINT;
```

To use this property, you must also set the HelpFile property. Help topics respond to Shift+F1

HelpFile Property

Returns or sets a file name for the Help topic attached to the command bar control. Read-write.

```
published
  property HelpFile: WideString;
```

To use this property, you must also set the HelpContextID property. Help topics respond to the user pressing Shift+F1.

IsPriorityDropped Property

True if the control is currently dropped from the menu or toolbar based on usage statistics and layout space. Read-only.

```
public
  property IsPriorityDropped: WordBool;
```

Note that this is not the same as the control's visibility, as set by the Visible property. A control with Visible set to True, will not be immediately visible on a Personalized Menu or Toolbar if IsPriorityDropped is True.

Left Property

Returns the horizontal position of the specified command bar control (in pixels) relative to the left edge of the screen. Read-only.

```
public
  property Left: SYSINT;
```

OfficeID Property

An integer that specifies the built-in control. If the value of this argument is 1, a blank custom control of the specified type will be added to the command bar. Read-write.

```
published
  property OfficeID: SYSINT default 1;
```

OfficeIndex Property

Returns the index number for an object in the host application command bar controls collection. Read-only.

```
public
property OfficeIndex: SYSINT;
```

The position of the first command bar control is 1. Separators are not counted in the CommandBarControls collection.

OfficeTag Property

A unique value that identifies a control in the command bars system of the host application. Read-write.

```
published
  property OfficeTag: WideString;
```

Do not change this property at design-time. It is generated automatically.

OLEUsage Property

Returns or sets the OLE client and the OLE server roles in which a command bar control will be used when two host applications are merged. Read-write.

```
public
  property OLEUsage: MsoControlOLEUsage;

// Constants for enum MsoControlOLEUsage
type
  MsoControlOLEUsage = TOleEnum;
const
  msoControlOLEUsageNeither = $00000000;
  msoControlOLEUsageServer = $00000001;
  msoControlOLEUsageClient = $00000002;
  msoControlOLEUsageBoth = $00000003;
```

OlExplorerItemTypes Property

Returns or sets types of the Outlook folders. Read-write.

```
published
  property olExplorerItemTypes: TadxOLItemTypes default [];
```

See OlItemTypeAction property.

OlInspectorItemTypes Property

Returns or sets types of the Outlook items. Read-write.

```
published
  property olInspectorItemTypes: TadxOLItemClasses default [];
```

See OlItemTypeAction property.

OlItemTypeAction Property

Returns or sets types of the action that will be done with the control. Read-write.

```
published
  property olItemTypeAction: TadxOLItemTypeAction default adxOLActionNone;

type
  TadxOLItemTypeAction = (adxOLActionNone, adxOLActionShow, adxOLActionEnable);
```

- AdxOLActionShow adds and shows the command bar control for the folder or item types that are specified by OlExplorerItemTypes or OlInspectorItemTypes, and adds and hides the command bar control for the folder or item types that are not specified by OlExplorerItemTypes or OlInspectorItemTypes.
- AdxOLActionEnable adds and enables the command bar control for the folder or item types that are specified by OlExplorerItemTypes or OlInspectorItemTypes, adds and disables the command bar control for the folder or item types that are not specified by OlExplorerItemTypes or OlInspectorItemTypes.
- AdxOLActionNone adds and enables the command bar control.

OnAction Property

Returns or sets the name of a Visual Basic procedure that will run when the user clicks or changes the value of a command bar control. Read-write.

```
public
  property OnAction: WideString;
```

The container application determines whether the value is a valid macro name. For all add-in controls OnAction is set to the ProgID of the add-in. Do not change this property.

Parameter Property

An additional property that is similar to the Tag property value. Read-write.

```
published
  property Parameter: WideString;
```

For built-in controls, this argument is used by the container application to run the command. For custom controls, you can use this argument to send information to Visual Basic procedures, or you can use it to store information about the control.

Parent Property

Returns a reference to the parent CommandBar interface for the command bar control. Read-only.

```
public
  property Parent: CommandBar;
```

Priority Property

Returns or sets the priority of a command bar control. Read-write.

```
published
  property Priority: SYSINT;
```

A control's priority determines whether the control can be dropped from a docked command bar if the command bar controls can't fit in a single row. Valid priority numbers are 0 (zero) through 7. A priority of 1 means that the control cannot be deleted from a toolbar. Other priority values are ignored. The Priority property is not used by command bar controls that are menu items.

Tag Property

Stores an integer value as part of a component.

```
published
  property Tag: Longint;
```

Temporary Property

Determines if a control is temporary. Read-write.

```
published
  property Temporary: boolean;
```

True to make the new control temporary. Temporary controls are automatically deleted when the container application is closed. The default value is False.

TooltipText Property

Returns or sets the text displayed in a command bar control's ToolTip. Read-write.

```
published
  property TooltipText: WideString;
```

By default, the value of the Caption property is used as ToolTip.

Top Property

Returns the distance (in pixels) from the top edge of the specified command bar control to the top edge of the screen. Read-only.

```
public
  property Top: SYSINT;
```

Type_ Property

Returns the type of command bar control. Read-only.

```
public
  property Type_: MsoControlType;

// Constants for enum MsoControlType
type
  MsoControlType = TOleEnum;
const
  msoControlCustom = $00000000;
  msoControlButton = $00000001;
  msoControlEdit = $00000002;
  msoControlDropdown = $00000003;
  msoControlComboBox = $00000004;
  msoControlButtonDropdown = $00000005;
  msoControlSplitDropdown = $00000006;
  msoControlOCXDropdown = $00000007;
  msoControlGenericDropdown = $00000008;
  msoControlGraphicDropdown = $00000009;
  msoControlPopup = $0000000A;
  msoControlGraphicPopup = $0000000B;
  msoControlButtonPopup = $0000000C;
  msoControlSplitButtonPopup = $0000000D;
```

```
  msoControlSplitButtonMRUPopup = $0000000E;
  msoControlLabel = $0000000F;
  msoControlExpandingGrid = $00000010;
  msoControlSplitExpandingGrid = $00000011;
  msoControlGrid = $00000012;
  msoControlGauge = $00000013;
  msoControlGraphicCombo = $00000014;
  msoControlPane = $00000015;
  msoControlActiveX = $00000016;
```

Visible Property

True if the command bar control is visible. Read-write.

```
published
  property Visible: WordBool;
```

Width Property

Returns or sets the width (in pixels) of the command bar control. Read-write.

```
published
  property Width: SYSINT;
```

TadxCommandBarControls

TadxCommandBarControls is a container for TadxCommandBarControl objects. Unit: adxAddIn

```
type
  TadxCommandBarControls = class(TOwnedCollection)
  public
    function Add(AType: TadxControlType): TadxCommandBarControl; overload;
    function Add(AType: TadxControlType; const ATag: WideString;
      ABefore: Integer; ATemporary: boolean): TadxCommandBarControl; overload;
    procedure DeleteControl(Index: integer; Temporary: boolean);
    function ItemByTag(const ATag: Integer): TadxCommandBarControl;
    property Items[Index: Integer]: TadxCommandBarControl; default;
  end;
```

This collection is used to add, to manage and to store command bar controls (TadxCommandBarControl) of the TadxCommandBar or TadxCommandBarPopup. This is a polymorphic collection the items of which can be command bar buttons, command bar edit boxes, etc. (all descendants of the TadxCommandBarControl class).

Items Property

Lists the items in the collection.

```
public
  property Items[Index: Integer]: TadxCommandBarControl; default;
```

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of the collection item. It represents the position of the item in the collection.

Use the TadxCommandBarControl.AsXXX methods to cast to the corresponding TadxCommandBarControl descendants.

Add Method

Adds a new item to the collection.

```
public
```

```
  function Add(AType: TadxControlType): TadxCommandBarControl; overload;
  function Add(AType: TadxControlType; const ATag: WideString;
    ABefore: Integer; ATemporary: boolean): TadxCommandBarControl; overload;
```

Use these methods to add a new command bar control to the command bar or to the command bar pop-up. These methods allow adding buttons, edit boxes, etc. with the type specified by the AType parameter. The second method allows initializing the Tag, Before and Temporary properties of the added control via the corresponding parameters.

Note. Do not use the standard Add method of TCollection.

DeleteControl Method

Deletes an item with the Index specified by the Index parameter.

```
public
  procedure DeleteControl(Index: integer; Temporary: boolean);
```

Deletes an item from the collection, destroys it and removes the corresponding control from the command bar of the host application.

ItemByTag Method

Searches for a control with the Tag property specified by the ATag parameter.

```
public
  function ItemByTag(const ATag: Integer): TadxCommandBarControl;
```

TadxCommandBarDropdownList

A wrapper over the CommandBarComboBox interface. Unit: adxAddIn

```
type
  TadxCommandBarDropdownList = class(TadxCustomCommandBarComboBox)
  public
    property List;
    property ListCount;
    property ListIndex;
  published
    property DropdownLines;
    property DropdownWidth;
    property ListHeaderCount;
    property Style;
    property Text;
    property Items;
    property OnChange;
  end;
```

TadxCommandBarDropdownList is a wrapper over the command bar combo box interface. Use this class to add a new command bar dropdown list to the command bar of the host application or to connect to the existing command bar dropdown list. TadxCommandBarDropdownList is an item of the TadxCommandBar.Controls collection. Add a dropdown list via the TadxCommandBar.Controls.Add method.

DropdownLines Property

Returns or sets the number of lines in a command bar dropdown list control. Read-write.

```
published
  property DropdownLines: SYSINT default 0;
```

The dropdown list control must be a custom control and it must be a drop-down list box or a combo box. If this property is set to 0 (zero), the number of lines in the control will be based on the number of items in the list.

An error occurs if you attempt to set this property for a dropdown list control that's an edit box (TadxCommandBarEdit) or a built-in combo box control.

DropdownWidth Property

Returns or sets the width (in pixels) of the list for the specified command bar dropdown list control. Read-write.

```
published
  property DropdownWidth: SYSINT default 0;
```

If this property is set to -1, the width of the list is based on the length of the longest item in the dropdown list. If this property is set to 0, the width of the list is based on the width of the control.

An error occurs if you attempt to set this property for a built-in control.

Items Property

Provides access to the list of items (strings) in the list portion of the dropdown list. Read-write.

```
published
  property Items: TStrings;
```

This is a design-time property. The dropdown list will be initialized with the contents of the Items property.

List Property

Returns or sets an item in the command bar dropdown list control. Read-write.

```
public
  property List [Index: SYSINT]: WideString;
```

ListCount Property

Returns the number of list items in the command bar dropdown list control. Read-only.

```
public
  property ListCount: SYSINT;
```

ListHeaderCount Property

Returns or sets the number of list items in the command bar dropdown list control that appears above the separator line. Read-write.

```
published
  property ListHeaderCount: SYSINT default -1;
```

A ListHeaderCount property value of -1 indicates that there's no separator line in the dropdown list control.

ListIndex Property

Returns or sets the index number of the selected item in the list portion of the command bar dropdown list control. Read-only.

```
public
  property ListIndex: SYSINT;
```

If nothing is selected in the list, this property returns zero. Setting the ListIndex property causes the specified control to select the given item and execute the appropriate action in the application.

This property fails when applied to controls other than TadxCommandBarComboBox or TadxCommandBarDropdownList.

Style Property

Returns or sets the way the command bar dropdown list control is displayed. Read-write.

```
published
  property Style: TadxMsoComboStyle default adxMsoComboNormal;

type
  TadxMsoComboStyle = (adxMsoComboNormal, adxMsoComboLabel);
```

Text Property

Returns or sets the text in the display or edit portion of the command bar dropdown list control. Read-write.

```
published
  property Text: WideString;
```

OnChange Event

Occurs when the end user changes the value of the command bar dropdown list. Read-write.

```
published
  property OnChange: TNotifyEvent;
```

Use the OnChange event handler to respond when the user changes the value of the TadxCommandBarDropdownList.

TadxCommandBarEdit

A wrapper over the CommandBarComboBox interface. Unit: adxAddIn

```
type
  TadxCommandBarEdit = class(TadxCustomCommandBarComboBox)
  published
    property Style;
    property Text;
    property OnChange;
  end;
```

TadxCommandBarEdit is a wrapper over the command bar combo box interface. Use this class to add a new command bar edit box to the command bar of the host application or to connect to the existing command bar edit box. TadxCommandBarEdit is an item of the TadxCommandBar.Controls collection. Add an edit box via the TadxCommandBar.Controls.Add method.

Style Property

Returns or sets the way a command bar edit box control is displayed. Read-write.

```
published
  property Style: TadxMsoComboStyle default adxMsoComboNormal;

type
  TadxMsoComboStyle = (adxMsoComboNormal, adxMsoComboLabel);
```

Text Property

Returns or sets the text in the display or edit portion of the command bar edit box control. Read-write.

```
published
  property Text: WideString;
```

OnChange Event

Occurs when the end user changes the value of a command bar edit box. Read-write.

```
published
```

```
  property OnChange: TNotifyEvent;
```

Use the OnChange event handler to respond when the user changes the value of the TadxCommandBarEdit.

TadxCommandBarPopup

A wrapper over the Office2000.CommandBarPopup interface. Unit: adxAddIn

```
type
  TadxCommandBarPopup = class(TadxCommandBarControl)
  public
    function ControlByCaption(const ACaption: WideString):
      TadxCommandBarControl;
    function ControlByTag(const ATag: Integer): TadxCommandBarControl;
    function ControlByOfficeTag(const AOfficeTag: WideString):
      TadxCommandBarControl;
    property DefaultInterface: CommandBarPopup;
  published
    property Controls: TadxCommandBarControls;
    property OfficeID;
  end;
```

TadxCommandBarPopup is a wrapper over the command bar pop-up interface. Use this class to add a new command bar pop-up to the command bar of the host application or to connect to the existing command bar pop-up. TadxCommandBarPopup is an item of the TadxCommandBar.Controls collection. Add an edit box via the TadxCommandBar.Controls.Add method.

DefaultInterface Property

Returns a reference to the Office2000.CommandBarpopup interface. Read-only.

```
public
  property DefaultInterface: CommandBarPopup;
```

Use DefaultInterface to access the default interface for this interfaced object.

Controls Property

Controls of the command bar pop-up. Read-write.

```
published
  property Controls: TadxCommandBarControls
```

Controls are a list of the pop-up's child controls. These controls are the buttons, the edit and combo boxes, and the pop-ups that appear on the pop-up.

OfficeID Property

An integer that specifies a built-in control. If the value of this argument is 1, a blank custom control of the specified type will be added to the command bar. The value of the OfficeID property for all custom controls is 1. Read-write.

```
public
  property OfficeID: SYSINT default 1;
```

ControlByCaption Method

Searches for a control with the specified Caption property within the Controls collection.

```
public
  function ControlByCaption(const ACaption: WideString): TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the caption specified by the ACaption parameter. Returns nil if the control isn't found.

ControlByOfficeTag Method

Searches for a control with the specified OfficeTag property within the Controls collection.

```
public
  function ControlByOfficeTag(const AOfficeTag: WideString):
    TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the OfficeTag specified by the AOfficeTag parameter. Returns nil if the control isn't found.

ControlByTag Method

Searches for a control with the specified Tag property within the Controls collection.

```
public
```

```
function ControlByTag(const ATag: Integer): TadxCommandBarControl;
```

Returns the first founded TadxCommandBarControl with the Tag specified by the ATag parameter. Returns nil if the control isn't found.

TadxCustomCommandBarComboBox

A base wrapper over the Office2000.CommandBarComboBox interface. Unit: adxAddIn

```
type
  TadxCustomCommandBarComboBox = class(TadxCommandBarControl)
  public
    property DefaultInterface: CommandBarComboBox;
  end;
```

TadxCustomCommandBarComboBox is an ancestor of TadxCommandBarComboBox, TadxCommandBarDropdownList and TadxCommandBarEdit.

DefaultInterface Property

Provides access to the default interface for the command bar combo box. Read-only.

```
public
  property DefaultInterface: CommandBarComboBox;
```

Returns a reference to the Office2000.CommandBarComboBox interface of the command bar combo box instance.

TadxFactory

A COM add-ins factory. Unit: adxAddIn

```
type
  TadxFactory = class(TAutoObjectFactory)
  public
    property FilePath: string;
    property RegistryKey[Index: TadxOfficeHostApp]: string;
  end;
```

TadxFactory is a descendant of TAutoObjectFactory. It implements all the specific subtleties of MS Office COM Add-in registering. When creating your add-ins, you must create an instance of this class and not TAutoObjectFactory. You create the instance by calling the TadxFactory.Create constructor.

FilePath Property

Returns a path to the add-in binary file. Read-only.

```
public
    property FilePath: string;
```

RegistryKey Property

Returns the registry key of the add-in registration record. Read-only.

```
public
  property RegistryKey[Index: TadxOfficeHostApp]: string;
```

The location of the key is “\Software\Microsoft\Office\%HOST%\AddIns\%PROGID%”, where HOST - the name of MS Office application and PROGID is the ProgID of the class factory.

TadxOlExplorerCommandbar

An Outlook Explorer specific command bar. Unit: adxAddIn

```
type
  TadxOlExplorerCommandbar = class(TadxCommandBar)
  published
    property FolderName: WideString;
    property FolderNames: TStrings;
    property ItemTypes: TadxOLItemTypes default [adxOLMailItem];
    property Position default adxMsoBarTop; // overrided
    property Temporary default True; // overrided
  end;
```

TadxOlExplorerCommandBar is a wrapper over the command bar interface and is a container for all command bar controls. Use this component to add a new command bar to the command bars system of the Outlook Explorer object or to connect to the existing command bar.

FolderName Property

Returns or sets a name of the Outlook folder. Read-write.

```
published
  property FolderName: WideString;
```

The command bar will be added to the Explorer command bars when Explorer shows the specified folder. Use the full path to the folder for this property.

FolderNames Property

Returns or sets names of the Outlook folders. Read-write.

```
published
  property FolderNames: TStrings;
```

The command bar will be added to the Explorer command bars when Explorer shows the specified folders. Use the full path to the folder for this property.

ItemTypes Property

Returns or sets types of the Outlook folders. Read-write.

```
published
  property ItemTypes: TadxOLItemTypes default [adxOLMailItem];
```

The command bar will be added to the Explorer command bars when Explorer shows a folder of the specified types.

TadxOlInspectorCommandbar

An Outlook Inspector specific command bar. Unit: adxAddIn

```
type
  TadxOlInspectorCommandbar = class(TadxCommandBar)
  published
    property FolderName: WideString;
    property FolderNames: TStrings;
    property ItemTypes: TadxOLItemTypes default [adxOLMailItem];
    property Position default adxMsoBarTop; // overrided
    property Temporary default True; // overrided
  end;
```

TadxOlInspectorCommandBar is a wrapper over the command bar interface and is a container for all command bar controls. Use this component to add a new command bar to the command bars system of the Outlook Inspector object or to connect to the existing command bar.

FolderName Property

Returns or sets a name of the Outlook folder. Read-write.

```
published
  property FolderName: WideString;
```

The command bar will be added to the Inspector command bars when Inspector shows an item of the specified folder. Use the full path to the folder for this property.

FolderNames Property

Returns or sets names of the Outlook folders. Read-write.

```
published
  property FolderNames: TStrings;
```

The command bar will be added to the Explorer command bars when Inspector shows an item of the specified folders. Use the full path to the folder for this property.

ItemTypes Property

Returns or sets types of the Outlook items. Read-write.

```
published
  property ItemTypes: TadxOLItemTypes default [adxOLMailItem];
```

The command bar will be added to the Inspector command bars when Inspector shows an item of the specified types.

TadxRTDFactory

An RTD Servers factory. Unit: adxRTDServ

```
type
  TadxRTDFactory = class(TAutoObjectFactory)
  public
    property FilePath: string;
  end;
```

TadxRTDFactory is a descendant of TAutoObjectFactory. It implements all the specific subtleties of Excel Real-Time Data Servers registering. When creating your RTD Servers you must create an instance of this class and not TAutoObjectFactory. You create the instance by calling the TadxRTDFactory.Create constructor.

FilePath Property

Returns a path to the RTD Server binary file. Read-only.

```
public
  property FilePath: string;
```

TadxRTDServer

TadxRTDServer is an interfaced class that implements the IRTDServer interface required by RTD Servers. Unit: adxRTDServ

```
type
  TadxRTDServer = class(TAutoObject, IRtdServer)
  public
    property Factory: TadxRTDFactory read GetFactory;
    property RTDModule: TadxXLRTDServerModule;
  end;
```

The TadxRTDServer class is a subsidiary class that implements the RTD Server interface. You needn't call its methods and properties.

Factory Property

Specifies the class factory object for the TadxRTDServer class. Read-only.

```
public
  property Factory: TadxRTDFactory;
```

Factory is the class factory that is used to instantiate a TadxRTDServer.

RTDServerModule Property

Specifies the server module object for the TadxRTDServer class. Read-only.

```
public
  property RTDModule: TadxXLRTDServerModule;
```

TadxRTDTopic

A topic of the RTD Server. Unit: adxRTDServ

```
type
  TadxRTDTopic = class(TComponent)
  public
    property TopicID: Integer;
    property Strings[Index: Integer]: WideString; default;
    property Text: string;
  published
    property Enabled: boolean default True;
    property DefaultValue: OLEVariant;
    property String01: WideString;
    property String02: WideString;
    property String03: WideString;
    property String04: WideString;
    property String05: WideString;
    property String06: WideString;
    property String07: WideString;
    property String08: WideString;
    property String09: WideString;
    property String10: WideString;
    property String11: WideString;
    property String12: WideString;
    property String13: WideString;
    property String14: WideString;
    property String15: WideString;
    property String16: WideString;
    property String17: WideString;
    property String18: WideString;
    property String19: WideString;
    property String20: WideString;
    property String21: WideString;
    property String22: WideString;
    property String23: WideString;
```

```
    property String24: WideString;
    property String25: WideString;
    property String26: WideString;
    property String27: WideString;
    property String28: WideString;
    property UseStoredValue: boolean default False;
    property OnRefreshData: TadxOnRefreshData;
    property OnDisconnect: TNotifyEvent;
  end;
```

TadxRTDTopic implements the RTD Server topic functionality. Add this component to the RTD Server module, specify topic names and handle topic events.

DefaultValue Property

A default value of the topic. Read-write.

```
published
  property DefaultValue: OLEVariant;
```

This value will be used for initialization Excel cell if UseStoredValue is False.

Enabled Property

Represents a topic state. Read-write.

```
published
  property Enabled: Boolean default True;
```

Use to enable or disable the topic. When the topic is disabled, the OnRTDTopicEvent event isn't arising.

Text Property

Returns a concatenation of the StringXX properties. Read-only.

```
public
    property Text: string;
```

TopicID Property

Returns a unique ID of the RTD function. Read-only.

```
public
  property TopicID: Integer;
```

TopicID represents the arbitrary, unique value automatically assigned by Excel. If -1 is returned, the topic is unused. The default value is -1.

String01..String28 Properties

Topic names. Read-write.

```
published
  property String01: WideString;
  .
  .
  property String28: WideString;
```

The StringXX properties are strings identifying the topic or passing parameters.

Strings Property

Returns or sets the topic strings (StringXX) as an indexed property. Read-write.

```
public
    property Strings[Index: Integer]: WideString; default;
```

UseStoredValue Property

Represent topic behavior.

```
published
  property UseStoredValue: boolean default False;
```

OnDisconnectTopic Event

Occurs when Excel is disconnected from the RTD server. Read-write.

```
published
  property OnDisconnectTopic: TNotifyEvent;
```

Excel can use this method to take some type of action before it loses its RTD connection.

OnRefreshData Event

Occurs when Excel pulls new values from the RTD Server (also known as topic refresh). Read-write.

```
published
  property OnRTDTopic: TadxOnRefreshData;

type
  TadxOnRefreshData = function(Sender: TObject): OleVariant of object;
```

Use the OnRefreshData event handler to pass a new data of the topic to Excel.

TadxXLRTDServerModule

TadxXLRTDServerModule centralizes handling of the RTD topics. Unit: adxRTDServ

```
type
  TadxXLRTDServerModule = class(TDataModule)
  public
    procedure UpdateTopics;
  published
    property Interval: Longword default 5000;
    property OnRTDInitialize: TNotifyEvent;
    property OnRTDFinalize: TNotifyEvent;
    property OnError: TadxRTDOnError;
  end;
```

TadxXLRTDServerModule is a descendant of TDataModule and is used to provide True RAD for add-in development. Use a TadxXLRTDServerModule object in an application to provide a location for centralized handling of the RTD topic components. XLRTDServerModules are not limited to the ADX components; they can also contain other nonvisual components, such as TTimer, TOpenDialog, or TImageList). At design time a TadxXLRTDServerModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them.

Interval Property

Returns or sets an refresh-data interval. Read-write.

```
  published
    property Interval: Longword default 5000;
```

UpdateTopics Method

Refreshes all topics.

```
public
  procedure UnpadteTopics;
```

Use this method to refresh topics manually.

OnError Event

Occurs when an exception is raised. Read-write.

```
published
  property OnError: TadxRTDOnError;
type
  TadxRTDOnError = procedure (const E: SysUtils.Exception;
    var Handled: boolean) of object;
```

Allows handling an exception. By default, Add-in Express handles exceptions and shows an error message box. You can handle the exception yourself and avoid the error messages via Handled := True.

OnRTDFinalize Event

Occurs when an RTD server is disconnected from Excel. Read-write.

```
published
  property OnRTDFinalize: TNotifyEvent;
```

OnRTDInitialize Event

Occurs when an RTD server is connected to Excel. Read-write.

```
published
  property OnRTDInitialize: TNotifyEvent;
```

OnConnectData Event

Occurs when an Excel workbook that contains real-time data functions is opened or when a user types in a new formula which contains the RTD function. Read-write.

```
published
  property OnConnectData: TNotifyEvent;
type
  TadxRTDOnConnectData = procedure (const Topics: array of string) of object;
```

Topics is a string array identifying the topic.

TadxRecognizerFactory

A smart tag recognizer factory. Unit: adxSmartTag

```
type
  TadxRecognizerFactory = class(TAutoObjectFactory)
  public
    property FilePath: string;
  end;
```

TadxRecognizerFactory is a descendant of TAutoObjectFactory, and implements all the specific subtleties of smart tag recognizer registering.

FilePath Property

Returns a path to the smart tag library file. Read-only.

```
public
  property FilePath: string;
```

TadxActionFactory

A smart tag action factory. Unit: adxSmartTag

```
type
  TadxActionFactory = class(TadxRecognizerFactory)
  end;
```

TadxActionFactory is a descendant of TadxRecognizerFactory. It implements all the specific subtleties of smart tag action registering.

TadxRecognizerObject

TadxRecognizerObject is an interfaced class that implements the ISmartTagRecognizer (Office XP) and the ISmartTagRecognizer2 (Office 2003) interfaces required by the smart tag technology. Unit: adxSmartTag

```
type
  TadxRecognizerObject = class(TAutoObject, ISmartTagRecognizer,
ISmartTagRecognizer2)
  public
    property Factory: TadxRecognizerFactory;
  end;
```

The TadxRecognizerObject class is a subsidiary class that implements the smart tag interfaces. You needn't call its methods and properties.

Factory Property

Specifies the class factory object for the TadxRecognizerObject class. Read-only.

```
public
  property Factory: TadxRecognizerFactory;
```

Factory is the class factory that is used to instantiate a TadxRecognizerObject.

TadxActionObject

TadxActionObject is an interfaced class that implements the ISmartTagAction (Office XP) and the ISmartTagAction2 (Office 2003) interfaces required by the smart tag technology. Unit: adxSmartTag

```
type
  TadxActionObject = class(TAutoObject, ISmartTagAction, ISmartTagAction2)
  public
    property Factory: TadxActionFactory;
  end;
```

The TadxActionObject class is a subsidiary class that implements the smart tag interfaces. You needn't call its methods and properties.

Factory Property

Specifies the class factory object for the TadxActionObject class. Read-only.

```
public
  property Factory: TadxActionFactory;
```

Factory is the class factory that is used to instantiate a TadxActionObject.

TadxSmartTagAction

A class that corresponds to one of the smart tag context menu items. Unit: adxSmartTag

```
TadxSmartTagAction = class(TCollectionItem)
public
  constructor Create(Collection: TCollection); override;
  destructor Destroy; override;
  property Captions[LocaleID: Integer]: WideString;
published
  property Caption: WideString;
  property Name: WideString;
  property Tag: Integer default 0;
  property OnClick: TadxActionEvent;
end;
```

An instance of the TadxSmartTagAction class is an item of the TadxSmartTag.Actions collection. It provides a way to handling events of the smart tag context menu via OnClick event.

To add a new menu item to the smart tag context menu you should add a new item to the TadxSmartTag.Actions collection, specify its caption, and add the OnClick event handler.

Caption Property

A caption of the corresponding item of the smart tag context menu. Read-write.

```
published
  property Caption: WideString;
```

The caption for any Locale that isn't specified in the Captions property.

Note. For Office 2003 only. If you want the menu to cascade, use "///" in the action captions. Text that precedes the "///" will be the name for the cascade menu, and everything after "///" will be the caption of the action. Note that "/" can still appear in menu items, so there are no conflicts with actions that include Web addresses in their names. Multiple levels of nesting are also supported.

Captions Property

A list of captions that depend on Locale. Read-write.

```
  public
    property Captions[LocaleID: Integer]: WideString;
```

Note. For Office 2003 only. If you want the menu to cascade, use "///" in the action captions. Text that precedes the "///" will be the name for the cascade menu, and everything after "///" will be the caption of the action. Note that "/" can still appear in menu items, so there are no conflicts with actions that include Web addresses in their names. Multiple levels of nesting are also supported.

Name Property

The unique identifiers of the smart tag action. Read-write.

```
  published
    property Name: WideString;
```

For internal use only. Do not change this property, its value is generated automatically.

Tag Property

Stores an integer value as part of the instance. Read-write.

```
  published
    property Tag: Integer default 0;
```

Tag has no predefined meaning. The Tag property is provided for the convenience of developers.

OnClick Event

Occurs when the user clicks the corresponding item of the smart tag context menu. Read-write.

```
  published
    property OnClick: TadxActionEvent;

type
  TadxActionEvent = procedure(Sender: TObject; const AppName: WideString;
    const Target: IDispatch; const Text, Xml: WideString;
    LocaleID: SYSINT) of object;
```

Parameters:

Sender – the TadxSmartTagAction instance that fires this event.

AppName – a string that represents the ProgID of the calling application.

Target – a reference to the application-specific object that is responsible for calling the smart tag. In Excel, it is an Excel range object that points to the cell that the smart tag was attached to. In Word and PowerPoint, it is a range object that wraps around the block of text that the smart tag refers to. In Access, it is a control. However, note that this object may be any other kind of pointer to an object appropriate for the calling application. It could also be nil.

Text – a string that represents the text of the smart tag.

Xml – a string that is an XML representation of a smart tag.

LocaleID – is zero for Office XP. For Office 2003 it is a language identifier that corresponds to the user interface language of the calling application.

TadxSmartTagActions

TadxSmartTagActions is a container for TadxSmartTagAction objects. Unit: adxSmartTag

```
TadxSmartTagActions = class(TOwnedCollection)
public
  property Items[Index: Integer]: TadxSmartTagAction; default;
end;
```

Items Property

Returns an item by its index. Read-write.

```
public
  property Items[Index: Integer]: TadxSmartTagAction; default;
```

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of the collection item. It represents the position of the item in the collection.

TadxSmartTag

A component that centralizes all smart tag actions, implements smart tag properties, and recognizes smart tag phrases. Unit: adxSmartTag

```
TadxSmartTag = class(TComponent)
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  property Captions[LocaleID: Integer]: WideString;
published
  property Actions: TadxSmartTagActions;
  property Caption: WideString;
  property DownloadURL: WideString;
  property Kind: TadxKindType default ktList;
  property RecognizedMask: TEditMask;
  property RecognizedWords: TStrings;
  property OnActionNeeded: TNotifyEvent;
  property OnPropertyPage: TadxPropertyPageEvent;
  property OnRecognize: TadxRecognizeEvent;
end;
```

This component is a representation of Smart Tag. It recognizes smart tag phrases according to the way specified through the Kind property and the phrases specified via the RecognizedWords or RecognizedMask properties.

To create a new smart tag, add this component to the smart tag module, specify the smart tag caption, indicate the way of recognizing, add smart tag phrases or a mask, and define smart tag actions via the Actions collection.

Actions Property

Centralizes all smart tag actions. Read-write.

```
published
  property Actions: TadxSmartTagActions;
```

Describes the smart tag context menu and attaches each item of this menu to its items.

Caption Property

A caption of the smart tag context menu. Read-write.

```
  published
    property Caption: WideString;
```

The caption for any Locale that isn't specified in the Captions property. The smart tag caption will be shown as a caption of the smart tag menu.

Captions Property

A list of captions that depend on Locale. Read-write.

```
  public
    property Captions[LocaleID: Integer]: WideString;
```

Depends on Locale.

DownloadURL Property

A property that represents the URL address for the smart tag. Read-write.

```
  published
    property DownloadURL: WideString;
```

The DownloadURL property is useful if a document is sent to someone who does not have the necessary smart tag file installed on their computer. The user can follow the URL to download the necessary smart tag file.

Kind Property

Specifies the way of smart tag phrases recognition. Read-write.

```
  published
    property Kind: TadxKindType default ktList;
```

- ktList – specifies that the smart tag phrases are located in the RecognizedWords property.
- ktMask - specifies that the smart tag phrase is recognized by a mask (the RecognizedMask property).
- ktCustom – specifies that the smart tag phrases are recognized dynamically by the OnRecognize event.

RecognizedMask Property

Specifies the mask that represents what text is valid for the smart tag recognizer. Read-write.

```
  published
    property RecognizedMask: TEditMask;
```

Affected if the Kind property is ktMask.

RecognizedWords Property

Specifies the phrases that represent what text is valid for the smart tag recognizer. Read-write.

```
  published
    property RecognizedWords: TStrings;
```

Affected if the Kind property is ktList.

OnActionNeeded Event

Fires when a user clicks the smart tag button. Read-write.

```
  published
    property OnActionNeeded: TNotifyEvent;
```

Using this event you can create the smart tag menu dynamically. In this case you should add items you need to Actions collection.

OnPropertyPage Event

Occurs when the user clicks the Properties button on the AutoCorrect Options dialog box. Read-write.

```
  published
    property OnPropertyPage: TadxPropertyPageEvent;

type
  TadxPropertyPageEvent = procedure (Sender: TObject;
    LocaleID: Integer) of object;
```

LocaleID – a language identifier that corresponds to the user interface language of the calling application.

Note. This event doesn't fire on Office XP.

OnRecognize Event

Occurs when the host application attempts to recognize a character string as smart-tag actionable. Read-write.

```
  published
    property OnRecognize: TadxRecognizeEvent;

type
  TadxCommitSmartTagProc = procedure (const Index, Len: Integer);

  TadxRecognizeEvent = procedure (Sender: TObject; const Text: WideString;
    DataType: IF_TYPE; LocaleID: Integer;
    const CommitSmartTagProc: TadxCommitSmartTagProc;
    AppName: string; const TokenList: ISmartTagTokenList) of object;
```

Parameters:

- Sender – the instance that fires the event.
- Text – the text for recognizing.
- DataType – a type of the text (IF_TYPE_PARA = $08 or IF_TYPE_CELL = $10).
- LocaleID – a language identifier that corresponds to the user interface language of the calling application.
- CommitSmartTagProc – a pointer to callback procedure that should be called if Text is recognized as smart-tag actionable.
- AppName – the ProgID of the host application.
- TokenList – a reference to the interface that is a word enumerate helper. For Office 2003 only. Returns nil for Office XP.

Example:

```
procedure TSmartTagModule.adxSmartTag1Recognize(Sender: TObject;
  const Text: WideString; DataType: TOleEnum; LocaleID: Integer;
  const CommitSmartTagProc: TadxCommitSmartTagProc; AppName: String;
  const TokenList: ISmartTagTokenList);
begin
```

```
  if Pos('Add-in Express', Text) > 0 then
    CommitSmartTagProc(Pos('Add-in Express', Text), Length('Add-in Express'));
end;
```

TadxSmartTagModule

The main object of the smart tag library. Unit: adxSmartTag

```
TadxSmartTagModule = class(TDataModule)
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  property SmartTagDescs[LocaleID: Integer]: WideString;
  property SmartTagNames[LocaleID: Integer]: WideString;
published
  property NamespaceURI: string;
  property SmartTagDesc: WideString;
  property SmartTagName: WideString;
  property OnActionInitialize: TInitializeEvent;
  property OnActionFinalize: TNotifyEvent;
  property OnInitialize: TInitializeEvent;
  property OnFinalize: TNotifyEvent;
  property OnError: TOnErrorEvent;
end;
```

Implements common smart tag logic, centralizes all smart tag events. A container for other Add-in Express components related to smart tags. You can't use this class manually, all its instances are generated automatically.

Use the SmartTagName property to name your smart tag, and handle the smart tag events to perform custom actions.

NamespaceURI Property

The unique identifier of the smart tag library. Read-write.

```
published
  property NamespaceURI: string;
```

The value of NamespaceURI is generated automatically.

SmartTagDesc Property

A description of the smart tag. Read-write.

```
published
  property SmartTagDesc: WideString;
```

The description of the smart tag for any Locale that isn't specified in the SmartTagDescs property.

SmartTagDescs Property

Descriptions of the smart tag. Read-write.

```
public
  property SmartTagDescs[LocaleID: Integer]: WideString;
```

Depends on Locale.

SmartTagName Property

A name of the smart tag. Read-write.

```
published
  property SmartTagName: WideString;
```

The name of the smart tag for any Locale that isn't specified in the SmartTagNames property. The name appears in the Smart Tags tab of the AutoCorrect Options dialog box.

SmartTagNames Property

Names of the smart tag. Read-write.

```
public
  property SmartTagNames[LocaleID: Integer]: WideString;
```

Depends on Locale.

OnActionInitialize Event

Occurs when a smart tag action object is connected. Read-write.

```
published
```

```
    property OnActionInitialize: TInitializeEvent;

type
  TInitializeEvent = procedure (Sender: TObject;
    ApplicationName: WideString) of object;
```

ApplicationName - a string that represents the ProgID of the calling application. This is a way of getting the name of an application regardless of object model. Using the OnActionInitialize method to initialize smart tags gives you a chance, early in the process, to change the behavior of the smart tag based on host information.

Note. For Office XP the ApplicationName parameter returns an empty string.

OnActionFinalize Event

Occurs when a smart tag action is disconnected. Read-write.

```
  published
    property OnActionFinalize: TNotifyEvent;
```

Allows you to release the resources allocated in the OnActionInitialize event handler.

OnInitialize Event

Occurs when the smart tag library is connected. Read-write.

```
  published
    property OnInitialize: TInitializeEvent;

type
  TInitializeEvent = procedure (Sender: TObject;
    ApplicationName: WideString) of object;
```

ApplicationName - a string that represents the ProgID of the calling application. This is a way of getting the name of an application regardless of object model. Using the OnInitialize method to initialize smart tags gives you a chance, early in the process, to change the behavior of the smart tag based on host information.

Note. For Office XP the ApplicationName parameter returns an empty string.

OnFinalize Event

Occurs when the smart tag library is disconnected. Read-write.

```
  published
    property OnFinalize: TNotifyEvent;
```

Allows you to release the resources allocated in the OnInitialize event handler.

OnError Event

Occurs when an exception is raised by the smart tag library. Read-write.

```
  published
    property OnError: TOnErrorEvent;

type
  TOnErrorEvent = procedure (const E: SysUtils.Exception;
    var Handled: boolean) of object;
```

Allows handling an exception. By default, Add-in Express handles exceptions and shows an error message box. You can handle the exception yourself and avoid the error messages via Handled := True.