

**Integrate MS Office functionality into your app with point-and-click ease**

# OfficePartner<sup>TM</sup>

Flexible VCL component library handles complicated COM and automation techniques for you.

- *Read from and write to Word, Excel, and PowerPoint*
- *Output your program results to Word, Excel, and PowerPoint*
- *Send Outlook mail, manage task lists, access Outlook databases*
- *Add and manipulate Excel workbooks, worksheets, ranges, and charts*

# OfficePartner<sup>TM</sup>

TurboPower Software Company  
P.O. Box 49009  
Colorado Springs, CO 80949-9009

Order line (U.S. and Canada): 800/333.4160  
Elsewhere: 719/260.9136  
Fax: 719/260.7151

[www.turbopower.com](http://www.turbopower.com)

© 2000 TurboPower Software Company. All rights reserved.

## License Agreement

This software and its documentation are protected by United States copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted to the best of our ability.

© 2000 by TurboPower Software Company. All rights reserved.

TurboPower Software Company authorizes you to make archival copies of this software for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this software or documentation for the purposes of

distribution to others. Under no conditions may you remove the copyright notices made part of the software or documentation.

You may distribute, without run-time fees or further licenses, your own compiled programs based on any of the source code of OfficePartner. You may not distribute any of the OfficePartner source code, compiled units, or compiled example programs without written permission from TurboPower Software Company. You may not use OfficePartner to create components or controls to be used by other developers without written approval from TurboPower Software Company. OfficePartner is licensed for use solely on Microsoft Windows platforms.

Note that the previous restrictions do not prohibit you from distributing your own source code or units that depend upon OfficePartner. However, others who receive your source code or units need to purchase their own copies of OfficePartner in order to compile the source code or to write programs that use your units.

The supplied software may be used by one person on as many computer systems as that person uses. Group programming projects making use of this software must purchase a copy of the software and documentation for each member of the group. Contact TurboPower Software Company for volume discounts and site licensing agreements.

This software and accompanying documentation is deemed to be "commercial software" and "commercial computer software documentation," respectively, pursuant to DFAR Section 227.7202 and FAR 12.212, as applicable. Any use, modification, reproduction, release, performance, display or disclosure of the Software by the US Government or any of its agencies shall be governed solely by the terms of this agreement and shall be prohibited except to the extent expressly permitted by the terms of this agreement. TurboPower Software Company, PO Box 49009, Colorado Springs, CO 80949-9009.

With respect to the physical media and documentation provided with OfficePartner, TurboPower Software Company warrants the same to be free of defects in materials and workmanship for a period of 60 days from the date of receipt. If you notify us of such a defect within the warranty period, TurboPower Software Company will replace the defective media or documentation at no cost to you.

TurboPower Software Company warrants that the software will function as described in this documentation for a period of 60 days from receipt. If you encounter a bug or deficiency, we will require a problem report detailed enough to allow us to find and fix the problem. If you properly notify us of such a software problem within the warranty period, TurboPower Software Company will update the defective software at no cost to you.

TurboPower Software Company further warrants that the purchaser will remain fully satisfied with the product for a period of 60 days from receipt. If you are dissatisfied for any reason, and TurboPower Software Company cannot correct the problem, contact the party from whom the software was purchased for a return authorization. If you purchased the product directly from TurboPower Software Company, we will refund the full purchase price of the software (not including shipping costs) upon receipt of the original program media and documentation in undamaged condition. TurboPower Software Company honors returns from authorized dealers, but cannot offer refunds directly to anyone who did not purchase a product directly from us.

**TURBOPOWER SOFTWARE COMPANY DOES NOT ASSUME ANY LIABILITY FOR THE USE OF OFFICEPARTNER BEYOND THE ORIGINAL PURCHASE PRICE OF THE SOFTWARE. IN NO EVENT WILL TURBOPOWER SOFTWARE COMPANY BE LIABLE TO YOU FOR ADDITIONAL DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THESE PROGRAMS, EVEN IF TURBOPOWER SOFTWARE COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

By using this software, you agree to the terms of this section and to any additional licensing terms contained in the DEPLOY.HLP file. If you do not agree, you should immediately return the entire OfficePartner package for a refund.

All TurboPower product names are trademarks or registered trademarks of TurboPower Software Company. Other brand and product names are trademarks or registered trademarks of their respective holders.

---

# Table of Contents

<b>Chapter 1: Introduction .....</b>	<b>1</b>
System Requirements .....	5
Installation .....	5
The OfficePartner Trial-Run Edition .....	6
Files Supplied .....	7
Object Hierarchies .....	13
Organization of this Manual .....	17
On-Line Help .....	18
Technical Support .....	19
<b>Chapter 2 : The Basics of Automation .....</b>	<b>21</b>
OfficePartner and Office Automation .....	22
Glossary .....	27
Suggested Reading .....	31
<b>Chapter 3: Working with Excel .....</b>	<b>33</b>
Using the Application Object .....	36
Using Workbook Objects .....	36
Formatting Cell Contents .....	39
Microsoft Excel Demo .....	54
TOpExcelBase Class .....	60
TOpExcel Component .....	61
TOpExcelWorkbooks Class .....	78
TOpExcelWorkbook Class .....	80
TOpExcelWorksheets Class .....	85
TOpExcelWorksheet Class .....	87
TOpExcelRanges Class .....	91
TOpExcelRange Class .....	93
TOpExcelCharts Class .....	110
TOpExcelChart Class .....	112
TOpExcelHyperlinks Class .....	117
TOpExcelHyperlink Class .....	119
<b>Chapter 4: Working with Outlook .....</b>	<b>123</b>
Understanding the Application and MAPINamespace Objects .....	124
Outlook Examples .....	129
TOpOutlookBase Class .....	142
TOpOutlook Component .....	143

TOpRecipientList Class .....	159
TOpRecipient Class .....	163
TOpAttachmentList Class .....	170
TOpAttachment Class .....	174
TOpAppointmentItem Class .....	177
TOpContactItem Class .....	199
TOpJournalItem Class .....	231
TOpMailItem Class .....	244
TOpNoteItem Class .....	252
TOpPostItem Class .....	259
TOpTaskItem Class .....	271
TOpBaseData Class .....	292
TOpBusinessData Class .....	293
TOpDefaultData Class .....	298
TOpMiscData Class .....	301
TOpNetworkData Class .....	305
TOpPersonalData Class .....	310
<b>Chapter 5: Working with Word .....</b>	<b>313</b>
Microsoft Word Tutorial .....	315
TOpWord Component .....	320
TOpWordDocuments Class .....	334
TOpWordDocument Class .....	336
TOpDocumentTables Class .....	355
TOpDocumentTable Class .....	357
TOpDocumentBookmarks Class .....	360
TOpDocumentBookmark Class .....	362
TOpDocumentHyperLinks Class .....	364
TOpDocumentHyperLink Class .....	366
TOpDocumentShapes Class .....	368
TOpDocumentShape Class .....	370
TOpMailMerge Class .....	373
<b>Chapter 6: Working with PowerPoint .....</b>	<b>375</b>
Microsoft PowerPoint Tutorial .....	377
TOpPowerPoint Component .....	380
TOpPresentations Class .....	384
TOpPresentation Class .....	386
TOpSlides Class .....	390
TOpSlide class .....	392
TOpSlideTransition Class .....	396

<b>Chapter 7: Working with the Office Assistant .....</b>	<b>401</b>
TOpAssistant Class .....	403
TOpBalloon Class .....	412
<b>Chapter 8: The OfficePartner Event Model .....</b>	<b>415</b>
TOpEventModel Component .....	418
TOpDatasetModel Component .....	422
TOpContactsDataSet Component .....	426
<b>Chapter 9: Low Level Classes .....</b>	<b>433</b>
TOpNestedCollection Class .....	434
TOpNestedCollectionItem Class .....	438
TOpOfficeComponent Class .....	443
<b>Identifier Index .....</b>	<b>i</b>
<b>Subject Index .....</b>	<b>xix</b>



# Chapter 1: Introduction

1

OfficePartner is a unique and powerful product designed to bridge the gap between VCL development and the COM Automation Servers available in Microsoft Office 97 and 2000 applications. Microsoft Office, a suite of programs that includes Word, Excel, Outlook, and PowerPoint, is the most popular business productivity software for Windows in the world. Over 20 million licenses of Microsoft Office are in use at some of the world's largest corporations.

In addition to being powerful programs on their own, each application in the Office Suite can also be viewed as a collection of services, such as spell checking, chart generation, e-mail, contacts, and task management, that can be used by programs outside the Office Suite. OfficePartner makes it easier for Delphi and C++Builder programmers to "reach into Office" to use the services it provides.

Each of the Office applications (Excel, Word, Outlook, and PowerPoint) also expose their functionality as Component Object Model (COM) Automation Servers. If you follow some very specific rules, you can access and control these services yourself from your own applications. That's what COM Automation is all about.

OfficePartner shields developers from complicated COM and automation techniques. You can now use the functionality of Microsoft Office to solve your specific business needs in a familiar environment. The OfficePartner hierarchy isn't just a wrapper around calls to Office applications. Rather, it's a complete object-oriented framework that encapsulates the COM services so well that you don't really have to understand COM or Automation to use OfficePartner effectively. There are no Variants or difficult to understand syntax rules for using OfficePartner.

## Component property mechanism

The architecture behind OfficePartner components allows property settings to immediately be reflected in the Automation Server, or read from the Automation Server, if it is connected. Component properties can also be read or written if the Automation Server has not been connected (launched). This behavior allows developers the luxury of working free of the Office interface. It also allows design-time previewing of the exact affect that property settings will have on the Office Application. All property settings are saved in the component stream and restored when the component later launches the Automation Server.

OfficePartner's advanced architecture gives developers the ability to configure many settings at design time without have to write code. It also lets them switch between connected and disconnected mode, or run the application and activate the server at a later time, all without losing any property settings.

All OfficePartner components have a PropDirection property that allows further customization of this behavior. Setting PropDirection to pdFromServer allows properties to be read from the Office Application when it is initially launched, instead of pushing properties to the server from the component stream, as is the case when PropDirection is set to pdToServer.

It's this powerful property architecture that gives OfficePartner its rich design-time support. Properties that were simply wrappers to Automation Server properties or methods could only be used when the server was connected, could not persist values across different instances of the Automation Server, and would require mostly runtime code in order to control the Office Application.

## OfficePartner classes make Office automation easy

All the Microsoft Office Applications define an object hierarchy for the COM interfaces that represent the entities within the application. For example, Excel has Application, Workbook, Worksheet, Range, Chart and Hyperlink, etc. Word contains Application, Document, Bookmark, Table, Hyperlink, Shape, and Range. To represent these entities as separate components would leave the developer with the daunting task of defining and handling all of the complex relationships between these interfaces.

The top-level OfficePartner components all represent the main Application object (interface) within the Automation Server. The VCL TCollection and TCollectionItem classes offer one mechanism for holding the children (e.g. Workbooks, Documents, etc.) of the main Application interface. Collections would allow properties to be persisted and thus restored correctly, however this does not solve the problem of modeling how these different objects are related within the Office Automation Server. OfficePartner solves the problem by implementing specialized classes derived from VCL collections.

TOpNestedCollection and TOpNestedCollectionItem provide the basis that allows the OfficePartner components to correctly track and define the relationships between the interfaces of a specific Automation Server. TOpNestedCollectionItems can themselves contain multiple collections that in turn may contain other TOpNestedCollectionItems. This design provides a persistable, hierarchical representation of the objects contained in the Automation Server. “Nested Collection” classes provide simple navigation throughout the entire hierarchy. Nested Collections and their items are ultimately owned by a top-level OfficePartner Component, and delegate most COM interface creation and retrieval to the top-level component. This allows the top-level component to correctly track the state of the Automation Server. When combined with our property mechanism described earlier, Nested Collections allow OfficePartner to provide clean, consistent, and incredibly powerful design-time support for creating, maintaining and restoring the entities within an Office Automation Server.

## Before you begin using OfficePartner

In order to get the most out of the OfficePartner Component Suite, it is important to understand the boundaries between OfficePartner functionality and the automation capabilities supported by Microsoft Office. A clear grasp on this will help you to effectively use the OfficePartner documentation and other resources that may be required in order to implement your Office automation tasks.

OfficePartner provides the connective tissue necessary to simplify development with Office applications. OfficePartner automatically handles some of the more complicated automation tasks such as launching and shutting down servers, installing connection points for event hooks, navigating the Office object hierarchy, and gaining design time access to Office automation servers.

One common pitfall encountered by new OfficePartner users is to expect that the OfficePartner library and documentation will cover all aspects of Office development. While OfficePartner does provide a component-based framework that significantly lowers the cost of entry for Office automation, OfficePartner does not attempt to encapsulate (or document) every method or property that is available on the Office automation servers. OfficePartner has been designed to encapsulate access to the most commonly used Office automation tasks. Simple operations such as populating Word documents, sending Outlook mail, filling Excel worksheets, and displaying PowerPoint slide shows, among many others, can be accomplished using OfficePartner with no additional Office automation knowledge.

It is inevitable that more complicated solutions will eventually require automation functionality that is not currently encapsulated by OfficePartner. The type libraries for the Office automation objects contain literally thousands of methods and properties, only some of which are directly encapsulated by OfficePartner. When the time comes to use these additional methods, OfficePartner still offers tremendous help by allowing simple, intuitive navigation to the desired interface. Every OfficePartner class that represents an Office automation interface provides a simple property to get that interface. This architecture allows the developer to use OfficePartner for the basics and yet easily grab the raw automation interface when things get hectic.

For the base-level components, the `Server` property provides access to the main automation `CoClass` interface. By using this property, a developer has access to any methods or properties of a particular Office application's main interface (commonly called `Application` or `_Application`). A hierarchy of collections underneath the base-level components provides a VCL wrapper to the interfaces contained within this main interface. These collections can be easily navigated using array subscript notation. A developer can read the `Intf` property and at any time to get access to the raw automation interface. The `Intf` property must be typecast to the interface type that it represents. However, most OfficePartner collections also have a property named `AsXXX` that will automatically perform the typecast to the appropriate underlying automation interface. For example, the `Intf` property of

TOpExcelRange can be typecast to `_Range`, or you could simply access the `AsRange` property of `TOpExcelRange`.

This architecture allows developers the best of both worlds—the simplicity that `OfficePartner` provides and the ability to access the complete capabilities of Office automation. It is important to realize that once you begin to wade these waters on your own, you will need more in-depth knowledge of Office automation capabilities than `OfficePartner` documentation provides. Given the above information, the following is an efficient process when you want to reach deeper into Office automation capabilities:

Consult the `OfficePartner` documentation to see if the functionality you require is directly encapsulated by any of the `OfficePartner` components or classes.

If `OfficePartner` does not directly encapsulate the task(s) you wish to perform, consult the Office Development section of the Microsoft Developer Network (MSDN). This can be found online at [msdn.microsoft.com/library](http://msdn.microsoft.com/library) or on the MSDN library CD if you are a subscriber.

Once you have located the property or method you are looking for, and the interface that supports it, use the `Server`, `AsXXX`, or `Intf` properties to retrieve a reference to the correct interface from the `OfficePartner` component or class.

Since the `OfficePartner` library includes the type libraries for all the automation servers, you will get help from Delphi or C++ Builder's code completion. Most of the MSDN documentation for the Office objects contains Visual Basic syntax. With practice, you will become accomplished at translating type library data types into the correct Pascal or C++ syntax.

Many automation methods found in the type library contain optional parameters. These can be passed using the `EmptyParam` identifier. We have defined this in the `OpShared` unit, even for versions of the VCL that do not define this value.

Finally, you will find that many of the example programs, supplied in the `Demos` directory of the `OfficePartner` installation, show actual code for accessing Office automation functionality that is not directly encapsulated by the `OfficePartner` components.

---

## System Requirements

To use OfficePartner, your system must meet or exceed the following hardware and software requirements:

Microsoft Windows 95, 98, NT 4, or Windows 2000

Any 32-bit edition or version of Borland Delphi or C++Builder

A hard disk with at least 20MB of free space

Microsoft Office 97, 98, or 2000 or a subset thereof (Excel, Word, Power Point, or Outlook)

OfficePartner does not support 16-bit applications. It is a toolset for 32-bit applications only.

---

## Installation

OfficePartner can be installed directly from the TurboPower Product Suite CD-ROM. Simply insert the CD in your CD drive, select OfficePartner from the list of products on the screen, click on “Install Product” and follow the instructions. If the introductory splash screen does not appear automatically upon insertion of the CD, run X:\CDROM.EXE where X is the letter of the CD drive.

---

## The OfficePartner Trial-Run Edition

If you were using the Trial-Run Edition of OfficePartner and are about to install the full product, it's best that you remove all vestiges of the trial beforehand. This is to avoid having the Trial-Run Edition pre-compiled units being linked into your program, which would prevent the program from running unless the Delphi or C++Builder IDE is also running.

1. Remove all references to OfficePartner in Delphi or C++Builder. Remove the OfficePartner packages from the list of packages in the Install Packages dialog. You should also remove the empty palette items from the component palette via the Environment Options | Palette menu.
2. Delete the directory where the OfficePartner trial-run was installed. Use Windows Explorer to make this operation easier.
3. Check the Windows\System (WINNT\System32 in NT) directory for OfficePartner packages. They can be identified by Pnnn\_Txx.BPL, where nnn is the OfficePartner version number, e.g., 300, and xx is the VCL version, e.g., 40 for Delphi 4, 41 for C++Builder 4, etc. If any OfficePartner packages are found, delete them.
4. If using Delphi 5 and later or C++Builder 4 and later, check the Projects\BPL directory under the directory where the compiler is installed. Look for the packages as described in the previous step and, if present, delete those packages. Do NOT delete any other packages.
5. Using REGEDIT, the Windows Registry editor, check the following key:

```
HKEY_CURRENT_USER\Borland\  
<Compiler>\<version\Disabled Packages
```

where <Compiler> is the particular compiler, e.g., Delphi, and <version> is the general version of the compiler, e.g., 4.0. If there are any OfficePartner packages listed in this key, delete those references.

You are now ready to install the full version of OfficePartner. Future releases of the Trial-Run Edition may include an uninstall program. If you installed from a version that did include an uninstall, the above steps should not be necessary. Should you encounter problems such as an application refusing to run because it contains code from the Trial-Run Edition, these steps will help you eliminate the problem.

---

# Files Supplied

Following is a list of files supplied with OfficePartner and a brief description of their contents.

## Source files

### **OpAbout.pas**

Contains the “About” screen that users will see when they right-click an Office component and select Application Info from the context menu.

### **OpDb0fc.pas**

Contains the declaration and implementation of TOpDataSetModel and the declaration of EmodelException.

### **OpDb0lk.pas**

Declares and implements several of the Outlook data related classes.

### **OpXL2K.pas**

Contains the type library interface for the Microsoft Excel 2000 object model (Excel9.olb). Additional reference for this object model is available in VBAXL9.CHM (included with MSDN).

### **OpXL97.pas**

Contains the type library interface for the Microsoft Excel 97 object model (Excel8.olb). Additional reference for this object model is available in VBAXL8.CHM (included with MSDN).

### **OpExcel.pas**

Declares and implements the Excel classes.

### **OpFrms97.pas**

Contains the type library interface for the Microsoft Forms 2000 object model (FM20.DLL). Additional reference for this object model is available in FM20.hlp (included with MSDN).

### **OpMS0.pas**

Declares and implements the Microsoft Assistant and Balloon classes.

### **OpColEd.pas**

Declares and implements the TOpNestedCollection class. This class defines the design-time nested collection editor you'll work with in Delphi's design mode.

**OpOfc2k.pas**

Contains the type library interface for the Microsoft Office 2000 object model (MSO9.DLL). Additional reference for this object model is available in vbaOff9.chm (included with MSDN).

**OpOfc97.pas**

Contains the type library interface for the Microsoft Office 97 object model (MSO97.DLL). Additional reference for this object model is available in vbaOff8.chm (included with MSDN).

**OpConst.pas**

Defines miscellaneous string constants.

**OpDesign.pas**

Defines property editors for the OfficePartner components.

**OpEvents.pas**

Declares and implements the event classes.

**OpModels.pas**

Declares and implements the TOpUnknownComponent and the IOpModel interface which is used to connect to a TDataSet descendant.

**OpShared.pas**

Declares and implements the foundation classes common to all Office versions.

**OpOlk2k.pas**

Contains the type library interface for the Microsoft Outlook 2000 object model (MSOutl.OLB). Additional reference for this object model is available in vbaOutl9.chm (included with MSDN).

**OpOlk97.pas**

Contains the type library interface for the Microsoft Outlook 97 object model (MSOutl8.OLB). Additional reference for this object model is available in vbaOutl.hlp (included with MSDN).

**OpOlk98.pas**

Contains the type library interface for the Microsoft Outlook 98 object model (MSOutl85.OLB). Additional reference for this object model is available in vbaOutl.hlp (included with MSDN).

**OpOutlk.pas**

Declares and implements the various Outlook Item classes.

**OpPpt2k.pas**

Contains the type library interface for the Microsoft PowerPoint 2000 object model (MSppt9.OLB). Additional reference for this object model is available in VBAPPT9.CHM (included with MSDN).

**OpPpt97.pas**

Contains the type library interface for the Microsoft PowerPoint 97 object model (Msppt8.olb). Additional reference for this object model is available in VBAPPT8.HLP (included with MSDN).

**OpPower.pas**

Declares and implements the PowerPoint Classes.

**OpVbld2k.pas**

Contains the type library interface for the Microsoft Visual Basic Editor 97 object model (VBEEXT1.OLB). Additional reference for this object model is available in VBOB6.chm (included with MSDN).

**OpVblb97.pas**

Contains the type library interface for the Microsoft Visual Basic Editor 2000 object model (VBE6EXT.OLB). Additional reference for this object model is available in VEENOB3.HLP (included with MSDN).

**OpWrd2k.pas**

Contains the type library interface for the Microsoft Word 2000 object model (msword9.olb). Additional reference for this object model is available in VBAWRD9.CHM (included with MSDN).

**OpWrd97.pas**

Contains the type library interface for the Microsoft Word 97 object model (Msword8.olb). Additional reference for this object model is available in VBAWRD8.HLP (included with MSDN).

**OpWord.pas**

Defines and implements the Word classes.

## Form files

### OpAbout.dfm

Contains the “About” form that users will see when they right-click an Office component and select Application Info from the context menu.

### OpColEd.dfm

Contains the form used by the design-time nested collection editor.

## Package files

To avoid version conflicts with applications using different versions of the OfficePartner packages, each new version of OfficePartner comes with packages using slightly different names. TurboPower's current package naming convention is as follows:

- The first letter represents one of the TurboPower products (P for OfficePartner).
- The next three digits are the product version number (160 for 1.60).
- Two product-specific digits (\_R for run-time, or \_D for design).
- The last two digits represent the VCL version (30, 35, 40, 41, 50, 51).

The OfficePartner run-time packages and design packages for Delphi 3/4/5 and C++Builder 3/4/5 are:

P160_R30.DPL, P160_D30.DPL	Delphi 3
P160_R40.BPL, P160_D40.BPL	Delphi 4
P160_R50.BPL, P160_D80.BPL	Delphi 5
P160_R35.BPL, P160_D35.BPL	C++Builder 3
P160_R41.BPL, P160_D41.BPL	C++Builder 4
P160_R51.BPL, P160_D51.BPL	C++Builder 5

## Example programs

The following examples are included with OfficePartner to demonstrate useful features of the product. They can be found in the Examples folder.

### ExAppend

ExAppend demonstrates how to create a new Word document and then append the contents of a preexisting word document to the new document.

## ExAppts

This example demonstrates how to access appointments from the Outlook calendar. Once the appointments are received, this example shows you how to iterate the list and access individual fields of the appointment items. (Not available in C++Builder)

## ExConn

This example demonstrates how to associate and connect an open Word document to the TOpWord component. This example requires a Word document to already be open.

## ExEvent

ExEvent demonstrates how to send an email using outlook. It also demonstrates the use of the TOpOutlook.OnItemSend event. Parameters passed to this event are casted to demonstrate how to extract particular information. (Not available in C++Builder)

## ExFields

This example demonstrates how to open a new Word document and add a hyperlink. This example also allows you to see the underling code body of the hyperlink.

## ExFind

This example allows you to open an existing Word document and then perform a user-defined search on this document.

## ExMerge

ExMerge uses data from the Animals.dbf sample table to populate a Word table via the TOpDatasetModel. You can then populate the fields of a mail merge for this example by providing a database alias and a query against the database. We used the following parameters when prompted:

Alias : "DBDEMONS"

SQL : "SELECT Name, Continent FROM Country.db"

Then we can add fields to the document in preparation for a mail merge. Finally, the merge is executed and the new merge result document is displayed.

## ExO2XI

This example demonstrates the use of the TOpDataSetModel to transfer contact information from Outlook to an Excel worksheet.

## ExPpt

This example demonstrates how to automate a PowerPoint presentation. The layout, transition effect, and transition speed can be customized. You can also set up the presentation to advance on user input (click) or on a user-definable time delay.

---

## XlRange

XlRange demonstrates the functionality of an Excel range. Color, pattern, font characteristics and effects, border appearance, column width, column height, text alignment and orientation are all customizable within this example program.

## ExSMail

This example demonstrates how to address and send an email. Rather than forcing you to address the mail manually, this example accesses Outlook's global address lists and allows you to select recipients.

## ExTbl1

ExTbl1 demonstrates how to use the TOpEventModel to populate a table within a Word document with data from a string grid.

## ExTbl2

This example is very similar to ExTbl1, but demonstrates how to achieve the same results without the use of the TOpEventModel component.

## XLDemo

This example demonstrates how to transfer data from a TDataset descendant to an Excel worksheet using the Populate method of the TOpDatasetModel component. It also demonstrates how to provide Excel with master-detail information via the PopulateCurrent method of the TOpDataset model. Finally, an Excel chart is created and associated with data from the application.

## Object Hierarchies

All OfficePartner components descend from TComponent through TOpBaseComponent which implements the Version property that follows:

Version	property
---------	----------

property Version : string

↳ Shows the current version of OfficePartner

Version is provided so you can identify your OfficePartner version if you need technical support.

Following are diagrams showing the OfficePartner object hierarchies. All of the diagrams show the component or class name and the unit name where the control is implemented.

# The OfficePartner class hierarchy

TObject

TPersistent

TOpBalloon

TOpSlideTransition

TOpMailMerge

TCollection

TOpNestedCollection

TOpExcelCharts

TOpExcelHyperlinks

TOpExcelRanges

TOpExcelWorkbooks

TOpExcelWorksheets

TOpPresentations

TOpSlides

TOpDocumentBookmarks

TOpDocumentHyperlinks

TObject

TPersistent

TCollection

TOpNestedCollection

TOpDocumentRanges

TOpDocumentShapes

TOpDocumentTables

TOpWordDocuments

TCollectionItem

```
TOpNestedCollectionItem
    TOpExcelChart
    TOpExcelHyperlink
    TOpExcelRange
    TOpExcelWorkbook
    TOpExcelWorksheet
    TOpPresentation
    TOpSlide
    TOpDocumentBookmark
    TOpDocumentHyperlink
    TOpDocumentRange
    TOpDocumentShape
    TOpDocumentTable
    TOpWordDocument

TObject
TPersistent
TComponent
    TOpAssistant
    TOpOfficeComponent
        TOpOutlookBase
            TOpOutlook
        TOpExcelBase
            TOpExcel
        TOpPowerPointBase
            TOpPowerPoint
        TOpWordBase
            TOpWord
```

---

TDataSet  
TOpContactsDataSet  
TOpUnknownComponent  
TOpDataSetModel  
TOpEventModel  
TObject  
TOpEventAdapter  
TOpEventSink  
TList  
TOpFreeList  
TOpBaseData  
TOpBusinessData  
TOpDefaultData  
TOpMiscData  
TOpNetworkData  
TOpPersonalData  
TOpBaseDataClass  
TOpMailListItem  
TObject  
TPersistent  
TOpAttachment  
TOpRecipient  
TOpMailListProp  
TOpAttachmentList  
TOpRecipientList  
TOpOutlookItem  
TOpAppointmentItem

TOpContactItem

TOpJournalItem

TOpMailItem

TOpNoteItem

TOpPostItem

TOpTaskItem

#### Exception

EOpEventSinkException

EOpOfficeError

EOpModelException

EOpRangeException

---

# Organization of this Manual

This manual is organized as follows:

- Chapter 2 explains various mechanisms for accessing automation servers, namely through variants, interfaces, and dispinterfaces. It also has a glossary of automation/COM terms and a listing of additional references.
- Chapters 3 through 7 describe how OfficePartner interacts with Microsoft Office tools.
- Chapter 8 explains the process of transferring data between Office applications and data sources.
- Chapter 9 describes the base classes that provide the common core functionality of the OfficePartner components.
- An identifier index and a conventional subject index are provided.

Each chapter starts with an overview of the automation task supported by the classes and components discussed in that chapter. Each class and component is then documented individually, in the following format:

## Hierarchy

This list of the ancestors of the class being described generally stops at a VCL class. It lists the unit in which each class is declared and the number of the first page of the documentation of each ancestor. Some classes in the hierarchy are identified with a number in a bullet: ❶. This indicates that some of the properties, methods or events listed for the class being described are inherited from this ancestor and documented in the ancestor class.

## Properties

This list of the properties in the class includes those inherited from ancestor classes. The properties identified with a number in a bullet, ❶, are documented in the ancestor class from which they are inherited.

## Methods

This list of the methods in the class includes those inherited from ancestor classes. The methods identified with a number in a bullet, ❶, are documented in the ancestor class from which they are inherited.

## Events

This list of the events in the class includes those inherited from ancestor classes. The events identified with a number in a bullet, ①, are documented in the ancestor class from which they are inherited.

### Reference section

This section includes detailed documentation for the properties, methods, and events. These descriptions are in alphabetical order. They have the following format:

Declaration of the property, method, or event.

Default value for properties, if appropriate.

A short, one-sentence purpose. A ↗ symbol marks each purpose to make skimming through these descriptions easier.

Description of the property, method, or event. Parameters are also described here.

Examples are provided in many cases.

The “See also” section lists other properties, methods, or events that are pertinent to this item.

### Naming conventions

To avoid class name conflicts with components and classes included with the VCL or from other third party suppliers, all OfficePartner class names begin with “Op”.

---

## On-Line Help

Although this manual provides a complete discussion of each component, keep in mind that there is an alternative source of information available. Once the help file is properly installed, you can access help from within the IDE by pressing <F1> when:

- The editing caret is on a property, method, or event in the Code Editor
- A component is selected in the Form Designer
- A property or event is selected in the Object Inspector

## Technical Support

The best way to get an answer to your technical support questions is to post them in the OfficePartner newsgroup on our news server ([news.turbopower.com](http://news.turbopower.com)). Many of our customers find the newsgroups a valuable resource where they can learn from other's experiences and share ideas in addition to getting answers to questions.

To get the most from the newsgroups, we recommend that you use dedicated newsreader software, such as Forté, Free Agent, or Microsoft Outlook Express.

Newsgroups are public, so please do not post your product serial number, unlocking code, or any other private information (such as credit card numbers) in your messages.

The TurboPower Knowledgebase is another excellent support option. It has hundreds of articles about TurboPower products accessible through an easy to use search engine ([www.turbopower.com/search](http://www.turbopower.com/search)). The Knowledgebase is open 24 hours a day, 7 days a week so it provides an alternative resource for support that is constantly available.

You can also read about further support options at [www.turbopower.com/support](http://www.turbopower.com/support).

# Chapter 2 : The Basics of Automation

The purpose of this chapter is to expose you to concepts, terms, and approaches that you are likely to encounter in your automation endeavors. This chapter is by no means intended to be all encompassing. Rather, it should be used as a springboard to inspire additional reading. The first part of this chapter discusses various mechanisms for accessing automation servers, namely through variants, interfaces, and dispinterfaces. The advantages and disadvantages of each will be addressed.

The remainder of this chapter is devoted to a glossary of common automation/COM terms, followed by a listing of additional references that you may find helpful.

In simplest terms, automation is the scheme that allows one application to be controlled (automated) by another application. The application that is being controlled is referred to as the automation server and the application that is controlling the automation server is the client. In order for this scheme to work properly, two crucial issues must be resolved:

- The automation server must expose its functionality to the outside world.
- The client must be made aware of this functionality.

Automation servers come in two flavors:

- In-Process—The server is implemented inside of a library (a DLL) that runs in the same process space as the client.
- Out-Of-Process—The server is implemented in an executable file (an EXE) that runs in its own process space, separate from the client process space. Out-of-process servers are slightly more complex than in-process servers because function parameters and return values must be passed between the two different process spaces. This process is known as marshalling. Out-of-process servers are also slightly more restrictive than in-process servers because only certain data types can be marshaled automatically across process boundaries. Specifically, only SmallInt, Integer, Single, Double, Currency, TDateTime, WideString, IDispatch, SCODE, WordBool, OleVariant, IUnknown, ShortInt and Byte currently enjoy automatic marshalling. It should be noted that Delphi does provide three additional interfaces (IPicture, IStrings, and IFONTS) for passing pictures, string lists, and fonts across process boundaries. You can also provide marshalling support for additional data types, but that is beyond the scope of this manual.

Microsoft Excel, Word, PowerPoint, and Outlook are all stand-alone executables and are therefore accessed as out-of-process servers. Fortunately, OfficePartner encapsulates the complexity of controlling these applications.

# OfficePartner and Office Automation

This section addresses both the server implementation and the client access methods. In the context of OfficePartner, we are concerned nearly exclusively with playing the client role. Consequently, this section is heavily weighted on accessing and controlling automation servers rather than creating them.

## Exposing the functionality (the server's role)

Automation is not simply a by-product of any and all applications written. Rather, automation servers must be designed with automation in mind. The developers of an application must decide that their application should be able to be controlled by another application. With that in mind, the developers agree on an interface to their application. The interface defines exactly what functionality is available to external applications. This functionality is generally exposed in the form of one or more interfaces and/or one or more dispinterfaces. In fact, the mere existence of an IDispatch interface technically qualifies the server for automation. How then do we access these interfaces?

## Tapping into the exposed functionality (the client's role)

As a user of the OfficePartner library, you are primarily concerned with controlling the Office applications from within your application. Once again, the OfficePartner class hierarchy has shielded you from this complexity. However, OfficePartners exposes several interface properties that are at your disposal.

The client gains access to the automation server using one of three mechanisms:

- Variants
- Dispinterfaces
- Interfaces

### Variants

Accessing automation servers through variants is by far the most inefficient approach. A vast amount of behind the scenes work must be done to accomplish a given task.

Consider the following code:

```

var
  v : Variant;
begin
  {Create an instance of the server and
  assign it to a variable of type Variant}
  v := CreateOleObject('Word.Basic');
  {manipulate the server via the Variant}
  v.AppShow;
  v.FileNew;
  v.Insert('This is very inefficient');
end;

```

This code starts Microsoft Word, creates a new document, and inserts the string “This is very inefficient” into the document. This is powerful code, but it raises a few questions.

For example, how did we know to pass the string Word.Basic to the CreateOleObject function? In Delphi’s documentation, the string parameter passed to the CreateOleObject function is described as “the string representation of the Class ID.” The string representation of the ClassID is simply the name of the automation server followed by a period that is followed by the name of the CoClass. If you know the GUID of the CoClass, you can search the registry for the GUID under the HKEY\_CLASSES\_ROOT\CLSID key. There you will find a sub-key called ProgID. The value of this sub-key defines the string you should pass to the CreateOleObject function.

The result of CreateOleObject is an IDispatch interface to the newly created object. Given that, how can you be sure that AppShow, FileNew, and Insert are valid methods of this interface? The truth is that you can’t be sure (unless you wrote the server class yourself). In fact, since you’re using late binding, your application will compile just fine regardless of the method calls used. Only at run time will you discover if you’ve called a non-existent method. The fact that you’re using late binding should lead to yet another question. Namely, how is the binding resolved at run time? The answer can be found within the methods of the IDispatch interface. The following sequence of events occurs for each and every method call on an IDispatch interface:

1. . The name is passed to IDispatch.GetIDsOfNames. This function returns an index (an integer) representing the method’s position in the virtual method table.
2. IDispatch’s Invoke method is called passing the index returned in step 1.

In summary, accessing servers through variants requires more direct knowledge of the server’s interface (which is often not available) and the additional level of indirection associated with variants is much less efficient than both interfaces and dispinterfaces (discussed next).

## Interfaces

Type libraries are included with all automation servers. Whether they are stand-alone type library files or embedded within a DLL or executable file is insignificant. The crucial point is that you must know the location of the type library so that you can import it into a Pascal version of Interface Declaration Language (IDL). Once a type library is imported, you can access its properties and methods just as you access properties and methods of any VCL class. To illustrate this point, let's step through an example:

The English type library for Microsoft Excel is embedded in a file named XL5EN32.OLB. You can easily import this type library into Delphi or C++Builder using the following steps:

3. . Select Project | Import Type Library from the IDE's main menu.
4. Click on the Add button, browse to the file XL5EN32.OLB, and click OK. The file is normally in the Microsoft Office\Office folder.
5. Click Create Unit in the Import Type Library dialog.
6. The declaration of Excel's interface will be extracted and saved to the compiler's Imports folder.

You can now invoke any method of Excel's automation interface. Note that since early binding is in effect, the IDE's code completion expert is able to extract the properties and methods as you edit your source code.

Due to the fact that early binding occurs with interfaces, there is no need to translate method names and parameters at run-time. This fact alone makes interfaces the most efficient means of controlling automation servers. As an added benefit of using interfaces, source code errors will be recognized at compile-time.

To control an automation server using interfaces, code similar to the following is used:

```
var
  i : ISomeServer;
begin
  {Create the server and get the interface}
  i := coSomeServer.Create; {coSomeServer = the coClass}
  {manipulate the server using methods of the interface}
  i.DoSomething;
end;
```

## Dispinterfaces

Given what you now know about interfaces and variants, you may wonder why you would consider using anything but interfaces for your future automation clients. The fact is, unless you're programming in Visual Basic, you probably wouldn't. Unfortunately, Visual Basic is incapable of accessing automation servers via interfaces. This is where dispinterfaces come in. Dispinterface declarations are identical to interface declarations with two exceptions:

- Dispinterfaces are declared using the keyword dispinterface instead of interface.
- Properties and methods of a dispinterface declaration are followed by the keyword “dispid” and an integer constant.

Below is an example of a fictitious interface declaration and its corresponding dispinterface declaration:

```
IFictitiousServer = interface(IDispatch)
  ['{8707504E-C5CA-49B4-992B-3F491CB59B68}']
  function DoSomething(Param1 : Integer) : Double ; safecall;
end;

IFictitiousServer = dispIinterface
  ['{8707504E-C5CA-49B4-992B-3F491CB59B68}']
  function DoSomething(Param1 : Integer) : Double ; dispID 1;
end;
```

Note that even the GUIDs are identical.

Dispinterfaces are very similar to variants but are slightly more efficient because the integer constant equates to the integer returned from the IDispatch.GetIDsOfNames function. Dispinterfaces do, in fact, utilize the same IDispatch interface mechanism as variants, but the number of steps required to resolve function calls and parameters are essentially cut in half.

To control an automation server using dispinterfaces, code similar to the following is used:

```
var
  di : ISomeServerDisp;           {server dispinterface}
begin
  {Create an instance of the server & get
  the dispinterface to it}
  di := coSomeServer.Create as ISomeServerDisp;
  {invoke the server's methods}
  di.do something;
end;
```

## Dual interfaces

A dual interface is an interface that supports both compile-time binding and run-time binding through automation. Dual interfaces must descend from IDispatch. All methods of a dual interface (except from those inherited from IUnknown and IDispatch) must use the safecall calling convention, and all method parameter and result types must be automatable. (The automatable types are Byte, Currency, Real, Double, Real48, Integer, Single, Smallint, AnsiString, ShortString, TDateTime, Variant, OleVariant, and WordBool.)

## How do Office applications expose their functionality?

Microsoft Office applications (specifically Excel, Word, Power Point, and Outlook) expose their functionality to automation clients via dispinterfaces. This is not too surprising considering the large number of Visual Basic programmers that use Office automation. If you take a moment (or several hours) to peruse one of the Office application's imported type library files, you'll find a wealth of exposed functionality. (Excel's imported type library file contains nearly 130 defined dispinterfaces, and each dispinterface defines anywhere from 5 or 6 methods to as many as 110.)

### The OfficePartner solution

OfficePartner's intuitive class hierarchy implements the most frequently used features of Excel, Word, Outlook, and Power Point. Navigating the class hierarchies of OfficePartner is much less daunting and more intuitive than sifting through the volumes of properties, methods, events, and constants that these Office applications have exposed in the form of dispinterfaces. Each of the following chapters contain tutorials that describe how to use OfficePartner to control Excel, Word, Outlook, and PowerPoint.

For further explanation of COM related terminology, refer to the glossary section.

---

# Glossary

The following definitions constitute a small percentage of the terms that you may encounter working with COM. However, they are representative of the group of terms you will encounter most frequently. For further explanation, refer to the list of suggested readings following this section.

## ActiveX control

This is simply a new name given to what was formerly known as OLE controls or OCX controls. ActiveX controls are components (objects) designed for reusability that can be inserted into an application or Web page.

## automation client

An application that makes requests of and controls an automation server. Applications that you develop using OfficePartner will behave as automation clients in the fact that they will be controlling and making requests of Microsoft Office applications.

See also: automation server.

## automation server

An application that has deliberately exposed functionality to external processes. The exposed functionality allows another program or process (automation client) to make requests and externally control the server. Microsoft Office applications play the role of automation server when using OfficePartner.

See also: automation client

## binding, early

The method by which compilers resolve function or procedure calls when enough information is available at design time to resolve the reference. The compiler will embed the function or procedure code directly into the executable file. Early binding equates to more efficient executables; however, there are times when flexibility is a priority over efficiency. For these cases, virtual functions or procedures can be defined and the compiler will be forced to use late binding to resolve the reference at run-time. OfficePartner exposes interfaces that allow early binding, thus increasing efficiency.

See also: binding, late

## binding, late

The method by which compilers resolve function or procedure calls when insufficient information is available at design time to resolve the reference. Late binding is a cornerstone of object-oriented programming. Late binding is what allows you to pass an ancestor object type to a function or procedure and allow the object to behave in its own way.

See also: binding, early

## class factory

A COM object whose sole purpose is to construct and return references to other COM objects. A class factory can create one or more different classes of objects.

See also: COM

## CLSID (class ID)

A GUID that is assigned to a class. Clients pass the CLSID to the COM library that, in turn, creates an instance of the object. The CLSID serves as the key to the appropriate class factory. Objects that are not created by the COM library do not require a CLSID.

See also: COM, class factory, GUID

## COM

Component Object Model. This is a Microsoft standard that allows objects to be instantiated and controlled by external processes. There are volumes related to this subject. See the recommended readings section for further details.

## connection point

An “outgoing” interface supported by the automation server. This allows the server to act as a pseudo-client to the actual client making requests. Automation servers use connection points to implement events and pass notification of their occurrence back to the client.

See also: automation server, automation client.

## DCOM

Distributed COM. DCOM is an extension of COM that allows processes (clients and servers) to communicate across machine boundaries.

See also: COM

## **DISPID**

A constant integer value assigned to a single function, procedure, or property of a dispinterface.

See also: dispinterface

## **dispinterface**

An interface that associates a DISPID with its functions, procedures, and properties. Dispinterfaces are generally implemented to support Visual Basic clients since Visual Basic is incapable of manipulating interfaces.

See also: DISPID

## **GUID**

Globally Unique Identifier. This is a 16-byte value (created programmatically) that is statistically proven unique in space and time. GUIDs are used extensively in COM as keys to classes, class factories, and interfaces.

See also: CLSID

## **IDispatch**

A commonly implemented interface that provides a mechanism for late binding. Refer to the VCL documentation for further treatment of this subject.

See also: binding, late

## **IDL**

Interface Definition Language. IDL is a standard language of COM. This language is often used to define COM interfaces.

See also: COM, interface

## **in-process/out-of-process**

These terms are used to describe where the automation server resides relative to the automation client. In-process servers execute in the same process space as the client and are typically implemented within DLLs. Out-of-process servers describe an executable server that is running within its own, external process space.

See also: automation server, automation client, marshalling

## interface

A COM “contract” that stipulates precisely the functions, procedures, properties, and data types that a COM object supports. Once an interface has been designed, implemented, and shipped, it can no longer be changed in any degree. Instead, new interfaces must be defined to support expanded functionality.

See also: COM, dispinterface

## IUnknown

The most basic and fundamental interface that all COM objects are required to implement. The IUnknown interface defines reference-counting methods to ensure that COM objects are destroyed when their service is no longer required. IUnknown also implements the QueryInterface method that provides the mechanism that allows clients to ask the server if a particular interface is implemented by the server.

See also: COM

## marshalling

The mechanism by which data is passed across process boundaries when an out-of-process server is being accessed. The most common data types are marshaled automatically. Some data-types and all user-defined data-types will require that you implement the marshalling yourself. Custom marshalling is however, beyond the scope of this manual.

See also: in-process/out-of-process

## type library

A standardized description of an automation server’s interfaces. The developer of an automation server must make this library available to all clients who intend to automate the server. This file can be distributed to the client in various forms, but is generally embedded in a DLL.

See also: automation server, automation client, interface

## variant

A data type used to hold values whose type cannot be determined at design time. Refer to the VCL documentation for further treatment of the Variant type.

---

## Suggested Reading

*COM/DCOM Unleashed* by Randy Abernathy (Sams Publishing)

*Delphi COM Programming* by Eric Harmon (Macmillan Technical Publishing)

*Effective COM* by Booch, Jacobsen and Rumbaugh (Addison Wesley Publishing)

*Essential COM* by Don Box (Addison Wesley Publishing)

*Inside COM* by Dale Rogerson (Microsoft Press)

*Inside Distributed COM* by Guy Eddon and Henry Eddon (Microsoft Press)

*Understanding ActiveX and OLE* by David Chappell (Microsoft Press)



# Chapter 3: Working with Excel

This chapter provides guidance on controlling Microsoft Excel using OfficePartner. The fundamental component representing the Excel server object is encapsulated by the TOpExcel component. The TOpExcel component maintains various collections representing workbooks, worksheets, ranges, charts, and hyperlinks.

Workbooks are analogous to the workbooks in Excel. A collection of workbooks is encapsulated by the TOpExcelWorkbooks class, which is itself a collection of individual workbook classes (TOpExcelWorkbook).

The TOpExcelWorkbook class maintains a collection of worksheets just as in Excel. These worksheets are a collection of individual worksheet objects. Each worksheet is encapsulated by the TOpExcel worksheet class.

The Worksheet class maintains three distinct collections. These collections are TOpExcelRanges, TOpExcelCharts, and TOpExcelHyperlinks. Each of these collections is a parent collection to singular items of the same name. For example, TOpExcelRanges is the parent collection of individual TOpExcelRange objects.

The TOpExcelRange class is analogous to cells within a spreadsheet or a group of cells within a worksheet. This is likely to be the class you interact with most frequently as it provides the link to transferring data to and from Excel.

Each Worksheet can also contain one or more charts and/or hyperlinks. When using OfficePartner to control Excel, all charts must be embedded charts. OfficePartner does not support chart sheets (\*.xlc files). Hyperlinks can also be embedded within a worksheet. Hyperlinks can point to addresses on the web, or they may simply reference another cell within the parent workbook.

## The Application object

The Excel Application object is the top-level object in Excel's object model. You use the Application object to determine or specify application-level properties or execute application-level methods. The Application object is also the entry point into the rest of the Excel object model.

Like other Office application object models, the Excel Application object exposes several properties that work with a currently active Excel object. The Application object exposes Excel workbooks, worksheets, ranges, charts, and hyperlinks, each of which exposes their own functionality. For example, the Workbooks property returns the Workbooks collection that contains all the currently open Workbook objects. The Worksheets property returns the Worksheets collection associated with the currently active workbook.

## The Workbook object

In the Excel object model, the Workbook object appears just below the Application object. The Workbook object represents an Excel workbook file (\*.xls or \*.xla). You use the Workbook object to work with a single Excel workbook. You use the Workbooks collection to work with all currently open Workbook objects.

The Workbooks collection's Count property determines how many visible and hidden workbooks are open. By default, Excel typically has one hidden workbook named Personal.xls. The Personal.xls workbook is created by Excel as a place to store macros. If the hidden Personal.xls workbook is the only open workbook, the ActiveWorkbook property returns nil, but the Workbooks collection's Count property returns 1. The Workbooks collection's Count property returns 0 only when there are no open workbooks, hidden or visible.

## The Worksheet object

Most of the work you do in Excel is within the context of a worksheet. A worksheet contains a grid of cells to work with data and hundreds of properties, methods, and events to work with the data in a worksheet.

To work with the data contained in a worksheet, a cell, or within a range of cells, you use a Range object. The Worksheet and Range objects are the two most basic and most important components of any custom solution you create within Excel.

Each Excel workbook contains one or more Worksheet objects and can contain one or more chart sheets as well. Charts in Excel are either embedded in a worksheet or contained on a chart sheet. OfficePartner provides support only for embedded charts. You can have multiple charts on a worksheet. Each embedded chart on a worksheet is a member of the Worksheet object's Charts collection. Worksheet objects are contained in the Worksheets collection, which you can access by using the Workbook object's Worksheets property.

Because a Worksheet object exists as a member of a Worksheets collection, you refer to a worksheet by its name or its index value.

## The Range object

In Excel, the Range object is the most powerful, dynamic, and often-used object. Once you develop a full understanding of the Range object and how to use it effectively within the OfficePartner framework, you will be well on your way to harnessing the power of Excel.

The Excel Range object is somewhat unique in terms of objects. In most cases, an object has a clearly identifiable corollary in the Excel user interface. For example, a Workbook object is recognizable as an .xls file. In a workbook, the collection of Worksheet objects is represented in the user interface by separate worksheets. However, the Range object is unique. A range

varies depending on the circumstances. A Range object can be a single cell or a collection of cells. It can be a single object or a collection of objects. It can be a row or column, and it can represent a three-dimensional collection of cells that span multiple worksheets. It is probably easiest to think of the Range object as the handle to the particular worksheet element with which you want to work.

Because the Range object is such a fundamental entity within Excel, many properties and methods return a Range object. Most of the useful properties of the range have been encapsulated by OfficePartner, but you can expand on this functionality by using the AsRange property to return an interface to the Range object. You can then use the properties and methods of that Range to work with the data in a cell or group of cells.

You can also use the AsRange property to return interfaces to Range objects as arguments to other methods in the Excel object model. When you use the AsRange property in this way, make sure you fully qualify the Worksheet object to which the Range interface property applies. Failing to use fully qualified references to the Range interface property in arguments for Excel methods is one of the most common sources of error in range-related code.

## The Chart object

Each Excel worksheet can contain a collection of embedded charts. OfficePartner does not support chart sheets (\*.xlc) at this time. Individual charts can be accessed by specifying an index into the collection. Chart objects expose their functionality just as workbooks and worksheets do, and therefore can be manipulated individually.

## The Hyperlink object

Each Excel worksheet can contain a collection of hyperlinks. Individual hyperlinks can be accessed by specifying an index into the collection. Hyperlink objects expose their functionality just as workbooks and worksheets do, and therefore can be manipulated individually.

## Using the Application Object

The application object encapsulated by the TOpExcel class is the top-level component for controlling Excel using OfficePartner. All workbooks, worksheets, ranges, charts, and hyperlinks are owned and managed by an instance of this class. To manipulate any of the application-owned objects (such as workbooks, worksheets, etc) you need to drop a TOpExcel component onto a form. There are two properties of the TOpExcel class that must be set to allow OfficePartner to communicate with Excel: Connected and Visible. Setting the Connected property to True establishes the connection between OfficePartner and Excel. However, automation servers are automatically created in the hidden state. Setting the Visible property to True will actually display the Excel application. The following code assumes that you have dropped a TOpExcel component onto the form from the component palette and accepted the default component name of “OpExcel1”. The following lines of code establish connection programmatically:

```
OpExcel1.Connected := True;  
OpExcel1.Visible := True;
```

Of course, these property settings are available at design-time in the Object Inspector. Once these properties are set, we can further explore the capabilities of OfficePartner.

---

# Using Workbook Objects

## Creating a new workbook

You create a new Workbook object with the Workbooks collection's Add method. The Add method creates a new workbook, and immediately opens the workbook as well. The Add method also returns an object variable that represents the new workbook. The following code adds a workbook to a TOpExcel object:

```
var  
  NewWorkBook : TOpExcelWorkbook;  
begin  
  NewWorkBook := OpExcel1.Workbooks.Add;  
  {Set properties of the workbook as needed}  
  :  
end;
```

## Saving a workbook

You can save a new workbook by using the Workbook object's SaveAs method and specifying the name of the workbook you want to save. If a workbook by that name already exists, an error occurs. Once a workbook has been saved by using the SaveAs method, additional changes are saved by using the Workbook object's Save method.

Before you save a new workbook with the SaveAs method, the Workbook object's Name property is set to a value assigned by Excel, such as Book1.xls. After you save the workbook, the Name property contains the name you supplied in the Filename argument of the SaveAs method. The Name property is read-only; to change the name of a workbook, you must use the SaveAs method again, and pass a different value in the Filename argument.

The following code demonstrates the Save and SaveAs methods:

```
xl.Workbooks[0].Save;  
xl.Workbooks[0].SaveAs('c:\SaveTest.xls');
```

## Opening an existing workbook

The Workbooks collection's Add method is also used to open an existing workbook. The workbook opened with the Add method becomes the active workbook. You must supply a file name to be used with the Add method, or you can use the standard OpenDialog to let the user select the workbook to open. The following code opens the workbook "c:\SaveTest.xls" that we saved in the previous section.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ExistingWorkBook : TOpExcelWorkbook;
begin
  ExistingWorkBook := Xl.workbooks.Add;
  ExistingWorkBook.Filename := 'c:\SaveTest.xls';
end;
```

## Closing a workbook

Use a Workbook object's Close method to close an open workbook. To specify whether pending changes to the workbook should be saved before the object is closed, you use the SaveChanges argument. If the SaveChanges argument is omitted, the user is prompted to save pending changes. You can also use the Close method of the Workbooks object to close all open workbooks. If there are unsaved changes to any open workbook when this method is called, the user is prompted to save changes. If the user clicks Cancel in this Save dialog box, an error occurs. You can suppress the Save dialog box by setting the Application object's DisplayAlerts property to False before executing the Close method. When you use the Workbooks object's Close method in this manner, any unsaved changes to open workbooks are lost. After the Close method has executed, set the DisplayAlerts property back to True. The Close procedure and its arguments are described below:

```
Procedure Close(
  SaveChanges, FileName, RouteWorkbook,
  LCID : OleVariant);
```

The SaveChanges parameter is an optional parameter that determines whether changes to the workbook are saved. If SaveChanges is True, changes are saved to the workbook. If the workbook has not already been saved, a dialog appears prompting the user for a new file name. If SaveChanges is False, changes are discarded.

The FileName parameter sets the filename for the workbook to be saved.

The RouteWorkbook parameter determines whether the workbook is routed to the next recipient if it has an associated routing slip. If RouteWorkbook is True, the workbook is sent to the next recipient. If RouteWorkbook is False, the workbook is not sent to the next recipient.

According to Microsoft documentation, you should always pass a 0 as the final argument to this procedure.

Note: Many exposed functions and procedures of Microsoft Office's interface allow for optional parameters. OfficePartner has included the constant value EmptyParam to accommodate this feature. Any Microsoft parameter that is optional can be replaced with EmptyParam.

The following code closes the first workbook and prompts the user for a file name:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  WorkBook : TOpExcelWorkbook;
begin
  WorkBook := xl.Workbooks[0];
  Workbook.AsWorkbook.Close(
    True, EmptyParam, EmptyParam, 0);
end;
```

# Formatting Cell Contents

In the context of this manual, it is impractical to document each and every method and property exposed in Excel's type library. Instead, we demonstrate how to achieve the most commonly used features. The following sections explain the proper methods for formatting a cell or range of cells, as well as the cell's contents.

The examples in this section assume that you have defined a variable `r` of type `TOpExcelRange` as follows:

```
var
  r : TOpExcelRange;
begin
  r := OpExcel.Workbooks[0].Worksheets[0].Ranges.Add;
  r.Address := 'B2';
  r.SimpleText := '12345.6789';
```

## Formatting the font

The following section describes how to control the appearance of the characters within a cell or a range of cells. You can set the typeface, number format, font style, font size, underline, color, and font effect.

### Setting the font typeface

The font typeface is a string property of the font object associated with the range. Care must be taken to ensure that you assign to this property the exact name (case-insensitive) of an existing font name on the user's computer.

Example:

```
r.FontName := 'Viva Regular';
```

### Setting the number format

The number format determines how the cell contents are formatted within the cell and can only be accessed via the interface contained in the `AsRange` property. This property is a string type and is highly customizable. This property is accessed with this code:

```
r.AsRange.NumberFormat := NumberFormatString;
```

`NumberFormatString` is a string variable with a multitude of possible values. We have broken the formatting possibilities into groups: decimal points and significant digits, thousands separator, color and conditional formatting, dates, time, currency, percentage, scientific notation, and characters. Each of these topics is explained in detail in the following section.

## Decimal points and significant digits

Formatting of numeric values is accomplished using the following symbols:

#      Use this symbol as a placeholder for significant digits. If the value contains insignificant (trailing) zeros, this symbol suppresses them. To display insignificant zeros, use the 0 (zero) symbol described next. Values containing more digits than placeholders are rounded to the number of placeholders appearing after the decimal point. Values containing more digits to the left of the decimal point are unaffected by this symbol and are displayed as if there were placeholder symbols for each of the digits.

0      Use this symbol (zero) to display insignificant zeros. For example, to ensure that currency values consistently display 2 digits after the decimal point, two zero symbols should be placed in the string following the decimal point.

?      Use this symbol to pad values containing insignificant zeros. This is useful if you want values containing insignificant zeros to align with each other when formatted with a fixed-width font such as Courier New. This symbol is also effective for aligning fractions whose number of digits may vary.

Following are examples of using these symbols:

**Table 1:**

Raw Value	Desired Display Format	Required NumberFormat String
1234.56	1234.6	#####.##
2.3	2.30	.##00
.478	0.5	0.##
21.348	17.0	##.0
6.75	6 %	??/???
1.2	1.200	????.???
24.34	24.340	????.???
4478.567	44398.567	????.???

## Thousands separator

Commas are not used by default as thousands separators. To display commas as thousands separators, they must be included in the NumberFormat property string. Commas are also used to scale a value to a multiple of 1000. The following table illustrates the proper use this symbol:

**Table 2:**

Raw value	Desired display format	Required NumberFormat string
1234.6	1,234.60	#,###.##0
42000	42	#,
24600000	24.6	0.0,,

## Color and Conditional Formatting

Square brackets are used to denote both color and conditional formatting. They are also used to denote elapsed time, but we'll defer that use of brackets until the section covering formatting dates and times.

### Specifying color

Enclosing a color in square brackets sets the color for a section of the format to the specified color. For example, [Black] sets the selected text to black. The available colors are limited to black, blue, cyan, green, magenta, red, white, or yellow.

### Specifying conditional formatting

You can selectively apply formatting based on whether or not the cell's value meets a certain criteria. Square brackets are used to define these criteria. A condition statement consists of a comparison operator ( $<$ ,  $>$ ,  $=$ ,  $<=$ ,  $>=$ ) followed by a value. The operator and value are compared against the value contained in the cell. The format specified is only applied if the cell's value satisfies the condition.

For example, the following number format string displays negative values in red and positive values in black:

```
r.AsRange.NumberFormat := '[Red]<0];[Black]>=0];
```

## Formatting dates

The following table contains examples of formatting cells to display days, months, and years:

**Table 3:**

Format	Result
M	Displays months as 1-12
Mm	Displays months as 01-12
Mmm	Displays months as Jan-Dec
Mmmm	Displays months as January-December
Mmmmm	Displays months as the first letter of the month
D	Displays days as 1-31
Dd	Displays days as 01-31
Ddd	Displays days as Sun-Sat
Dddd	Displays days as Sunday-Saturday
Yy	Displays years as 00-99
YYYY	Displays years as 1900-9999

**Caution:** If you use “m” immediately after the “h” or “hh” code or immediately before the “ss” code, Microsoft Excel displays minutes instead of the month.

## Formatting time

The following table contains examples of formatting cells to display hours, minutes, and seconds:

**Table 4:**

Format	Result
H	Displays hours as 0-23
Hh	Displays hours as 00-23
M	Displays minutes as 0-59
Mm	Displays minutes as 00-59
S	Displays seconds as 0-59
Ss	Displays seconds as 00-59
H AM/PM	Appends AM or PM following the time: 4 AM
h:mm AM/PM	Appends AM or PM following the time: 4:36 PM

**Table 4:**

<b>Format</b>	<b>Result</b>
H:mm:ss A/P	Appends A or P following the time: 4:36:03 P
[h]:mm	Displays elapsed time in hours: 25:02
[mm]:ss	Displays elapsed time in minutes: 63:46
[ss]	Displays elapsed time in seconds
h:mm:ss.00	Displays fractions of a second

**Caution:** If the format contains an AM or PM, the hour is based on the 12-hour clock, where “AM” or “A” indicates times from midnight until noon and “PM” or “P” indicates times from noon until midnight. Otherwise, the hour is based on the 24-hour clock. The “m” or “mm” code must appear immediately after the “h” or “hh” code or immediately before the “ss” code; otherwise, Microsoft Excel displays the month instead of minutes.

### Currency

To enter one of the following currency symbols in a number format, turn on NumLock and use the numeric keypad to enter the ANSI code for the symbol.

**Table 5:**

<b>Desired currency symbol</b>	<b>Keystrokes</b>
¢	Alt+0162
£	Alt+0163
¥	Alt+0165
€	Alt+0128

**Note:** Custom formats are saved with the workbook. To have Microsoft Excel always use a specific currency symbol, change the currency symbol selected in Regional Settings in Control Panel before you start Excel.

### Percentage

Values can be expressed as percentages by using the percent sign (%) in the number format string. For example, 0.32 appears as 32% whereas 3.45 appears as 345%.

### Scientific notation

Cell values can be displayed in scientific notation by including the following symbols in the number format string: E-, E+, e-, and e+

The case that E (or e) appears in the number format string determines the case of E (or e) after the cell has been formatted. Use of the minus sign (-) places a minus sign by negative exponents only. Positive exponents appear without a sign. Use of the plus sign (+) places a minus sign by negative exponents and a plus sign by positive exponents.

**Note:** If a format contains a 0 or # to the right of an exponent code, Excel displays the number in scientific format and inserts an “E” or “e”. The number of 0’s or #’s to the right of a code determines the number of digits in the exponent. E- or e- places a minus sign by negative exponents. E+ or e+ places a minus sign by negative exponents and a plus sign by positive exponents.

### Characters

The following symbols can be entered in the number format string and translated directly into the formatted cell contents without any intervention:

\$	dollar sign	!	factorial	}	right curly bracket
-	hyphen	^	caret	=	equal sign
+	plus sign	&	ampersand	<	less than sign
/	forward slash	`	left single quotation mark	>	greater than sign
(	left parentheses	'	right single quotation mark		the space character
)	right parentheses	~	tilde		
:	colon	{	left curly bracket		

Additional characters or strings can be added to the cell by including any of the following symbols:

**Table 6:**

Symbol	Description
" " (double quotes)	Use a string of characters enclosed in double quotes to include the string in the formatted cell.
\ (backslash)	A single character preceded by a backslash in the number format string includes the single character in the formatted cell.
@ (at sign)	Include a section of text. If included, a text section is always the last section in the number format. Include an at sign (@) in the section where you want to display any text entered in the cell. If the @ character is omitted from the text section, text you enter is not displayed. If you want to always display specific text characters with the entered text, enclose the additional text in double quotation marks (" ") - for example, "total sales for"@. If the format does not include a text section, text you enter is not affected by the format.
spaces	Spaces can be added to the number format string by including the underscore symbol ( _ ). This can be useful to coerce negative and positive values to line up on the decimal point. Simply precede all positive values with an underscore, which compensates for the additional space added by the negative sign.
* (asterisk)	Use an asterisk in the number format string to instruct Excel to fill the cell (to column width) with the character immediately following the asterisk.

The following examples illustrate the default number formats used by Microsoft Excel and how they are accessed from within OfficePartner. These examples can be used as is or they can be modified as discussed in the previous section. Either way, these examples are a good springboard into Excel's world of number formatting.

```

r.AsRange.NumberFormat := 'General'
r.AsRange.NumberFormat := '0'
r.AsRange.NumberFormat := '#,##0'
r.AsRange.NumberFormat := '#,##0.00'
r.AsRange.NumberFormat := '#,##0_);(#,##0)'
r.AsRange.NumberFormat := '#,##0_);[Red](#,##0)'
r.AsRange.NumberFormat := '#,##0.00_);(#,##0.00)'
r.AsRange.NumberFormat := '#,##0.00_);[Red](#,##0.00)'
r.AsRange.NumberFormat := '$#,##0_);($#,##0)'
r.AsRange.NumberFormat := '$#,##0_);[Red]($#,##0)'
r.AsRange.NumberFormat := '$#,##0.00_);($#,##0.00)'
r.AsRange.NumberFormat := '$#,##0.00_);[Red]($#,##0.00)'
r.AsRange.NumberFormat := '0%'
r.AsRange.NumberFormat := '0.00%'
r.AsRange.NumberFormat := '0.00E+00'
r.AsRange.NumberFormat := '###0.0E+0'
r.AsRange.NumberFormat := '# ?/?'
r.AsRange.NumberFormat := '# ??/?'
r.AsRange.NumberFormat := 'm/d/yyyy'
r.AsRange.NumberFormat := 'd-mmm-yy'
r.AsRange.NumberFormat := 'd-mmm'
r.AsRange.NumberFormat := 'mmm-yy'
r.AsRange.NumberFormat := 'h:mm AM/PM'
r.AsRange.NumberFormat := 'h:mm:ss AM/PM'
r.AsRange.NumberFormat := 'h:mm'
r.AsRange.NumberFormat := 'h:mm:ss'
r.AsRange.NumberFormat := 'm/d/yyyy h:mm'
r.AsRange.NumberFormat := 'mm:ss'
r.AsRange.NumberFormat := 'mm:ss.0'
r.AsRange.NumberFormat := '@'
r.AsRange.NumberFormat := '[h]:mm:ss'
r.AsRange.NumberFormat :=
    '_($* #,##0_);_($* (#,##0);_($* "-"_) ;_(@_) '
r.AsRange.NumberFormat :=
    '_(* #,##0_);_(* (#,##0);_(*      "-"_) ;_(@_) '
r.AsRange.NumberFormat :=
    '_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_) ;_(@_) '
r.AsRange.NumberFormat :=
    '_(* #,##0.00_);_(* (#,##0.00);_(*      "-"??_) ;_(@_) '

```

## Setting the font style

The font style is a string property of the range. Care must be taken to ensure that you assign to this property the exact name (case-insensitive) of an existing font style. The four possible font styles are as follows:

**Table 7:**

Desired font	String representation
Regular	Regular
Bold	Bold
Italic	Italic
Bold Italic	Bold Italic

The following examples show you how to set each of these font styles.

```
r.FontName := 'Regular';
r.FontName := 'Italic';
r.FontName := 'Bold';
r.FontName := 'Bold Italic';
```

## Setting the font size

The font size is an integer property of the range. The font size of an Excel cell's contents can be set in the following manner:

```
r.FontSize := 24;
```

## Setting the underline style

The underline style is an enumerated property type of the font object associated with the range. Excel defines these enumerated constants as follows:

**Table 8:**

XlUnderlineStyle constants	Resulting underline style
XlUnderlineStyleNone	No underline
XlUnderlineStyleSingle	Single underline
XlUnderlineStyleDouble	Double underline
XlUnderlineStyleSingleAccounting	Single accounting underline

The following examples show you how to set each of these underline styles.

```
r.AsRange.Font.Underline := xlUnderlineStyleNone;
r.AsRange.Font.Underline := xlUnderlineStyleSingle;
r.AsRange.Font.Underline := xlUnderlineStyleDouble;
r.AsRange.Font.Underline :=
    xlUnderlineStyleSingleAccounting;
```

### Setting the font color

The `FontColor` property is a `TColor` property of the range. The font color can be set directly, or can be set by including the color directive in the `NumberFormat` string as explained earlier. If you set the font color by specifying the color directive in the `NumberFormat` string, you are limited to only eight colors. Excel's type library defines 40 color constants that can be used to set the font color directly. In addition, since Excel uses 24-bit color, you can set the font color to one of over 16 million plus colors. To set the font color to a custom color, you must employ the `RGB` method to convert red, blue, and green components to an `OleColor`.

To set the color directly using a color constant, simply supply the color constant as follows:

```
r.FontColor := clTeal;
```

To set the color using the `RGB` function, set the `FontColor` property as follows:

```
r.FontColor := RGB(0,255, 0);
```

The preceding example sets the font color to pure green.

### Setting the font effects

Font effects are achieved by adding members to or removing members from the `FontAttributes` set property of the range. Font effects can include any combination of bold, italic, underline, and strikethrough. `OfficePartner` defines the following enumerations that may be added to the `FontAttributes` set property:

**Table 9:**

<b>FontAttribute set element</b>	<b>Result</b>
<code>xlfaBold</code>	<code>Bold</code> type
<code>xlfaItalic</code>	<code>Italic</code> type
<code>xlfaUnderline</code>	<code>Single underline</code>
<code>xlfaStrikeThrough</code>	<code>Strike-through</code>

The following example demonstrates how to set the font style to bold and italic:

```
r.FontAttributes := [xlfBold, xlfItalic];
```

## Setting cell alignment

### Horizontal alignment

HorizontalAlignment is an enumerated property type of the range object. The following enumerations are available within OfficePartner for setting horizontal alignment:

**Table 10:**

HorizontalAlignment constant	Resulting alignment
xlHAlignCenter	Center
xlHAlignJustify	Justified
xlHAlignLeft	Left
xlHAlignRight	Right

The following code right-justifies the text in the range r:

```
r.HorizontalAlignment := xlchaRight;
```

### Vertical alignment

VerticalAlignment is also an enumerated property type of the range object. The following enumerations are available within OfficePartner for setting vertical alignment:

**Table 11:**

VerticalAlignment constant	Resulting alignment
xlVAlignBottom	Bottom
xlVAlignCenter	Center
xlVAlignJustify	Justified
xlVAlignTop	Top

The following code vertically centers the text contained in the cell:

```
R.VerticalAlignment := xlcvCenter;
```

## Orientation

Orientation is an enumerated property type of the range object. The following enumerations are available from within OfficePartner for setting the orientation of text within a cell:

**Table 12:**

Orientation constant	Resulting orientation
XlDownward	Downward
XlHorizontal	Horizontal
XlUpward	Upward
XlVertical	Vertical
-90 to 90	Specifies degrees of text rotation

The following code demonstrates how to rotate text 45 degrees CCW from the horizontal:

```
R.Orientation := xlcoRotated;
R.RotateDegrees := 45;
```

## Indent

IndentLevel is an integer property type of the range object. This property determines how many spaces that text within the cell is indented.

**Caution:** This value must be between 0 and 15.

The following line of code demonstrates how to set this property:

```
R.IndentLevel := 15;
{must be between 0 and 15 (inclusive)}
```

## Control

Control is actually surfaced through two Boolean properties of the range object. These properties determine the interaction between text size and column width. The two Boolean values that control this interaction are WrapText and ShrinkToFit. Each of these properties can be set as follows:

```
r.WrapText := True;
r.ShrinkToFit := True;
```

## Setting cell borders

The individual borders of an Excel cell are maintained in a collection of borders. This collection differs from other collections in that items cannot be added or removed from the collection. The number of valid Excel borders contained in this collection is fixed at eight. The constants referring to each of these borders are: `XlDiagonalDown`, `XlDiagonalUp`, `XlEdgeBottom`, `XlEdgeLeft`, `XlEdgeRight`, `XlEdgeTop`, `XlInsideHorizontal`, and `XlInsideVertical`.

Accessing each of the borders individually is a simple process of indexing their position within the borders collection by the above named constants. For example:

```
r.AsRange.Borders[xlEdgeBottom].(other methods..)
```

**Caution:** `xlInsideHorizontal` and `xlInsideVertical` apply only when a range of cells is included in the range. Do not attempt to index these borders if the range consists of a single cell.

### Border line style

`LineStyle` is a property of each of the Excel borders. Excel provides enumerated constants to define the line style. These constants are: `XlContinuous`, `XlDash`, `XlDashDot`, `XlDashDotDot`, `XlDot`, `XlDouble`, `XlSlantDashDot`, and `XlLineStyleNone`.

The following line of code sets the bottom border of the cell(s) in the range to a double line:

```
r.AsRange.Borders[xlEdgeBottom].LineStyle := xlDouble;
```

### Border line weight

Similar to the `LineStyle` property, `Weight` is a property of each of the Excel borders. Excel provides enumerated constants that define the line weight. These constants are: `XlHairline`, `XlThin`, `XlMedium`, and `XlThick`.

The following line of code sets the bottom border of the cell(s) in the range to thick:

```
r.AsRange.Borders[xlEdgeBottom].Weight := xlThick;
```

### Border line color

Similar to the `LineStyle` and `Weight` properties, `Color` is a property of each of the Excel cell borders. The color property is set using the `RGB` function. The following line of code demonstrates how to set the bottom border to green:

```
r.AsRange.Borders[xlEdgeBottom].Color := RGB(0,255,0);
```

## A shortcut to setting all borders

As an alternative to setting each of the cell's border's individually, the Range interface provides a method to set all borders to the same values in one fell swoop. Here is the declaration of the method:

```
BorderAround(
    LineStyle,Weight,ColorIndex,Color : OleVariant)
```

The following table defines the constants that can be used for LineStyle, Weight, and ColorIndex. Color must be set using the RGB function.

**Table 13:**

LineStyle	Weight	ColorIndex
XlContinuous*	XlHairline	XlColorIndexAutomatic*
XlDash	XlThin*	XlColorIndexNone
XlDashDot	XlMedium	
XlDashDotDot	XlThick	
XlDot		
XlDouble		
XlLineStyleNone		
XlSlantDashDot		

\* Default values

The following example displays a continuous, thick, green border around the entire cell:

```
r.AsRange.BorderAround(
    XLContinuous, XlThick, XlColorIndexAutomatic,
    RGB(0,255,0));
```

## Setting cell shading

The Range interface provides another interface named Interior. There are three significant properties of the Interior interface that control the shading and patterns of the cell: Color, Pattern, and PatternColor.

### Color

The Color property sets (or returns) the primary interior color of the cell. Use the RGB function to specify the cell color. The following example fills the interior of the cell in green:

```
r.AsRange.Interior.Color := RGB(0,255,0);
```

## Pattern

The Pattern property of the interior defines a pattern used to draw the cell's interior. The pattern is always drawn using the color defined in the PatternColor property.

Pattern is an enumerated type that can accept any of the following constants:

**Table 14:**

Constant	Result
xlPatternAutomatic	Excel default
XlPatternChecker	Dark NE/SW and NW/SE diagonal lines
XlPatternCrissCross	Light NE/SW and NW/SE diagonal lines
XlPatternDown	Dark NW/SE diagonal lines only
xlPatternGray16	Lighter shading
xlPatternGray25	Medium shading
xlPatternGray50	Heavy shading
xlPatternGray75	Heaviest shading
xlPatternGray8	Lightest shading
XlPatternGrid	Dark horizontal and vertical lines
XlPatternHorizontal	Dark horizontal lines only
XlPatternLightDown	Light NW/SE diagonal lines only
XlPatternLightHorizontal	Light horizontal lines only
XlPatternLightUp	Light SW/NE diagonal lines only
XlPatternLightVertical	Light vertical lines only
XlPatternNone	No shading
xlPatternSemiGray75	Course heavy shading
XlPatternSolid	Solid shading
XlPatternUp	Dark SW/NE diagonal lines only
xlPatternVertical	Dark vertical lines only

The following line of code sets the cell interior to a criss-cross pattern:

```
r.AsRange.Interior.Pattern := xlPatternCrissCross;
```

## PatternColor

The PatternColor defines what color the pattern should be drawn in. Once again, the RGB function is used to return a color that you can use to set this property.

The following lines of code draw a blue criss-cross pattern in the cell:

```
r.AsRange.Interior.Pattern := xlPatternCrissCross;  
r.AsRange.Interior.PatternColor := RGB(0,0,255);
```

## Sizing rows and columns

Sizing rows and columns is a simple matter in OfficePartner. OfficePartner provides you with two simple properties, RowHeight and ColumnWidth, to set the row height(s) and column width(s) respectively. The following example demonstrates how to set these property values:

```
r.ColumnWidth := 30;  
r.RowHeight := 30;
```

These property settings force both the column width and row heights to 30.

You may also find it very useful to be able to set the column width and row height according to the size of the cell's contents. Within OfficePartner, this is a simple process of calling the AutoFit method of the Columns and Rows objects. For example:

```
r1.Columns.AutoFit;  
r1.Rows.AutoFit;
```

## Implementing Excel functions

Functions may be added to a cell within a worksheet simply by setting the SimpleText property of the cell. Consult Microsoft's documentation for the availability of functions and required syntax and parameters.

The following line of code inserts the Cos function in cell A2 and passes as a parameter the value of Cell A1:

```
var  
  r1 : TOpExcelRange;  
  r2 : TOpExcelRange;  
begin  
  r1 := OpExcel.Workbooks[0].Worksheets[0].Ranges.Add;  
  r1.Address := 'A1';  
  r1.SimpleText := '34';  
  
  r2 := OpExcel.Workbooks[0].Worksheets[0].Ranges.Add;  
  r2.Address := 'A2';  
  r2.SimpleText := '=Cos(A1)';  
end;
```

---

## Microsoft Excel Demo

The following demonstration guides you step by step through of the process of displaying and manipulating database data in Excel spreadsheets. We'll show you how to create workbooks, worksheets, and ranges. We'll also attach datasets to existing worksheets and show you how to copy the dataset's contents into Excel.

The demo uses two Paradox database tables, customers.db and orders.db, in a master-detail relationship. These tables are provided with Delphi in the DBDEMOS alias, and with C++Builder in the BCDEMOS alias. Clicking a customer in the uppermost dbGrid (connected to customers.db) filters the lower dbGrid (connected to orders.db) to display only orders pertaining to the selected customer.

### Setting properties and getting connected

The obvious first step is to write the code that launches an instance of Excel. The code that launches Excel is found in the demo application's ExcelConnect procedure. Before calling that procedure, the demo gives the user the capability of setting fundamental properties of the Excel application object. XLPropForm is a simple form that allows users to specify the settings of Excel when it is launched. Excel's parameters are extracted from this form and

assigned to TOpExcel property values before actually launching Excel. This serves only to show some of the fundamental startup properties of the Excel application object. The following code accomplishes this:

```
procedure TfrmExcelDemo.LaunchExcelBtnClick(
  Sender: TObject);
var
  XLPropForm : TfrmExcelProp;
  i : Integer;
begin
  XLPropForm := nil;
  if OpExcel.Connected = True then exit;
  try
    XLPropForm := TfrmExcelProp.Create(Self);
    if XLPropForm.ShowModal = mrOK then begin
      with OpExcel, XLPropForm do begin
        {must be fully qualified - (caption ambiguity)}
        OpExcel.Caption := XLPropForm.CaptionEdit.Text;
        {Set Excel properties}
        EnableAnimations := EnableAnimationsCb.Checked;
        EnableAutoComplete := EnableAutoCompleteCb.
          Checked;
        Interactive := InteractiveCb.Checked;
        LargeButtons := LargeButtonsCb.Checked;
        ServerHeight := StrToInt(HeightEdit.Text);
        ServerLeft := StrToInt(LeftEdit.Text);
        ServerTop := StrToInt(TopEdit.Text);
        ServerWidth := StrToInt(WidthEdit.Text);
        {Launch Excel}
        ExcelConnect;
        {Add default workbook}
        OpExcel.Workbooks.Add;
        {now, add 2 more worksheets (a default book
        was added when we added the workbook)}
        for i := 1 to 2 do
          OpExcel.Workbooks[0].Worksheets.Add;
      end;
    end;
    finally
      XLPropForm.Free;
    end;
  end;
```

The LaunchExcelBtnClick procedure calls ExcelConnect which sets two properties of the Excel application object:

```
procedure TfrmExcelDemo.ExcelConnect;
begin
  {launch empty instance of Excel (no workbooks open)}
  OpExcel.Connected := True;
  {Office applications start up hidden by default.
  To show Excel, all we need to do is set its
  visible property to True.}
  OpExcel.Visible := True;
end;
```

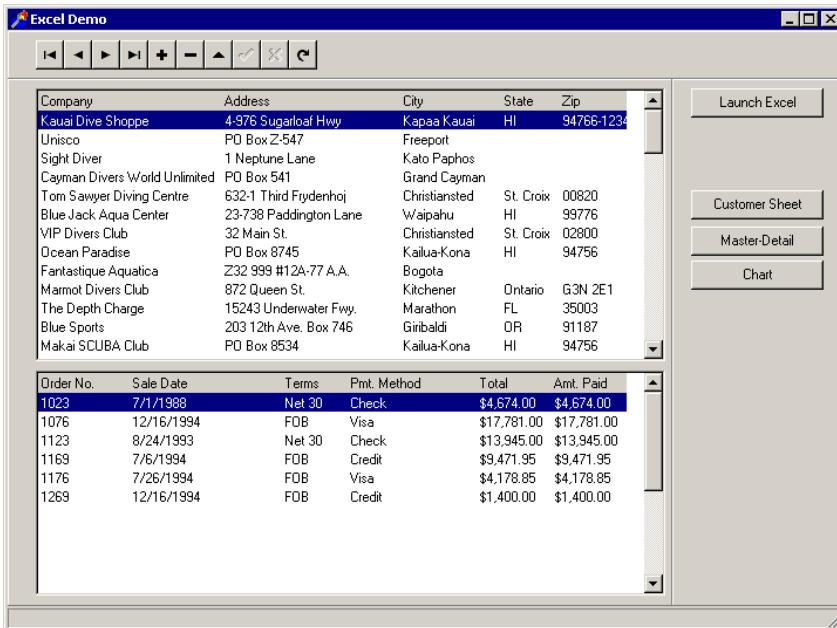
This procedure only sets the Connected property to True, which does the real work of launching Excel, and sets the Visible property to True. By default, automation servers start as hidden windows. In many instances this may be desirable, but in this demo, we want to see what effect our client program has on the Excel application object.

## Adding a workbook and worksheets

When Excel initially starts, there are no workbooks and consequently no worksheets. The following lines of code add a workbook and two additional worksheets for a grand total of three worksheets:

```
{Add default workbook}
OpExcel.Workbooks.Add;
{now, add 2 more worksheets (a default sheet was added when we
added the workbook)}
for i := 1 to 2 do
  OpExcel.Workbooks[0].Worksheets.Add;
```

Workbooks and worksheets can be added at design-time using the nested collection editor. To invoke the editor, click the ellipsis button of the Workbooks property of the TOpExcel component. The workbook and worksheets were added in code merely to show how to achieve the same effect programmatically. The following figure shows the demonstration program running and is referred to in subsequent examples:



---

## Exporting all customers to Excel

Clicking the button captioned “Customer Sheet” demonstrates how to create and populate a range using a TOpDataSetModel as a conduit between an Excel range and the customers table. The code that accomplishes this is found in the btnCustomerClick procedure and is as follows:

```
procedure TfrmExcelDemo.btnCustomerClick(  
  Sender: TObject);  
var  
  Rng : TOpExcelRange;  
begin  
  {the following 5 properties can be set at design-time.  
  They are set here in code for demonstration purposes  
  only}  
  {add a range to the first worksheet}  
  Rng := OpExcel.Workbooks[0].Worksheets[0].Ranges.Add;  
  {Give the name a range (simplifies access)}  
  Rng.Name := 'CustomerRange';  
  {We'll only set the anchor cell since we are  
  populating the range through a dataset (unknown  
  columns & rows)}  
  Rng.Address := 'A1';  
  {associate the DataSeodel with the range}  
  Rng.OfficeModel := dmExcelDemo.mdlCustomers;  
  {associate the DataSeodel with the DataSet}  
  dmExcelDemo.mdlCustomers.Dataset :=  
    dmExcelDemo.tblCustomers;  
  
  {Fill the first worksheet (starting with the anchor  
  cell) with every data column and row in the DataSet}  
  OpExcel.RangeByName['CustomerRange'].Populate;  
  {Activate the first worksheet}  
  OpExcel.Workbooks[0].Worksheets[0].Activate;  
  
  {Sets the column width of all cell in the range to 5}  
  Rng.ColumnWidth := 5;  
end;
```

We should call your attention to the fact that three TOpDataSetModel components have been added to the data module (udmExcelDemo.pas) and each connected to one of the three TDataSetDescendants found there.

Every property assignment statement made in the preceding code could have easily been set at design time, but we set them at runtime to show you how it should be done programmatically as well as to comment each line to explain what the effect the property settings have.

In summary, this procedure adds a Range to the first worksheet and gives it a name of “CustomerRange”. We then set the Address of this Range to A1. As you’ll see in a moment (when we get to the Populate method), this address corresponds to the anchor cell (upper-left) of the Range.

Now we link this new range to the customers.db table with the following lines of code:

3

```
{associate the DataSeodel with the range}
Rng.OfficeModel := dmExcelDemo.mdlCustomers;
{associate the DataSeodel with the DataSet}
dmExcelDemo.mdlCustomers.DataSet :=
    dmExcelDemo.tblCustomers;
```

Now that this link is established, we call the Populate method of the Range. This copies every field and every record to the Excel worksheet starting at the AnchorCell.

```
OpExcel.RangeByName[ 'CustomerRange' ].Populate;
```

Finally, for the sake of aesthetics, we instruct Excel to auto size all of the columns with the last line of the procedure:

```
Rng.Columns.AutoFit;
```

This section has essentially demonstrated how to transfer the contents of a database table (or query/stored procedure) to an Excel worksheet. With the one exception of the Populate call, this example could have been set up entirely at design time.

## Exporting a single record of a dataset to Excel

Clicking the button captioned “Master Detail” demonstrates how to populate an Excel worksheet with the contents of a single record of a DataSet descendant. This example transfers the current record in the master dataset (customers) to Cell A1 and the entire contents of the detail dataset (orders) immediately below in cell A3. As in the above example, these cell addresses define the anchor cell of the range and not the entire range. The output of this example is transferred to the second worksheet leaving the first intact.

This example creates two ranges instead of one. One range becomes the master record, and the second range displays the detail records. The only other difference is that the PopulateCurrent method is invoked on the master dataset. This has the effect of copying only the current record in the master dataset into the Excel worksheet.

The setup code for this example is nearly identical to that of the previous example. For brevity’s sake, the code is not duplicated here. The only difference is that we have added a second range at design-time and called it “Orders.” This range will be populated by the detail dataset using the Populate method as in the preceding example. Instead of calling the Populate method of the range associated with the master dataset (CustomerRange) we call the PopulateCurrent method as follows:

```
OpExcel.RangeByName[ 'CustomerRange' ].PopulateCurrent;
```

This statement copies the fields in the current record of the master dataset to the spreadsheet starting in the anchor cell. The final lines of this procedure serve only to format the cells of the master record to differentiate them from the records of the detail records. For more information on formatting cells, see “Formatting Cell Contents” on page 39.

## Adding a chart

The final example borrows from the first two examples with regards to creating, naming, and populating ranges. Clicking the button captioned “Chart” demonstrates how to add a chart to the third worksheet of the workbook.

```
{Add and position a chart to the worksheet}
Chart := OpExcel.Workbooks[0].Worksheets[2].Charts.Add;
Chart.Left := 350;
Chart.Width := 550;
Chart.Height := 400;

{We'll create a pie chart..}
Chart.ChartType := xlct3DPie;
{Tell the chart where to find the data}
Chart.DataRange := 'A1:C100';

Rng.Columns.AutoFit;

{Close the query}
dmExcelDemo.qryChartData.Active := False;
OpExcel.Workbooks[0].Worksheets[2].Activate;
```

The first four lines of code simply add a chart to a worksheet and set its size and position in pixels.

The next step is to set the type of chart, a pie chart in this case. The following line of code employs one of Microsoft’s defined constants to set the type of chart:

```
Chart.ChartType := xlct3DPie;
```

The final step is to set the data range of the chart. This range defines the range that Excel uses to build the chart. In this example, we’ll use the range that was populated from the query (qryChartData) result. Once the range is defined and populated, Excel does the rest of the work to build the chart for you.

---

## TOpExcelBase Class

TOpExcelBase class is the immediate ancestor of the TOpExcel component. It implements all of the methods and properties used by the TOpExcel component and is identical to the TOpExcel component except that no properties are published.

TOpExcelBase is provided to facilitate creation of descendent components. For property and method descriptons, see “TOpExcel Component” on page 61.

### Hierarchy

TComponent (VCL)

TOpOfficeComponent (OpOfficeShared)

TOpExcelBase (OpExcel2k)





# TOpExcel Component

3

The TOpExcel component represents an instance of Microsoft Excel. This class contains a collection representing all workbooks in Excel and worksheets, ranges, charts, and hyperlinks within each Workbook. The PropDirection property determines whether data is sent from your application to Excel or whether data is being acquired from an existing Excel worksheet.

PropDirection controls what happens to the component properties when the component is initially connected. If PropDirection is pdToServer, the streamed design-time, properties are pushed to Excel immediately after Excel is launched. If PropDirection is set to pdFromServer, the streamed properties are ignored and the component properties represent the state of Excel when it is launched.

## Hierarchy

TComponent (VCL)

- ① TOpOfficeComponent (OpOfficeShared)443
  - TOpExcelBase (OpExcel2k)
    - TOpExcel (OpExcelClient)

## Properties

Caption	① MachineName	ServerTop
① ClientState	① OfficeVersion	ServerWidth
① Connected	① OfficeVersionStr	UserName
EnableAnimations	① PropDirection	Visible
EnableAutoComplete	RangeByName	WindowState
EnableCancelKey	Server	Workbooks
Interactive	ServerHeight	
LargeButtons	ServerLeft	

## Methods

① AddConnectListener	CreateItem	GetFileInfo
① CoCreate	Destroy	HandleEvent
Create	GetAppInfo	① RemoveConnectListener

## Events

BeforeSheetDoubleClick	OnSheetCalculate	OnWorkbookActivate
BeforeSheetRightClick	OnSheetChange	OnWorkbookAddinInstall
BeforeWorkbookClose	OnSheetDeactivate	OnWorkbookAddinUnin...
BeforeWorkbookPrint	OnSheetSelectionChange	OnWorkbookDeactivate
BeforeWorkbookSave	OnWindowActivate	OnWorkbookNewSheet
OnNewWorkbook	OnWindowDeactivate	OnWorkbookOpen
OnSheetActivate	OnWindowResize	

## Reference Section

### BeforeSheetDoubleClick

event

```
property BeforeSheetDoubleClick: TBeforeSheetDoubleClick
TBeforeSheetDoubleClick = procedure (
  Sender : TObject; const Sh : IDispatch;
  const Target : ExcelRange;
  var Cancel : WordBool) of object;
```

Defines an event handler that is called when the user double-clicks on a cell in the Excel spreadsheet.

Sender is the TOpExcel component that initiated the event. Sh is the IDispatch interface to the Excel worksheet that receives the double-click. Target is a dispinterface to a TOpExcelRange object. The Address property of Target contains the cell reference in A1 format of the cell that received the double-click. To abort processing of this event, set Cancel to False.

See also: BeforeSheetRightClick

**BeforeSheetRightClick****event**

```
property BeforeSheetRightClick: TBeforeSheetRightClick
TBeforeSheetRightClick = procedure (
  Sender : TObject; const Sh : IDispatch;
  const Target : ExcelRange;
  var Cancel : WordBool) of object;
```

- Defines an event handler that is called when the user right-clicks on a cell in the Excel spreadsheet.

Sender is the TOpExcel component that initiated the event. Sh is the IDispatch interface to the Excel worksheet that receives the right-click. Target is a dispinterface to a TOpExcelRange object. The Address property of Target contains the cell reference in A1 format of the cell that received the right-click. To abort processing of this event, set Cancel to False.

See also: BeforeSheetDoubleClick

**BeforeWorkbookClose****event**

```
property BeforeWorkbookClose: TBeforeWorkbookClose
TBeforeWorkbookClose = procedure (
  Sender : TObject; const Wb : Workbook;
  var Cancel : WordBool) of object;
```

- Defines an event handler that is called prior to the user closing the workbook.

Sender is the TOpExcel component that initiated the event. Wb is the IDispatch interface to the Excel workbook that is being closed. To prevent the workbook from being closed, set Cancel to True.

See also: OnSheetDeactivate, OnWindowDeactivate, OnWorkbookDeactivate

**BeforeWorkbookPrint****event**

```
property BeforeWorkbookPrint: TBeforeWorkbookPrint
TBeforeWorkbookPrint = procedure (
  Sender : TObject; const Wb : Workbook;
  var Cancel : WordBool) of object;
```

- Defines an event handler that is called prior to the user printing the workbook.

Sender is the TOpExcel component that initiated the event.

Wb is the IDispatch interface to the Excel workbook that is being printed.

To prevent the workbook from being printed, set Cancel to True in the event handler for this event.

See also: BeforeWorkbookClose, BeforeWorkbookSave

### **BeforeWorkbookSave**

**event**

```
property BeforeWorkbookSave: TBeforeWorkbookSave
TBeforeWorkbookSave = procedure (
    Sender : TObject; const Wb : Workbook;
    SaveAsUI : WordBool; var Cancel : WordBool) of object;
```

Defines an event handler that is called prior to the user saving the workbook.

Sender is the TOpExcel component that initiated the event. Wb is the IDispatch interface to the Excel workbook that is being saved. If SaveAsUI is True, the user is attempting to save the document using the SaveAs method. If SaveAsUI is False, the user is attempting to save the document using the Save command. To prevent the workbook from being saved, set Cancel to True.

See also: BeforeWorkbookPrint, BeforeWorkbookClose

### **Caption**

**property**

```
property Caption : string
```

Defines a text string displayed in the Caption of Microsoft Excel.

This is the string that appears on Excel's title bar.

See also: Connected, Visible

### **Connected**

**property**

```
property Connected : Boolean
```

Specifies whether the component is connected to a live instance of Excel.

The OfficePartner components are designed to allow the setting of properties whether the server is running or not. When Connected is set to True, Excel is launched and properties are initialized in accordance with the setting of the PropDirection property. When Connected is set to False, automation interfaces are released to allow Excel to properly shut down.

See also: OnNewWorkbook, OnWindowActivate, PropDirection, Visible,

**Create****constructor**


---

```
constructor Create(AOwner : TComponent);
```

- ↳ Creates an instance of TOpExcel component.

A Workbooks collection is also created with a default workbook added to the collection.

See also: Connected, PropDirection, Visible

**CreateItem****method**


---

```
function CreateItem(
  Item : TOpNestedCollectionItem) : IDispatch;
```

- ↳ Allows the component to properly synchronize and manage CoClass/Interface retrieval and maintenance.

All Collection children delegate initial automation connections to this method. Developers should not call this method directly.

See also: Create

**Destroy****destructor**


---

```
destructor Destroy;
```

- ↳ Destroys the instance of TOpExcel.

The Excel destructor cleans up all owned collections and ensures that all COM interfaces are released in order to facilitate proper shutdown of the Excel automation server.

See also: Create, CreateItem

**EnableAnimations****property**


---

```
property EnableAnimations : WordBool
```

- ↳ Specifies whether Excel shows delayed animation when rows, cols, etc. are inserted or deleted.

If EnableAnimations is True, Excel's default animations occur when rows and columns are inserted or deleted. Otherwise, rows and columns are deleted and inserted without the animations.

---

**EnableAutoComplete** property

```
property EnableAutoComplete : WordBool
```

↳ Sets the Excel AutoComplete property.

EnableAutoComplete corresponds to the “Enable Autocomplete for all cell values” option (from Excel’s main menu, select Tools | Options, and then click the Edit tab).

---

**EnableCancelKey** property

```
property EnableCancelKey : TOpXLCancelKey
```

↳ Specifies whether Ctrl-C can be used to interrupt running Excel macros.

Determines the behavior of the Cancel (Ctrl-C) key when running a macro. Refer to MSDN documentation for described behavior.

The following properties result in these behaviors when Ctrl-C is entered during a macro.

**Table 15:**

<b>Value</b>	<b>Description</b>
XlckDisabled	Disables user invoked interrupts.
XlckErrorHandler	Traps user interrupts and allows the macro to gracefully handle the interrupt.
XlckInterrupt	User interrupts are effective immediately.

---

**GetAppInfo** method

```
procedure GetAppInfo(Info : TStrings);
```

↳ Returns system/application information from the automation server in “Key=Value” pairs.

Info contains the strings that are filled with system/application information. If the component is not connected, GetAppInfo temporarily launches the server.

See also: Connected, GetFileInfo, PropDirection

**GetFileInfo****method**

```
procedure GetFileInfo(var Filter, DefExt : string);
```

- ↳ Returns filename extensions supported by this component.

Filter is filled with filter names and extensions supported by the TOpExcel component. DefExt is filled with a file extension that is appended automatically to a selected file named in the Save and SaveAs dialogs. This property contains the file type extensions that are displayed in the Save and SaveAs dialogs.

See also: [GetAppInfo](#), [PropDirection](#)

**HandleEvent****method**

```
procedure HandleEvent(
  const IID : TIID; DispId : Integer;
  const Params : TVariantArgList);
```

- ↳ Overridden in each TOpOfficeComponent subclass in order to correctly dispatch automation events to VCL Event Handlers.

IID contains the interface identifier common to all TOpExcel events. DispID contains an integer that maps a general TOpExcel event to the specific event. For example, a DispId value of 1558 maps an event to the OnSheetSelectionChange event. Params contains an array of variants whose values depend on the specific event being triggered.

This method is called internally.

**Interactive****property**

```
property Interactive : Boolean
```

- ↳ Allows for tighter automation control of the Excel server.

If Interactive is set to False, the end user cannot manipulate Excel, even if it is visible. If Interactive is True, the user is able to change the Excel spreadsheet directly. These changes are reflected in the corresponding OfficePartner Excel classes.

See also: [PropDirection](#), [Visible](#)

---

**LargeButtons** property

```
property LargeButtons : WordBool
```

- ↳ Specifies whether large or small buttons are displayed in the Excel MDI toolbar.

---

**MachineName** property

```
property MachineName
```

- ↳ Defines the name of a remote PC when using Distributed COM

Excel can be launched on another computer provided DCOM is configured correctly.

See also: PropDirection

---

**OnNewWorkbook** event

```
property OnNewWorkbook : TOnNewWorkbook
```

```
TOnNewWorkBook = procedure (
  Sender : TObject; const WorkBook : _Workbook) of object;
```

- ↳ Defines an event handler that is called when a new workbook is created.

Sender is the TOpExcel component that initiated the event. WorkBook contains the Workbook interface of the newly created Workbook.

See also: OnSheetActivate, OnSheetChange, OnWindowActivateOnWorkbookActivate

---

**OnSheetActivate** event

```
property OnSheetActivate : TOnSheetActivate
```

```
TOnSheetActivate = procedure (
  Sender : TObject; const Sh : IDispatch) of object;
```

- ↳ Defines an event handler that is called when a workbook receives focus.

Sender is the TOpExcel component that initiated the event. Sh contains the IDispatch interface to the worksheet receiving focus.

See also: OnSheetChange, OnSheetDeactivate, OnWorkbookNewSheet

## OnSheetCalculate

event

```
property OnSheetCalculate : TOnSheetCalculate  
TOnSheetCalculate = procedure (  
  Sender : TObject; const Sh : IDispatch) of object;
```

3

Defines an event handler that is called when Excel calculates a value for a cell.

Sender is the TOpExcel component that initiated the event. Sh contains the IDispatch interface for the worksheet on which the calculation is being performed.

See also: OnSheetChange

## OnSheetChange

event

```
property OnSheetChange : TOnSheetChange  
TOnSheetChange = procedure (  
  Sender : TObject; const Sh : IDispatch;  
  const Target : ExcelRange) of object;
```

Defines an event handler that is triggered immediately following a change in the contents of a cell.

Sender is the TOpExcel component that initiated the event. Sh is the IDispatch interface to the Excel worksheet whose cell contents have changed. Target is a dispinterface to a TOpExcelRange object. The Address property of Target contains the cell reference of the cells whose contents have changed in A1 format. To abort processing of this event, set Cancel to False.

Note: This event is not fired when a different or new worksheet is activated.

See also: OnSheetCalculate, OnWorkbookActivate

## OnSheetDeactivate

event

```
property OnSheetDeactivate : TOnSheetDeactivate  
TOnSheetDeactivate = procedure (  
  Sender : TObject; const Sh : IDispatch) of object;
```

Defines an event handler that is called when a workbook loses focus.

Sender is the TOpExcel component that initiated the event. Sh contains the IDispatch interface to the worksheet losing focus.

See also: OnSheetActivate, OnWorkbookDeactivate

**OnSheetSelectionChange****event**

```
property OnSheetSelectionChange : TOnSheetSelectionChange
TOnSheetSelectionChange = procedure (
  Sender : TObject; const Sh : IDispatch;
  const Target : ExcelRange) of object;
```

↳ Defines an event handler that is called when Excel's selected cells change.

Sender is the TOpExcel component that initiated the event. Sh contains the IDispatch interface to the worksheet whose selection has changed. Target is a dispinterface to a TOpExcelRange object. The Address property of Target contains the cell reference of the new selection in A1 format.

This event is not to be confused with OnSheetChange which is called when the contents of a cell changes. All that is required for this event to fire is for the user to select a different cell or range of cells.

See also: OnSheetCalculate, OnSheetChange

**OnWindowActivate****event**

```
property OnWindowActivate : TOnWindowActivate
TOnWindowActivate = procedure (
  Sender : TObject; const Wb : Workbook;
  const Wn : ExcelWindow) of object;
```

↳ Defines an event handler that is called when a workbook is activated.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the workbook being activated. Wn contains the dipsinterface to the Excel window that owns the Wb parameter.

See also: OnSheetActivate, OnWorkbookActivate

**OnWindowDeactivate****event**

```
property OnWindowDeactivate : TOnWindowDeactivate
TOnWindowDeactivate = procedure (
  Sender : TObject; const Wb : Workbook;
  const Wn : ExcelWindow) of object;
```

↳ Defines an event handler that is called when a workbook is deactivated.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the workbook losing focus. Wn contains the dipsinterface to the Excel window that owns the Wb parameter.

See also: OnWindowActivate

## OnWindowResize

event

property OnWindowResize : TOnWindowResize  
TOnWindowResize = procedure (  
  Sender : TObject; const Wb : Workbook;  
  const Wn : ExcelWindow) of object;

3

Defines an event handler that is called when the Excel window is resized.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to active workbook at the time of resizing. Wn contains the dipsinterface to the Excel window that owns the Wb parameter.

See also: ServerHeight, ServerLeft, ServerTop, ServerWidth

## OnWorkbookActivate

event

property OnWorkbookActivate : TOnWorkbookActivate  
TOnWorkbookActivate = procedure (  
  Sender : TObject; const Wb : Workbook) of object;

Defines an event handler that is called when a workbook is first created or focus is transferred from one workbook to another. Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the workbook losing focus.

See also: OnSheetActivate, OnWorkbookDeactivate, OnWorkbookOpen

## OnWorkbookAddinInstall

event

property OnWorkbookAddinInstall : TOnWorkbookAddinInstall  
TOnWorkbookAddinInstall = procedure (  
  Sender : TObject; const Wb : Workbook) of object;

Defines an event handler that is called when the user selects Tools | Add-Ins from Excel's main menu and selects an add-in to install.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the active workbook at the time the add in was installed.

See also: OnWorkbookAddinUninstall

## OnWorkbookAddinUninstall

event

```
property OnWorkbookAddinUninstall :  
  TOnWorkbookAddinUninstall  
  
TOnWorkbookAddinUninstall = procedure (  
  Sender : TObject; const Wb : Workbook) of object;
```

Defines an event handler that is called when the user selects Tools | Add-Ins from Excel's main menu and de-selects an add-in that is already installed.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the active workbook at the time the add in was removed.

See also: OnWorkbookAddinInstall

## OnWorkbookDeactivate

event

```
property OnWorkbookDeactivate : TOnWorkbookDeactivate  
  
TOnWorkbookDeactivate = procedure (  
  Sender : TObject; const Wb : Workbook) of object;
```

Defines an event handler that is called when a workbook is closed or focus is transferred away from the workbook specified in Wb.

Sender is the TOpExcel component that initiated the event. Wb contains the IDispatch interface to the workbook that is losing focus.

See also: BeforeWorkbookClose, OnSheetDeactivate, OnWorkbookActivate

## OnWorkbookNewSheet

event

```
property OnWorkbookNewSheet : TOnWorkbookNewSheet  
  
TOnWorkbookNewSheet = procedure (  
  Sender : TObject; const Wb : Workbook;  
  const Sh : IDispatch) of object;
```

Defines an event handler that is called when a worksheet is added to the workbook defined by Wb.

Sender is the TOpExcel component that initiated the event. Wb contains the interface to the workbook that owns the new worksheet. Sh contains the IDispatch interface of the new Excel worksheet.

See also: OnNewWorkbook, OnSheetActivate, OnWorkbookActivate, OnWorkbookOpen

## OnWorkbookOpen

event

```
property OnWorkbookOpen : TOnWorkbookOpen  
TOnWorkbookOpen = procedure (  
    Sender : TObject; const Wb : Workbook) of object;
```

↳ Defines an event handler that is called when a workbook is opened from disk.

Sender is the TOpExcel component that initiated the event. Wb contains the interface to the workbook that was just opened.

See also: BeforeWorkbookClose, OnSheetActivate

## PropDirection

property

6

```
property PropDirection
```

↳ Controls what happens to the component properties when the component is initially connected.

If PropDirection is pdToServer, the streamed design-time properties are pushed to Excel immediately after Excel is launched. If PropDirection is set to pdFromServer, the streamed properties are ignored and the component properties represent the state of Excel when it is launched.

See also: Connected

## RangeByName

property

```
property RangeByName[Name : string] : TOpExcelRange
```

↳ A shortcut that locates and returns any named range in all Workbooks represented by the component.

In the case of range name clashes, this property returns the first range found. It is the responsibility of the developer to ensure that unique range names are used.

See also: PropDirection

**Server****read-only property**


---

```
property Server : _Application
```

>Returns an interface to the Excel Application Object.

The majority of the functionality of this interface has been exposed in the form of properties and methods within the OfficePartner framework. It is recommended that you use the OfficePartner properties and methods whenever possible. This interface should be utilized only after consulting the source code and any additional information from available Microsoft resources, such as MSDN.

**ServerHeight****property**


---

```
property ServerHeight : Integer
```

>Returns the height, in pixels, of the Excel MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerLeft, ServerTop, ServerWidth

**ServerLeft****property**


---

```
property ServerLeft : Integer
```

>Returns the left position, in pixels, of the Excel MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerTop, ServerWidth

**ServerTop****property**


---

```
property ServerTop : Integer
```

Return the top position, in pixels, of the Excel MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerWidth

**ServerWidth****property**


---

```
property ServerWidth : Integer
```

>Returns the width, in pixels, of the Excel MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerTop

---

**UserName** property

---

```
property UserName : WideString
```

- ↳ Returns the name of the document author from Excel.

UserName is read/write and updates Excel accordingly. However, it can only be set when at least one Workbook is open.

---

**Visible** property

---

```
property Visible : Boolean
```

- ↳ Determines whether Excel is visible on the user's machine.

If Connected is True, this property specifies the visibility of the Excel automation server. If Connected is False, Visible specifies the desired visibility of the server when it is eventually connected.

See also: Connected, PropDirection

---

**WindowState** property

---

```
property WindowState : TOpXLWindowState
```

- ↳ Specifies the state of the Excel MDI window.

WindowState can be assigned a TOpXLWindowState enumeration value to control the state of the Excel MDI window. The window size is adjusted immediately after the value of this property is changed.

The following are valid values for TOpXLWindowState:

**Table 16:**

<b>Value</b>	<b>Result</b>
XlwsNormal	No change is made to the window size.
XlwsMinimized	The window is minimized.
XlwsMaximized	The window is maximized.

See also: OnWindowActivate, OnWindowDeactivate, OnWindowResize

---

**Workbooks** property

---

```
property Workbooks : TOpExcelWorkbooks
```

- ↳ Contains the collection of Workbooks open in Excel.

Workbooks is an indexed collection. Therefore, individual workbooks must be accessed using indexed parameters. The following code demonstrates how to access a single Workbook within this collection:

```
Excel1.Workbooks[0].SaveAs('Default.xls');
```

See also: BeforeWorkbookClose, BeforeWorkbookPrint, OnWorkbookActivate,  
OnWorkbookDeactivate, OnWorkbookNewSheet, OnWorkbookOpen,

# TOpExcelWorkbooks Class

The TOpExcelWorkbooks Class is a descendant of TOpNestedCollection. All of the items contained in this collection are TOpExcelWorkbook objects. This class encapsulates and provides access to all of the Workbooks owned by the current instance of Excel. The Add method of this class is the most common way that you'll add individual workbooks to the collection.

## Hierarchy

TCollection (VCL)

- ① TOpNestedCollection (OpOfficeShared)
  - TOpExcelWorkbooks (OpExcelClient)

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ① ItemName | ① ParentItem     | ① RootComponent |
| Items      | ① RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ① FindItem    |
| ① Create | ① ForEachItem |

## Reference Section

### Add method

```
function Add : TOpExcelWorkbook;
```

↳ Adds a new workbook to Excel.

The return value of this function is a pointer to the newly created Workbook. This is the preferred method of adding a workbook to the collection.

See also: Items

Items	property
<pre>property Items[index : Integer] : TOpExcelWorkbook</pre>	<p>Array property containing individual Workbook (TOpExcelWorkbook) items.</p> <p>Use the Items property to access individual workbooks. Items is the default array property for TOpExcelWorkbooks. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:</p> <pre>Excel1.Workbooks[0].SaveAs('Default.Txt'); Excel1.Workbooks.Items[0].SaveAs('Default.txt');</pre> <p>Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.</p> <p>See also: Add</p>

# TOpExcelWorkbook Class

The TOpExcelWorkbook class represents an Excel Workbook. Each workbook has a parent collection (TOpExcelWorkbooks) to which the workbook belongs.

The TOpExcelWorkbook class allows you to save, name, and activate a workbook. Each workbook maintains a collection of worksheets, representative of the manner in which Excel handles workbooks and worksheets.

## Hierarchy

TCollectionItem (VCL)

- ① TOpNestedCollectionItem (OpOfficeShared)438
  - TOpExcelWorkbook (OpExcelClient)

## Properties

AsWorkbook	PropDirection	① Verb
Filename	① RootCollection	① VerbCount
① Intf	① RootComponent	Worksheets
① ParentCollection	① SubCollection	
① ParentItem	① SubCollectionCount	

## Methods

Activate	Destroy	SaveAs
Connect	ExecuteVerb	
Create	Save	

---

## Reference Section

---

### Activate

method

```
procedure Activate;
```

- ↳ Activates the workbook if Excel is visible and connected.

If Excel is not visible or is not connected, then calling this method has no effect. If Excel is connected and visible, then the Excel window receives focus, and the workbook object on which this method was invoked is brought to the front. The following code demonstrates how to activate the second workbook in the collection:

```
Excel1.Workbooks[1].Activate;
```

See also: PropDirection

---

### AsWorkbook

read-only property

```
property AsWorkbook : _Workbook
```

- ↳ Contains the automation interface for the Workbook.

Most of the functionality of a workbook has been exposed through the interface of the TOpWorkbook class. However, it may sometimes be necessary to expand on what OfficePartner has encapsulated. This interface allows you to do so. It provides access to virtually all of the workbook functionality that Microsoft has elected to expose. It would not be practical to document the entire functionality of this interface and it is suggested that you consult the source code and any additional information published by Microsoft (MSDN for example).

---

### Connect

method

```
procedure Connect;
```

- ↳ Connects the workbook to the TOpExcel component.

If the TOpExcel component is not connected, calling this method has no effect.

See also: Activate, PropDirection

---

### Create

constructor

```
constructor Create(Collection : TCollection);
```

- ↳ Creates an instance of a new Excel workbook.

Collection defines the TCollection object that the new workbook is added to.

A default worksheet is automatically added to the workbook for you.

It is recommended that you create a workbook by calling the Add method of the TOpExcelWorkbooks class. This method automatically creates and returns a workbook and adds it to the collection.

See also: Destroy

## Destroy

## **destructor**

---

**destructor** **Destroy**;

↳ Destroys the workbook and all worksheets contained in the workbook.

Excel handles the destruction of workbooks when the Excel server is closed. It is not recommended that you call this method directly.

See also: Create

## ExecuteVerb

## **method**

---

**procedure** **ExecuteVerb(index : Integer)**;

↳ Invokes the Save method if, and only if, index = 0.

It is recommended that you call the Save method directly instead of using this method. Any parameter other than 0 (zero) passed to this method has no effect.

See also: Save

## Filename

## **property**

---

**property** **Filename : TFilename**

↳ Loads an existing Excel workbook or template defined by Filename.

This property is intended to open the Excel filename defined by this property. Do not set this property to a name of a file that doesn't exist on disk. When setting the Filename property directly, you must ensure that the corresponding \*.xls or \*.xla file exists on disk. If Excel cannot find the file specified by Filename, an exception is raised.

The Fileame property can be assigned at design time using the nested collection editor. To access the nested collection editor, click the ellipsis button adjacent to the Workbooks property.

The Filename can also be set at run time using code similar to the following:

```
ExistingWorkBook.Filename := 'c:\SaveTest.xls';
```

---

If **Filename** is assigned at design time, workbook members do not persist automatically and are populated with entities from the file or template.

See also: [Save](#), [SaveAs](#)

## **Print**

## **method**

```
procedure Print;
```

↳ Prints all of the worksheets in the workbook.

See also: [Worksheets](#), [TOpExcelWorksheet.Print](#)

## **PropDirection**

## **property**

```
property PropDirection : TPropDirection
```

↳ Specifies whether the workbook is initialized by the component, or loaded from an existing file.

If **PropDirection** is set to **pdToServer** then the Excel workbook is initialized by the properties of this object. If **PropDirection** is set to **pdFromServer**, then the properties and methods of this object are initialized according to the properties of the workbook defined by **Filename**.

See also: [Activate](#)

## **Save**

## **method**

```
procedure Save;
```

↳ Saves the workbook using the value of the current **Filename** property.

The following code saves the first open workbook under its current file name:

```
Excel1.Workbooks[0].Save;
```

If you want to save the workbook under a different file name then use the **SaveAs** method and provide the filename as a parameter.

See also: [SaveAs](#), [Filename](#)

## **SaveAs**

## **method**

```
procedure SaveAs(FileName : TFileName);
```

↳ Saves the workbook using the **FileName** parameter as the name of the file.

**FileName** defines the disk file name of the Excel workbook.

The following example saves the first open workbook as **SaveTest.xls** on the user's C: drive.

```
xl.Workbooks[0].SaveAs('c:\SaveTest.xls');
```

See also: Save, Filename

## Worksheets

property

3

```
property Worksheets : TOpExcelWorksheets
```

Collection containing all worksheets in the workbook.

Worksheets can be added to the workbook by calling the Add method of the TOpExcelWorksheets class. Excel chart and macro sheets are not currently supported, however embedded charts are supported.

The following code demonstrates how to add a worksheet to the first open workbook:

```
Excel1.Workbooks[0].Worksheets.Add;
```

Individual worksheets within the collection can be indexed within this collection using code similar to the following:

```
Excel1.Workbooks[0].Worksheets[1].Name := 'Financials';
```

See also: PropDirection, Activate

# TOpExcelWorksheets Class

The TOpExcelWorksheets object is a collection of worksheets. Every TOpExcelWorksheets object must belong to a TOpExcelWorkbook object (the parent). Individual worksheets within a TOpExcelWorkbook, are accessed by indexing into the Worksheets collection.

Individual worksheets are added to the collection by calling the Add method of this class. The Add method returns a TOpExcelWorksheet. You can set the properties of this TOpExcelWorksheet as required.

## Hierarchy

TCollection (VCL)

① TOpNestedCollection (OpOfficeShared)434

TOpExcelWorksheets (OpExcelClient)

## Properties

① ItemName

Items

① ParentItem

RootCollection

① RootComponent

## Methods

Add

① Create

① FindItem

ForEachItem

## Reference Section

### Add

### method

---

```
function Add : TOpExcelWorksheet;
```

↳ Adds a new worksheet to the parent workbook.

This is the method you should use to add a worksheet to the collection. This function returns a TOpExcelWorksheet, whose properties you can manipulate as required.

See also: Items

Items	property
property Items[ Index : Integer ]: TOpExcelWorksheet	

⌚ Array property containing individual worksheet items.

Index is the zero based index of the worksheet.

Use the Items property to access individual worksheets. Items is the default array property for TOpExcelWorksheets. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
Excel1.Workbooks[ 0 ].Worksheets.Items[ 1 ].DoSomething;  
Excel1.Workbooks[ 0 ].Worksheets[ 1 ].DoSomething;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add

# TOpExcelWorksheet Class

The TOpExcelWorksheet class encapsulates the functionality of an Excel worksheet. All Excel worksheets are associated with an Excel workbook and an Excel workbook must contain one or more worksheets. Additionally, Excel worksheets may contain a collection of ranges, charts, and/or hyperlinks. Ranges define one or more contiguous cells within the worksheet. Charts define one of various Excel chart types, and hyperlinks define either a URL or an Excel hyperlink. A URL hyperlink activates the default web browser. An Excel hyperlink simply navigates to a cell (described in Excel A1 format) within the Excel workbook.

A default worksheet is automatically created when a workbook is added to a TOpExcel component. Additional worksheets may be added by calling the TOpWorksheets.Add method associated with the TOpExcel component.

Additionally, all Excel worksheets have a name property that appears on the tab at the bottom of the worksheet.

## Hierarchy

TCollectionItem (VCL)

① TOpNestedCollectionItem (OpOfficeShared)438

TOpExcelWorksheet (OpExcelClient)

## Properties

AsWorksheet	① ItemName	Ranges
Charts	Name	① RootCollection
Hyperlinks	① ParentItem	① RootComponent

## Methods

Activate	Destroy	Print
Connect	① FindItem	
① Create	① ForEachItem	

# Reference Section

3

## Activate method

---

procedure Activate;

↳ Selects the worksheet if Excel is visible and connected.

If Excel is not visible or not connected, calling this method has no effect. Otherwise, the worksheet on which this method was invoked receives focus and is brought to the forefront.

See also: Connect

## AsWorksheet read-only property

---

property AsWorksheet : \_Worksheet

↳ Allows access to the underlying automation interface for the worksheet.

The functionality of this interface is largely encapsulated within the OfficePartner framework. Use this interface property to access functionality beyond that which is encapsulated by OfficePartner.

It is not feasible to document all methods and properties of this interface. You should consult the source code and any additional information (such as MSDN) prior to invoking methods of this interface.

## Charts property

---

property Charts : TOpExcelCharts

↳ A collection representing all the Excel chart objects in the Worksheet.

Charts may be added to the worksheet by calling the TOpExcelCharts.Add method. Individual chart properties may then be accessed by iterating through the TOpExcelCharts.Items property. The following code creates an area chart on the first worksheet of the workbook:

```
var
  Xlc : TOpExcelChart;
begin
  Xlc := WorkSheets[0].Charts.Add;
  Xlc.ChartType := xlctArea;
  //Set other chart properties as needed
end;
```

See also: Hyperlinks, Ranges

---

<b>Connect</b>	<b>method</b>
----------------	---------------

---

```
procedure Connect;
```

- ↳ Activates the Excel workbook that is the parent of this worksheet.

If Excel is not visible or not connected, calling this method has no effect. Otherwise, the worksheet on which this method was invoked receives focus and is brought to the forefront.

See also: Activate

---

<b>Destroy</b>	<b>destructor</b>
----------------	-------------------

---

```
destructor Destroy;
```

- ↳ Destroys the worksheet collection item and deletes the worksheet from Excel if Excel is connected and it is not the last sheet in the workbook.

A workbook cannot exist without at least one worksheet open. Any attempt to destroy the last sheet in the workbook will fail.

Excel automatically handles the destruction of worksheets when the Excel server is disconnected. You should not call this method directly.

---

<b>Hyperlinks</b>	<b>property</b>
-------------------	-----------------

---

```
property Hyperlinks : TOpExcelHyperlinks
```

- ↳ A collection representing all the hyperlinks in the worksheet.

Hyperlinks can be added to a worksheet by invoking the TOpExcelHyperlinks.Add method. The following code demonstrates how to add a hyperlink that points to [www.turbopower.com](http://www.turbopower.com) in cell A1 of the first worksheet:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  hl : TOpExcelHyperlink;
begin
  hl := xl.workbooks[0].worksheets[0].Hyperlinks.Add;
  HL.Address := 'http://www.turbopower.com';
  HL.AnchorCell := 'A1';
  HL.Visible := True;
end;
```

See also: Charts, Ranges

**Name****property**

```
property Name : WideString
```

↳ The string that appears on the worksheet tab in the Excel Workbook.

The following code sets the worksheet name to MyWorkSheet:

```
Excel1.Workbooks[0].Worksheets[0].Name := 'MyWorkSheet';
```

**Print****method**

```
procedure Print;
```

↳ Prints the active worksheet.

See also: Activate, TOpExcelWorkbook.Print

**Ranges****property**

```
property Ranges : TOpExcelRanges
```

↳ A collection representing all the mapped ranges in the worksheet.

Ranges can be added to individual worksheets by invoking the TOpExcelRanges.Add method of the corresponding worksheet. The following code demonstrates how to add a range to the first worksheet:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  r : TOpExcelRange;
begin
  r := xl.Workbooks[0].Worksheets[0].Ranges.Add;
  r.Address := 'B2';
  r.SimpleText := 'Hello';
end;
```

See also: Charts, Hyperlinks

---

## TOpExcelRanges Class

Every TOpExcelWorksheet object contains a single TOpExcelRanges object. This object is a collection of zero or more TOpExcelRange objects. Ranges are the most fundamental objects of Excel. Individual ranges are added to the collection by calling the Add method of this class. The Add method returns a TOpExcelRange. You can set the properties of this TOpExcelRange object as required.

For additional information on working with the Range object, refer to the TOpExcelRange class on page 93.

### Hierarchy

TCollection (VCL)

① TOpNestedCollection (OpOfficeShared)434

TOpExcelRanges (OpExcelClient)

### Properties

① ItemName  
Items

① ParentItem  
① RootCollection

① RootComponent

### Methods

Add  
① Create

① FindItem  
① ForEachItem

## Reference Section

### Add method

---

```
function Add : TOpExcelRange;
```

3

↳ Adds a new range to the parent worksheet.

This is the method you should use to add a range to a worksheet. The Add method returns a TOpExcelRange object. You can set the properties of this TOpExcelRange object as required. The following code demonstrates how to add a range and set its text to 'Hello':

```
procedure TForm1.Button1Click(Sender : TObject);
var
  r : TOpExcelRange;
begin
  r := xl.Workbooks[0].Worksheets[0].Ranges.Add;
  r.Address := 'B2';
  r.SimpleText := 'Hello';
end;
```

See also: Items

### Items property

---

```
property Items[Index : Integer] : TOpExcelRange
```

↳ Array property containing individual Range items.

Index is a zero based index of a TOpExcelRange object.

Use the Items property to access individual ranges. Items is the default array property for TOpExcelRanges. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
Excel1.Workbooks[0].Worksheets[0].Ranges.Items[0].
  SimpleText := 'Hello';

Excel1.Workbooks[0].Worksheets[0].Ranges[0].
  SimpleText := 'Hello';
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add

# TOpExcelRange Class

A TTopExcelRange object represents a single cell or group of contiguous cells in the parent worksheet. The range is the most fundamental object of Excel, and it is the one with which you will most often interact. Any Excel cell reference that can be described in Excel's A1 format can be referenced by objects of the TTopExcelRange class.

The TTopExcelRange class has a Name property which allows you to define a cell or range of cells in A1 format, assign a name to that range, and then programmatically reference the range using the RangeByName method.

Since there are an infinite number of Range combinations in any given worksheet, ranges can be mapped and persisted at design-time in order to be easily manipulated at run-time.

## Hierarchy

TCollectionItem (VCL)

① TTopNestedCollectionItem (OpOfficeShared)438

TOpExcelRange (OpExcelClient)

## Properties

Address	FontSize	① RootComponent
AnchorCell	HorizontalAlignment	RotateDegrees
AsRange	IndentLevel	RowHeight
BorderLineWeight	① Intf	ShrinkToFit
Borders	IsEmpty	SimpleText
BorderStyle	Name	① SubCollection
ClearOnMove	OfficeModel	① SubCollectionCount
Color	Orientation	Value
ColumnWidth	① ParentItem	① Verb
FontAttributes	Pattern	① VerbCount
FontColor	PatternColor	VerticalAlignment
FontName	① RootCollection	WrapText

## Methods

Activate	AutoFitRows	ClearRange
AutoFitColumns	Clear	Connect

- ① Create
- ① ExecuteVerb

- Populate
- PopulateCurrent

- Select
- SetAddressFromSelection

## Reference Section

3

### Activate method

```
procedure Activate;
```

↳ Activates the worksheet containing the range.

If the Excel server is not connected or is not visible, calling this method has no effect. Otherwise, the worksheet to which this range belongs receives focus and becomes the active worksheet.

### Address property

```
property Address : string
```

↳ Represents the address of the range in Excel A1 format.

Excel defaults to using the A1 reference style, which refers to columns by letters and rows by numbers. An Excel spreadsheet contains 256 columns with column headers of A through IV. The first 26 columns simply have column headers of A-Z. For columns 27 through 256, two characters are required to define a distinct column header as follows:

**Table 17:**

Column #	Column header
27	AA
28	AB
29	AC
...	...
52	BA
53	BB
54	BC
...	...
256	IV

Row headers are defined simply by their numeric position. An Excel spreadsheet contains 65,536 rows with corresponding row headers of 1 through 65,536.

A single cell is referenced by concatenating the column and row headers. For example AC17 refers to the cell defined by the intersection of the 29<sup>th</sup> column and the 17<sup>th</sup> row.

A range of contiguous cells can be defined by referencing the cell in the upper-left corner and the cell in the lower-right corner. These cell references must be separated by a colon (:). For example, an address of B2:D2 refers to the three cells B2, C2, and D2.

For further information on Excel's A1 format, please refer to Excel's online help.

See also: AnchorCell, SimpleText, Value

## **AnchorCell**

**property**

**property** AnchorCell : string

↳ Represents the top left cell of the range in A1 format.

The AnchorCell property defines the first cell filled by the Populate and PopulateCurrent methods.

See also: Address, Populate, PopulateCurrent

## **AsRange**

**read-only property**

**property** AsRange : Range

↳ Allows access to the underlying automation interface for the range.

The functionality of this interface is largely encapsulated within the OfficePartner framework. AsRange allow you to extend the functionality of the Range object beyond that which OfficePartner has encapsulated. It is impractical to document all of the functionality of this interface. We suggest that you consult the source code and any additional information from Microsoft (such as MSDN) prior to using this interface.

## **AutoFitColumns**

**method**

**procedure** AutoFitColumns;

↳ Automatically expands the column width to accomodate text.

The column width will only be increased if the text in the cells of the column is wider than the column width.

See also: AutoFitRows, ShrinkToFit, WrapText, ColumnWidth, RowHeight

**AutoFitRows****method**

```
procedure AutoFitRows;
```

- ↳ Automatically expands the row height to accomodate text.

The row height will only be increased if the text in the cells of the column is taller than the row height.

See also: AutoFitColumns, ShrinkToFit, WrapText, ColumnWidth, RowHeight

**BorderLineWeight****property**

```
property BorderLineWeight : TOpXlBorderWeights
  TOpXlBorderWeights = (
    xlbwHairline, xlbwThin, xlbwMedium, xlbwThick);
```

Default: xlbwHairline

- ↳ Determines the thickness of the border applied to the range.

This property setting is applied to all borders. The Borders collection determines which of the four borders are actually painted and visible around the range. Only borders specified in the Borders collection will be visible.

See also: Borders, BorderStyle

**Borders****property**

```
property Borders : TOpXlRangeBorders
  TOpXlBorders = (xlbLeft, xlbRight, xlbTop, xlbBottom);
  TOpXlRangeBorders = set of TOpXlBorders;
```

Default: empty set

- ↳ Defines a set whose elements correspond to the borders of the range.

Each element of this set corresponds to one of the borders of the cells defined in the range. For example, xlbLeft corresponds to the left border of the cell(s). The elements included in this set determine which borders of the cells are affected when setting either the BorderLineWeight property or the BorderStyle property. If an element is not included in this set, then setting either the BorderLineWeight property or the BorderStyle property will

produce no visual effect. If an element is present in this set, then setting either the BorderLineWeight property or the BorderStyle property will immediately update the corresponding border to reflect the new setting.

See also: BorderLineWeight, BorderStyle

---

**BorderStyle** property

```
property BorderStyle : TOpXlBorderLineStyles
TObjBorderLineStyles = (
  xlblsContinuous, xlblsDash, xlblsDashDot, xlblsDashDotDot,
  xlblsDot, xlblsDouble, xlblsLineStyleNone, xlblsSlantDashDot);
```

Default: xlblsLineStyleNone

- ↳ Determines the line style of the border(s) applied to the range.

This property setting is applied to all borders. The Borders collection determines which of the four borders will visually inherit the characteristics of this property setting. Only borders specified in the Borders collection will be visible. The BorderLineWeight property determines the thickness of the line. The results of this property setting are described in the following table:

**Table 18:**

Value	Description
xlblsContinuous	Solid, continuous line
xlblsDash	Dashed line
xlblsDashDot	Alternating dashes and dots
xlblsDashDotDot	Dash followed by two dots, repeating
xlblsDot	Dotted line
xlblsDouble	Two parallel lines
xlblsLineStyleNone	No line
xlblsSlantDashDot	Slanted sequence of alternating dashes and dots

See also: Borders, BorderLineWeight

---

**Clear** method

```
procedure Clear;
```

- ↳ Clears the text and formatting of the cell(s) included in the range.

Call this method to delete any text that appears in the cell and to remove all formatting from the cell. After calling this method, the cell reverts back to Excel's default formatting.

See also: BorderLineWeight, BorderStyle, Pattern, PatternColor, SimpleText

**ClearOnMove****property**

```
property ClearOnMove : Boolean
```

- ↳ Determines whether or not the contents of the range are cleared when the address is changed.

Set ClearOnMove to True if you want the contents of the range to be cleared when the address changes.

See also: Address, AnchorCell

**ClearRange****method**

```
procedure ClearRange;
```

- ↳ Clears the contents of the cells defined by the Address property.

The contents of all cells contained in the range are deleted after calling this method.

See also: Address, AnchorCell

**Color****property**

```
property Color : TColor
```

Default: clWindow

- ↳ Determines the background color of the cells within the range.

The background of all cells defined by the range will be painted in the color specified in this property.

See also: PatternColor

**ColumnWidth****property**

```
property ColumnWidth : Integer
```

Default: 20

- ↳ Determines the column width of the cells within the range.

This number represents the number of characters that can be displayed in a cell formatted with Excel's standard font. If the column width is 0, the column is hidden. Valid values for ColumnWidth are 0 (zero) through 255.

See also: RowHeight

---

**Connect** method

```
procedure Connect;
```

↳ Activates the workbook if Excel is visible and connected.

If Excel is not connected or is not visible, then calling this method has no effect.

---

**ExecuteVerb** method

```
procedure ExecuteVerb(index : Integer);
```

↳ Invokes one of the following methods based on the value of index:

**Table 19:**

Integer	Method invoked
0	Populate
1	Select
2	SetAddressFromSelection

All of these methods can be invoked directly using other methods of this class. You should use those methods in favor of the ExecuteVerbMethod.

Note: Passing an integer parameter other than 0, 1, or 2 has no effect.

See also: PopulateCurrent, Select, SetAddressFromSelection

---

**FontAttributes** property

```
property FontAttributes : TOpXlRangeFontAttributes
  TOpXlFontAttributes = (
    xlfaBold, xlfaItalic, xlfaUnderline, xlfaStrikethrough);
  TOpXlRangeFontAttributes = set of TOpXlFontAttributes;
```

Default: empty set

↳ Defines a set of font formatting attributes.

Each member element of this set corresponds to a font attribute that is applied to all text in all cells defined by the range (xlfaBold corresponds to bold font, for instance). Any combination of set elements can exist to achieve the desired result. For example, to format the text so that it appears bold and italic, simply add both the xlfaBold and xlfaItalic members to the set.

See also: FontColor, FontName, FontSize

**FontColor****property**

```
property FontColor : TColor
```

Default: clWindowText

↳ Determines the font color for all text included in the range.

All text in the cells defined by the range will immediately update when this property is changed.

See also: FontAttributes, FontName, FontSize

**FontName****property**

```
property FontName : string
```

↳ Determines the name of the font typeface.

Care must be taken to avoid setting the FontName property to the name of a typeface that is not present on the user's machine. It is recommended that you use one of Windows's standard fonts when changing this property, or let the user select the typeface from a standard font dialog.

See also: FontAttributes, FontColor, FontSize

**FontSize****property**

```
property FontSize : Integer
```

Default: 12

↳ Defines the point size of the font for all text included in the range.

Use this property to customize the font size of the text included in the range. Smaller numbers correspond to smaller font sizes while larger numbers correspond to larger font sizes. In practice, FontSize is normally a value between 5 and 72, but other sizes are applicable.

See also: FontAttributes, FontColor, FontName

**HorizontalAlignment****property**

```
property HorizontalAlignment : TOpXlCellHorizAlign
```

```
TOpXlCellHorizAlign = (
  xlchaGeneral, xlchaLeft, xlchaCenter, xlchaRight, xlchaFill,
  xlchaJustify, xlchaCenterAcrossSelection);
```

Default: xlchaLeft

↳ Determines the horizontal positioning of text within the range.

Possible values for this property and their results are described in the following table:

**Table 20:**

<b>Value</b>	<b>Description</b>
xlchaGeneral	Excel's default alignment
xlchaLeft	Text is left-aligned
xlchaCenter	Text is centered
xlchaRight	Text is right-aligned
xlchaFill	Text is repeated to fill the width of the cell
xlchaJustify	Text is justified
xlchaCenterAcrossSelection	Text is centered across one or more selected cells

See also: [VerticalAlignment](#), [IndentLevel](#)

### **IndentLevel**

### **property**

`property IndentLevel : Integer`

Default: 0

↳ Determines the distance that text is indented.

The value entered sets the number of characters by which text is indented to the right.

See also: [HorizontalAlignment](#), [VerticalAlignment](#), [Orientation](#), [RotateDegrees](#)

### **IsEmpty**

### **property**

`property IsEmpty : Boolean;`

↳ Indicates whether the range contains data.

`IsEmpty` returns True if and only if all of the cells defined by the range are empty. If any data exists in any of the cells defined by the range, `IsEmpty` returns False.

See also: [Address](#), [SimpleText](#), [Value](#), [ClearRange](#)

Name	property
------	----------

`property Name : string`

↳ Assigns a Name to the mapped range.

This property has no meaning in Excel itself, but allows developers to use the RangeByName property to easily find any mapped ranges in the TOpExcel component. It is the developer's responsibility to ensure that range names do not conflict.

See also: Address, AnchorCell

OfficeModel	property
-------------	----------

`property OfficeModel : TOpUnknownComponent`

↳ Defines the TOpDataSetModel component that implements the IOpModel interface.

The TOpDataSetModel is essentially the conduit between a range and a TDataSet descendant. This property is all that is required to establish this link. Simply drop a TOpDataSetModel onto the form and set the OfficeModel property of the range to the TOpDataSetModel. The TOpExcelRange class provides two methods to populate the range as follows:

**Table 21:**

Method	Result
Populate	Populates the range (starting in the AnchorCell) with all columns and rows in the dataset.
PopulateCurrent	Populates the range (starting in the AnchorCell) with all columns of the current record only.

The IOpModel interface implemented by the TOpDataSetModel also defines several functions used to navigate a dataset such as First, Next, Prior, etc.

See also: Address, AnchorCell, Populate, PopulateCurrent

Orientation	property
-------------	----------

`property Orientation : TOpXlCellOrientation`

```
TOpXlCellOrientation = (
    xlcoDownward, xlcoHorizontal, xlcoUpward,
    xlcoVertical, xlcoRotated);
```

Default: xlcoHorizontal

↳ Determines the orientation of text included in the range.

Use this property to set the direction and orientation that text is displayed. All property settings take effect immediately, except for xlcoRotated. If Orientation is set to xlcoRotated, you must supply the number of degrees to rotate the text in the RotateDegrees property. Possible values for this property and their results are described in the following table:

**Table 22:**

<b>Value</b>	<b>Description</b>
XlcoDownward	Text is rotated 90 degrees clockwise
XlcoHorizontal	Normal horizontal text
XlcoUpward	Text is rotated 90 degrees counter-clockwise
XlcoVertical	Individual characters are not rotated, but text is displayed vertically
XlcoRotated	Text is rotated to the number of degrees specified in the RotateDegrees property

See also: VerticalAlignment, HorizontalAlignment, IndentLevel, RotateDegrees

**Pattern****property**

```
property Pattern : TOpXlInteriorPatterns
TTopXlInteriorPatterns = (
  xlipPatternAutomatic, xlipPatternChecker, xlipPatternCrissCross,
  xlipPatternDown, xlipPatternGray16, xlipPatternGray25,
  xlipPatternGray50, xlipPatternGray75, xlipPatternGray8,
  xlipPatternGrid, xlipPatternHorizontal, xlipPatternLightDown,
  xlipPatternLightHorizontal, xlipPatternLightUp,
  xlipPatternLightVertical, xlipPatternNone, xlipPatternSemiGray75,
  xlipPatternSolid, xlipPatternUp, xlipPatternVertical);
```

Default: xlipPatternNone

 Determines the pattern displayed in the interior of the cells within the range.

The pattern selected for this property will be displayed in the color specified in the PatternColor property. The results of this property setting are described in the following table:

**Table 23:**

<b>Value</b>	<b>Description</b>
xlipPatternAutomatic	Excel's default pattern
xlipPatternChecker	Dark lines on both diagonals
xlipPatternCrissCross	Light lines on both diagonals
xlipPatternDown	Top, left to bottom, right diagonal lines
xlipPatternGray8	Very sparse dotted pattern
xlipPatternGray16	Sparse dotted pattern
xlipPatternGray25	Moderately heavy dotted pattern
xlipPatternGray50	Heavy dotted pattern
xlipPatternGray75	Heaviest dotted pattern
xlipPatternGrid	Vertical and horizontal lines
xlipPatternHorizontal	Horizontal lines
xlipPatternLightDown	Light top, left to bottom, right diagonal lines
xlipPatternLightHorizontal	Light horizontal lines
xlipPatternLightUp	Light top, right to bottom, left diagonal lines

**Table 23:**

<b>Value</b>	<b>Description</b>
xlipPatternLightVertical	Light vertical lines
xlipPatternNone	No pattern
xlipPatternSemiGray75	Dark interlaced appearance
xlipPatternSolid	Solid
xlipPatternUp	Dark top, right to bottom, left diagonal lines
xlipPatternVertical	Dark vertical lines

See also: [PatternColor](#), [BorderLineWeight](#), [BorderLineStyle](#), [Borders](#)

---

<b>PatternColor</b>	<b>property</b>
---------------------	-----------------

---

**property PatternColor : TColor**

Default: clWhite

↳ Determines the color of the cell's interior pattern.

This property determines the color of the pattern specified in the [Pattern](#) property. Changing this property value immediately updates the interior of the cell(s) included in the range.

See also: [Pattern](#), [BorderLineWeight](#), [BorderLineStyle](#), [Borders](#)

---

<b>Populate</b>	<b>method</b>
-----------------	---------------

---

**procedure Populate;**

↳ Fills a range of cells with data from a [TOpDataSetModel](#).

The [TOpDataSetModel](#) provides the link between a Range object and a descendant of [TDataSet](#). Setting the [OfficeModel](#) property of the range to an existing [TOpDataSetModel](#) is all that is required to make this link.

When the [Populate](#) method is called, the model is iterated and the range is filled with all rows and columns. Although the [Populate](#) method can be called at design time (from the [NestedCollection Editor](#)), populated values do not persist. The [Populate](#) method should be called at run-time to fill the range from the associated model. This method supplies column headers if they are supported by the model. If the model has a [DetailModel](#) (for [TOpDataSetModels](#)) then the Range is generated in outline form, showing the master-detail

---

relationship. This method fills data starting at the AnchorCell and proceeding down for rows, and across for columns. After calling the Populate method, the TOpExcelRange is updated to reflect all of the cells filled by the Populate method.

See also: PopulateCurrent, TOpDataSetModel

---

## **PopulateCurrent** method

```
procedure PopulateCurrent;
```

« Populates a range with data from a single record associated with a TOpDataSetModel.

The TOpDataSetModel provides the link between a Range object and a descendant of TDataSet. Setting the OfficeModel property of the Range to an existing TOpDataSetModel is all that is required to make this link.

PopulateCurrent does not iterate the model but only fills a range with the single, currently positioned row in the model. The PopulateCurrent method does not retrieve column headings from the model and is therefore useful for more customized population of worksheets. After calling PopulateCurrent, the TOpExcelRange is updated to reflect all of the cells filled by the PopulateCurrent method.

See also: Populate

---

## **RotateDegrees** property

```
property RotateDegrees : Integer
```

Default: 0

« Determines the number of degrees that text is rotated.

This property setting only applies if the Orientation property is set to xlcoRotated. This property value is in degrees and must be between -90 and 90.

A value of 0 corresponds to horizontal text. The maximum value of 90 rotates the text 90 degrees counter-clockwise from horizontal and the minimum value of -90 rotates the text 90 degrees clockwise from horizontal.

See also: Orientation

---

## **RowHeight** property

```
property RowHeight : Integer
```

Default: 12

« Determines the height of the rows in cells included in the range.

This value relates to font point size. If the value of the FontSize property is larger than this property setting, then the resulting text will be clipped. Generally, you will want to specify a value for this property setting that is equal to or greater than the value specified in the FontSize property.

See also: [FontSize](#), [ColumnWidth](#)

## **Select**

**method**

---

```
procedure Select;
```

↳ Highlights the Excel cells defined by the Address property.

If Excel is not visible, or is not connected, a call to this procedure has no effect.

See also: [Address](#), [AnchorCell](#)

## **SetAddressFromSelection**

**method**

---

```
procedure SetAddressFromSelection;
```

↳ Sets the Address property of the range to represent the current selection in the Excel workbook.

This method provides a convenient way to set the address of the range based on the user's input.

See also: [Address](#), [AnchorCell](#)

## **SimpleText**

**property**

---

```
property SimpleText : string
```

↳ Places a simple string value into the range.

Defines the text value for each cell in the defined range.

See also: [Address](#), [Value](#)

## **ShrinkToFit**

**property**

---

```
property ShrinkToFit : Boolean
```

Default: False

↳ Determines whether the font size is automatically reduced to fit text in a cell.

If the width of text in a cell exceeds the ColumnWidth property setting and ShrinkToFit is set to True, the FontSize property will be automatically reduced as necessary to fit the text in the cell. If the width of the text in a cell is less than the ColumnWidth property setting or ShrinkToFit is set to False, then no adjustments will be made to the FontSize property.

See also: [FontSize](#), [ColumnWidth](#)

---

<b>Value</b>	<b>property</b>
--------------	-----------------

property Value : Variant

↳ Represents the value(s) in the represented range.

If the Range represents multiple cells, Value is a variant array. Multiple cell values can also be set by setting this property equal to a Variant array.

6

See also: [Address](#), [SimpleText](#)

---

<b>VerticalAlignment</b>	<b>property</b>
--------------------------	-----------------

property VerticalAlignment : TOpXlCellVertAlign

TOpXlCellVertAlign = (xlcvatop, xlcvacenter, xlcvabottom,  
xlcvajustify, xlcvadistributed);

Default: xlcvabottom

↳ Determines the vertical alignment of text within the cell(s) included in the range.

Possible values for this property and their result are described in the following table:

**Table 24:**

<b>Value</b>	<b>Description</b>
Top	Text is aligned to the top of the cell
Center	Text is vertically centered in the cell
Bottom	Text is aligned to the bottom of the cell
Justify	Text is vertically justified
Distributed	Text spacing is altered to span the row height of the cell

See also: [HorizontalAlignment](#), [Orientation](#), [RowHeight](#)

```
property WrapText : Boolean;
```

Default: False

↳ Determines whether text is wrapped within the cell(s) of the range.

If WrapText is True and text within a cell is wider than the column width of the cell, then the text will wrap automatically. If WrapText is False or text within a cell is narrower than the width of the cell, then no change will take place.

See also: ShrinkToFit, AutoFitColumns, AutoFitRows

# TOpExcelCharts Class

Every TOpExcelWorksheet object contains a TOpExcelCharts object. This EOpExcelCharts object is a collection of zero or more individual TOpExcelChart objects. OfficePartner currently supports only embedded charts. Full-sheet charts (\*.xlc) files are not supported.

Charts are added to an existing worksheet by calling the Add method. The Add method returns a TOpExcelChart object. You can set the properties of this TOpExcelChart object as required.

## Hierarchy

TCollection (VCL)

① TOpNestedCollection (OpOfficeShared) 434

    TOpExcelCharts (OpExcelClient)

## Properties

① ItemName  
    Items

① ParentItem  
    ① RootCollection

① RootComponent

## Methods

    Add  
① Create  
        ① FindItem  
        ① ForEachItem

## Reference Section

### Add

### method

function Add : TOpExcelChart;

↳ Adds a new chart to the parent worksheet.

The return value of this function is a TOpExcelChart. You can set the properties of this TOpExcelChart as desired.

The following code adds a chart to the first worksheet and sets basic chart properties:

```
{Add and position a chart to the worksheet}
Chart := OpExcel.Workbooks[0].Worksheets[0].Charts.Add;
Chart.Left := 350;
Chart.Width := 550;
Chart.Height := 400;

{We'll create a pie chart..}
Chart.ChartType := xlct3DPie;
{Tell the chart where to find the data}
Chart.DataRange := 'A1:C100';
```

See also: TOpExcelChart

Items	property
-------	----------

property Items[Index : Integer] : TOpExcelChart

↳ Contains individual chart object items.

Index is a zero based index of a TOpExcelChart object.

Use the Items property to access individual charts. Items is the default array property for TOpExcelCharts. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
Excel1.Workbooks[0].WorkSheets[0].
    Charts.Items[0].DoSomething;

Excel1.Workbooks[0].WorkSheets[0].Charts[0].DoSomething;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add

# TOpExcelChart Class

A TOpExcelChart item represents a ChartObject in the parent worksheet. There are two types of charts in Excel: charts embedded in a worksheet (ChartObject) and charts that are a sheet (ChartSheet). OfficePartner supports ChartObjects only. Charts are added to a worksheet using the Charts.Add method of the parent worksheet. You can then set the position and size of the chart using the Height, Left, Top, and Width properties of this class. The ChartType property of this class defines the style of the chart: pie, bar, etc. Data is supplied to the chart by referencing the data in the cells described by the DataRange property.

## Hierarchy

TCollectionItem (VCL)

① TOpNestedCollectionItem (OpOfficeShared)438

TOpExcelChart (OpExcelClient)

## Properties

AsChart	Left	Top
AsChartObject	① ParentItem	① Verb
ChartType	① RootCollection	① VerbCount
DataRange	① RootComponent	Width
Height	① SubCollection	
① Intf	① SubCollectionCount	

## Methods

Activate	① Create	ExecuteVerb
Connect	Destroy	SetAddressFromSelection

---

## Reference Section

### Activate

method

```
procedure Activate;
```

- ↳ Activates the ChartObject represented by the collection item.

This method brings the workbook and worksheet containing the ChartObject to the front in Excel. If Excel is not visible or is not connected, this method has no effect.

See also: Connect

### AsChart

read-only property

```
property AsChart : Chart
```

- ↳ Allows access to the underlying automation interface for the chart.

The functionality of this interface is largely encapsulated within the OfficePartner framework. This interface property allows you to extend the functionality of the ChartObject beyond that which is encapsulated for you within OfficePartner. It is not practical to document all of the functionality of Microsoft's ChartObject. It is recommended that you consult the source code and any additional information from Microsoft (for example, MSDN) prior to using the methods and properties of this interface.

### AsChartObject

read-only property

```
property AsChartObject : ChartObject
```

- ↳ Allows access to the underlying automation interface for the ChartObject.

The functionality of this interface is largely encapsulated within the OfficePartner framework. This interface property allows you to extend the functionality of the ChartObject beyond that which is encapsulated for you within OfficePartner. It is not practical to document all of the functionality of Microsoft's ChartObject. It is recommended that you consult the source code and any additional information from Microsoft (for example, MSDN) prior to using the methods and properties of this interface.

---

**ChartType** **property**

`property ChartType : TOpXLChartType`

↳ Specifies the type of chart.

This property defines the basic style of the chart (eg area, bar, line). Possible values for the chart type constants (TxlChart) are defined below:

**Table 25:**

<b>Value</b>	<b>Results</b>
XlctArea	2-dimensional stacked area
XlctBar	2-dimensional clustered bar
XlctLine	2-dimensional line
XlctPie	2-dimensional pie
XlctRadar	Radar with markers at each data point
XlctXYScatter	Comparison of pairs of values
xlct3Darea	3-dimensional stacked area
xlct3Dbar	3-dimensional clustered bar
xlct3Dline	3-dimensional line
xlct3Dpie	3-dimensional pie
xlct3Dcolumn	3-dimensional vertical bar
XlctDoughnut	Similar to pie but can contain multiple series
xlctUnknown	Custom

Consult Microsoft's Excel documentation for further information regarding chart types.

See also: DataRange

---

**Connect** **method**

`procedure Connect;`

↳ Activates the parent worksheet.

If the TOpExcel component's PropDirection property is set to pdToServer, then the Excel chart is initialized to the values of the TOpExcelChart's properties. Otherwise, the properties of the TOpExcelChart object are initialized by the Excel chart properties. If Excel is not visible or is not connected, this method has no effect.

---

See also: Activate

**DataRange**
**property**

```
property DataRange : string
```

↳ The address of the charted data in Excel's A1 notation.

The values within this address contain a column of category headers on the left (rows) and series labels on the top (columns).

See also: ChartType

**Destroy**
**destructor**

```
destructor Destroy;
```

↳ Deletes this instance of the TOpExcelChart.

This method should not be called directly. Excel handles the destruction of ChartObjects when the worksheet containing the chart is closed.

See also: TOpExcelCharts.Add

**ExecuteVerb**
**method**

```
procedure ExecuteVerb(index : Integer);
```

↳ Invokes the SetAddressFromSelection method if Index equals 0.

This method should not be used. Instead, you should call the SetAddressFromSelection method directly.

See also: SetAddressFromSelection

**Height**
**property**

```
property Height : Integer
```

↳ The height, in pixels, of the ChartObject within the worksheet.

Use this property in conjunction with the Left, Top, and Width properties to determine the size and position of the chart.

See also: Left, Top, Width

**Left****property**

```
property Left : Integer
```

↳ The left-most position, in pixels, of the ChartObject within the worksheet.

Use this property in conjunction with the Height, Top, and Width properties to determine the size and position of the chart.

See also: Height, Top, Width

**SetAddressFromSelection****method**

```
procedure SetAddressFromSelection;
```

↳ Assigns the chart's DataRange to the currently selected range in the Excel worksheet.

This method provides a handy way to set the DataRange of the ChartObject based on the user's input.

See also: DataRange

**Top****property**

```
property Top : Integer
```

↳ The top position, in pixels, of the ChartObject within the worksheet.

Use this property in conjunction with the Height, Left, and Width properties to determine the size and position of the chart.

See also: Height, Left, Width

**Width****property**

```
property Width : Integer
```

↳ The width, in pixels, of the ChartObject within the worksheet.

Use this property in conjunction with the Height, Left, and Top properties to determine the size and position of the chart.

See also: Height, Left, Top

# TOpExcelHyperlinks Class

Every TOpExcelWorksheet object contains a TOpExcelHyperlinks object. This object is a collection of zero or more TOpExcelHyperlink objects. Hyperlinks are added to a worksheet by invoking the Hyperlinks.Add method of the worksheet object.

## Hierarchy

TCollection (VCL)

① TOpNestedCollection (OpOfficeShared)434

TOpExcelHyperlinks (OpExcelClient)

## Properties

① ItemName

Items

① ParentItem

RootCollection

① RootComponent

## Methods

Add

① Create

① FindItem

① ForEachItem

## Reference Section

### Add

### method

```
function Add : TOpExcelHyperlink;
```

↳ Adds a new hyperlink to the parent worksheet.

This function returns a TOpExcelHyperlink whose properties can be set as required. The following code illustrates how to add a hyperlink and set basic properties:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  Link : TOpExcelHyperlink;
begin
  Link := xl.Workbooks[0].Worksheets[0].Hyperlinks.Add;
  Link.AnchorCell := 'A1';
  Link.Address := 'http://www.turbopower.com';
  Link.Visible := True;
end;
```

Items	property
property Items[Index : Integer] : TOpExcelHyperlink	

Contains individual hyperlink items.

Index is a zero based index of a TOpExcelHyperlink object.

Use the Items property to access individual hyperlinks. Items is the default array property for TOpExcelHyperlinks. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
Excel1.Workbooks[0].WorkSheets[0].hyperlinks.Items[0].  
DoSomething;  
  
Excel1.Workbooks[0].WorkSheets[0].hyperlinks[0].  
DoSomething;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add

# TOpExcelHyperlink Class

A TOpExcelHyperlink item represents a hyperlink in the parent worksheet. There are two types of hyperlinks in Excel, those that define a URL and those that define a link to another cell within the spreadsheets of the parent workbook.

Hyperlinks that define URLs automatically launch your default web browser. The Web page address should be specified in the Address property. To define a hyperlink to another cell within one of the parent workbook's worksheets, you must supply a cell reference in Excel A1 format in the SubAddress property.

By default, the text displayed in a cell containing a hyperlink is the address of the hyperlink itself. To set the text to something other than the Address or SubAddress property value, you must add a range to the corresponding cell and then set its SimpleText property to the string that you want displayed in the cell. You can then add a hyperlink to the same cell, specifying the address of the link.

## Hierarchy

TCollectionItem (VCL)

① TOpNestedCollectionItem (OpOfficeShared)438

TOpExcelHyperlink (OpExcelClient)

## Properties

Address	① ParentItem	① SubCollectionCount
AnchorCell	① RootCollection	① Verb
AsHyperlink	① RootComponent	① VerbCount
① Intf	SubAddress	Visible
NewWindow	① SubCollection	

## Methods

① Activate	Create	ExecuteVerb
Connect	Destroy	Follow

## Reference Section

3

### Address property

---

property Address : string

↳ A URL that specifies a location on the Web.

The full address and protocol must be supplied in order to successfully launch your web browser and navigate to the Web page. For example, to point the link to TurboPower's home page, you would use code like this:

```
var
  HyperLnk : TOpExcelHyperlink
Begin
  HyperLnk := Worksheets[0].hyperlinks.add;
  HyperLnk.Address := 'http://www.turbopower.com';
  HyperLnk.AnchorCell := 'A1';
  HyperLnk.Visible := True;
end;
```

See also: AnchorCell, Follow

### AnchorCell property

---

property AnchorCell : string

↳ An Excel range in A1 notation that denotes where in the parent worksheet the hyperlink should appear.

Hyperlinks use the text that is currently in the cell. If hyperlinks are moved or deleted, the text remains but the style formatting is lost. This is standard Excel behavior.

See also: Address, Follow

### AsHyperlink read-only property

---

property AsHyperlink : Hyperlink

↳ Allows access to the underlying automation interface for the hyperlink.

The functionality of this interface is largely encapsulated within the OfficePartner framework. This interface property allows you to extend the functionality of the Hyperlink object beyond that which is encapsulated within OfficePartner. Unfortunately, it is not practical to document all of the functionality of Microsoft's Hyperlink object. It is recommended that you consult the source code and any additional information from Microsoft (for example, MSDN) prior to using the methods and properties of this interface.

**Connect****method**

```
procedure Connect;
```

- ↳ Activates the parent worksheet.

If the PropDirection of the TOpExcel component is set to pdToServer then the Excel spreadsheet is updated to reflect the values of the hyperlink. Otherwise, the properties of this instance are initialized by the values contained in the Excel spreadsheet. If the Excel server is not visible or is not connected, this property has no effect.

**Destroy****destructor**

```
destructor Destroy;
```

- ↳ Deletes this instance of the hyperlink.

This method should not be called directly. Rather, Excel automatically handles the destruction of hyperlink objects when the parent worksheet is closed.

**ExecuteVerb****method**

```
procedure ExecuteVerb(index : Integer);
```

- ↳ Invokes the Follow method if and only if index equals 0.

The Follow method should be called directly instead. Passing any other parameter besides 0 to this method has no effect.

See also: Follow

**Follow****method**

```
procedure Follow;
```

- ↳ Causes the hyperlink to be followed.

If the value specified in Address is a URL, then your internet browser is launched. If the value specified in SubAddress is an Excel reference, then Excel navigates to the proper cell location. The state of the window displayed when a link is followed is controlled by the NewWindow property.

See also: Address, SubAddress, NewWindow

**NewWindow****property**

```
property NewWindow : Boolean
```

- ↳ Determines whether a link is displayed in Excel or a browser.

This property is only applicable to Office 2000. Setting NewWindow to False will cause the browser to open in the current window. Previous versions of Office mandate that the browser opens in a new window.

See also: Follow

**SubAddress****property**

```
property SubAddress : string
```

- ↳ The target of the hyperlink within Excel in A1 notation.

Specifying a value for this property forces the hyperlink to point to a local cell within one of the worksheets belonging to the parent workbook.

Example:

```
Link.SubAddress := 'Sheet1!C12';
```

When the above link is followed, cell C12 of Sheet1 becomes the active cell.

See also: Address, AnchorCell, Follow

**Visible****property**

```
property Visible : Boolean
```

- ↳ Specifies whether the hyperlink can be seen or not.

A hyperlink whose visible property is set to False cannot be followed.

See also: Follow

---

## Chapter 4: Working with Outlook

The Microsoft Outlook object model exposes Outlook objects, which you can use to gain programmatic access to Outlook functionality. OfficePartner gives you the ability to create custom Outlook objects and manipulate those objects from within Outlook or from another application. OfficePartner's encapsulation of the Outlook reference model differs from OfficePartner's encapsulation of Excel, Word, and PowerPoint in one significant way: there is no `Visible` property of the `TOpOutlook` component. Therefore, Outlook's standard user interface, by default, is hidden. This is actually by design because Outlook's user interface varies depending on which folders and item types are currently active. This chapter includes instructions for the proper methods for making Outlook's user interface visible should the need arise.

# Understanding the Application and MAPINamespace Objects

When you manipulate Outlook objects, you always start with the Application object. There can only be one instance of Outlook available at one time. If an instance of Outlook is already running, then setting the Connected property of the TOpOutlook component to True returns a reference to the running instance. If Outlook is not running, then setting Connected to true will launch an instance of Outlook.

4

The Outlook server contains a MAPINamespace object that is a container for Outlook folders.

In understanding Outlook, there are two fundamental concepts that must be discussed: folders and items. In simplest terms, Outlook could be considered an item manager that organizes items in folders. Understanding how to control Outlook essentially boils down to understanding these two elements.

## Items

Outlook is designed to handle several specific types of items: Mail items, Appointment items, Contact items, Task items, Journal items, Note items, Post items, and DistributionList items. Outlook provides different user interfaces for each of the item types. For example, when you click on the Calendar within Outlook, the user interface depicts a calendar. Similarly, when you click a contact, a different form is displayed that captures information specific to contacts. The same is true for the rest of Outlook's item types. The following table depicts Outlook's different item types and OfficePartner's associated constants.

**Table 1:**

<b>Outlook item type</b>	<b>OfficePartner constant</b>
Mail item	olitMail
Appointment item	olitAppointment
Contact item	olitContact
Task item	olitTask
Journal item	olitJournal
Note item	olitNote
Post item	olitPost
Distribution list item	oldistributionlistitem

The following code creates a MailItem type and sends it to a recipient:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  mi : TOpMailItem;
begin
  if Outlook1.Connected = False then
    Outlook1.Connected := True;
  mi := Outlook1.CreateMailItem;
  mi.Recipients.add('xyz@somewhere.com');
  mi.Subject := 'Greeting';
  mi.Body := 'Hi there!!!';
  mi.Send;
end;
```

## Folders

You're probably very familiar with the fact that folders present a hierarchical system of storing related information. Outlook folders are no different, but Outlook goes one step further. All items contained in a single Outlook folder must be of the same type and a default item type is associated with every Outlook folder. Additionally, Outlook has the ability to transform types when necessary. The result of such a transformation can be adversely affected by disparities in the types, however.

The following table depicts the default folders for every instance of Outlook along with their corresponding OfficePartner constant:

**Table 2:**

<b>Default Outlook folder name</b>	<b>OfficePartner constant</b>	<b>Default Outlook item type</b>
Calendar	olFolderCalendar	Appointment item
Contacts	olFolderContacts	Contact item
Deleted Items	olFolderDeletedItems	Post/Mail item
Drafts	olFolderDrafts	Post/Mail item
Inbox	olFolderInbox	Post/Mail item
Journal	olFolderJournal	Journal Entry item
Notes	olFolderNotes	Note item
Outbox	olFolderOutbox	Post/Mail item
Sent Items	olFolderSentMail	Post/Mail item
Tasks	olFolderTasks	Task item

Each of the folders appearing in the preceding table can be accessed using OfficePartner. The following code displays the view for the default Notes folder:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  Folder : MapiFolder;
begin
  if Outlook1.Connected = False then
    Outlook1.Connected := True;
  Folder := Outlook1.
  MapiNamespace.GetDefaultFolder(olFolderNotes);
  Folder.Display; {show Outlook}
end;
```

OfficePartner encapsulates classes to model each of these Outlook items. You can create object instances that model each of the Outlook items and then use the object to set additional properties of the item.

You use the Outlook Application object's GetNameSpace method to instantiate an object variable representing a recognized data source.

The following code demonstrates how to access the application's MAPINamespace:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ns : _NameSpace;
begin
  with Outlook1 do begin
    if Connected = False then
      connected := True;
    ns := Server.GetNamespace('MAPI');
  end;
end;
```

Currently, Outlook supports the MAPI message store as the only valid NameSpace object. Therefore, "MAPI" is the only string that you should ever pass to the GetNameSpace function.

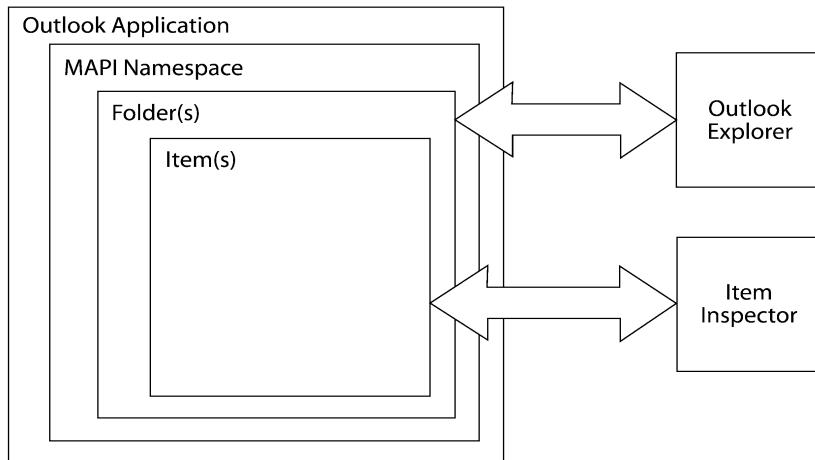
If Outlook is not running when you create a NameSpace object variable, and the user's mail services startup setting requires a profile, the user is prompted for a profile. Startup settings are on the Mail Services tab of Outlook's Options dialog box (Tools menu). You can use the NameSpace object's Logon method to specify a profile programmatically. In Windows 95 and 98, profiles are stored in the Windows registry under the following key:

`\HKEY_CURRENT_USER\Software\Microsoft\Windows Messaging`

In Windows NT and Windows 2000, the profiles are stored under the following key:

\HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem

The relationship between the Application, NameSpace, Folder, and Item object types is depicted in the illustration below:



Each Folder is associated with an explorer window that allows you to view the contents of the folder. Each ItemType is associated with an inspector (editor) window that allows you to edit the individual properties of the item.

## Working with Outlook folders and items

You can think of the **NameSpace** object as the gateway to all existing Outlook folders. By default, Outlook creates two top-level folders representing all public folders and all mailbox folders. Mailbox folders contain all Outlook built-in and custom folders. Each folder is a **MAPIFolder** object. **MAPIFolder** objects can contain subfolders (which are also **MAPIFolder** objects), as well as individual Outlook item objects, such as Mail item objects, Contact item objects, Journal item objects, and so on.

Note: In Outlook, an “item” is the object that holds information (similar to files in other applications). Items include mail messages, appointments, contacts, tasks, journal entries, and notes.

Once you have created an Application object and accessed its NameSpace object variable, you can access the top-level folder for any built-in Outlook item by using the NameSpace object's GetDefaultFolder method. The following code demonstrates this process:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ns : _NameSpace;
  mf : MAPIFolder;
begin
  with Outlook1 do begin
    if Connected = False then
      connected := True;
    {get the MAPI NameSpace}
    ns := Server.GetNamespace('MAPI');
    {Access the default note folder }
    mf := ns.GetDefaultFolder(olFolderNotes);
    {display the (Notes) explorer window}
    mf.Display;
  end;
end;
```

You can also return a reference to any folder by using the name of the folder. Folders may contain additional folders, individual Outlook items, or both.

## Events in Outlook

There are two classes of events in Outlook, and you work with each class differently. The first class represents item-level events that are associated with a particular Outlook item. For example, an Outlook Mail item object has events representing actions such as Open, Close, Forward, and Send.

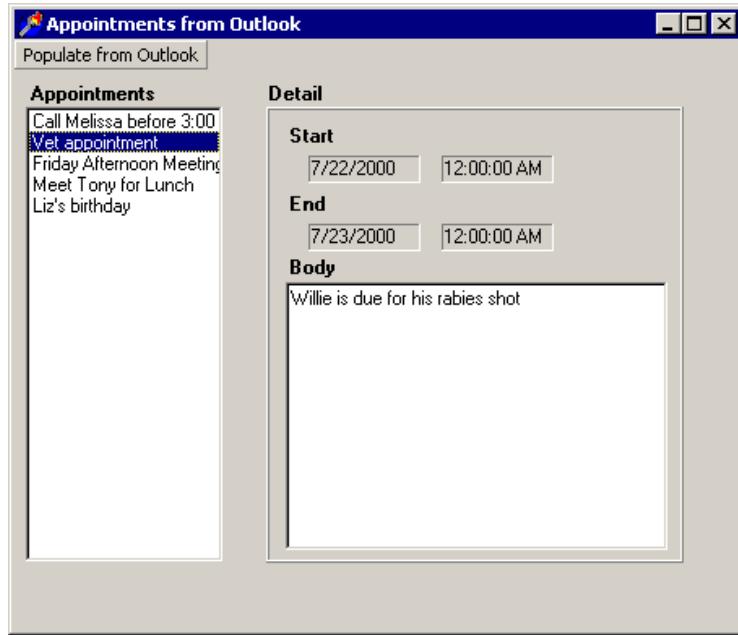
The second class of events supported in Outlook represents application-level events. These events are associated with the application itself, or with top-level objects within the application, such as navigation of folders or changes to the Outlook Bar.

## Outlook Examples

The four example programs in this section illustrate commonly used Outlook features: the calendar, e-mail, address lists and interfacing with Excel. The code for the programs is located in the “examples” folder.

### Example 1: Accessing calendar information

This demonstration describes how to use OfficePartner to gain access to the information stored in your Microsoft Outlook calendar. The program is called ExAppts.dpr and is located in the “examples” folder. The following figure shows the ExAppts.dpr window:



This application uses a single TOpOutlook component whose properties are all set to their defaults. When the user clicks the Populate From Outlook menu item, the TOpOutlook.Connected property is set to True and the list box is filled with each appointment item you have set up in Outlook (this demonstration assumes that you have at least one calendar item established in Outlook). As with other TOpOffice components, setting the Connected property to True launches the associated Office application. Unlike Word, Excel, or PowerPoint, however, TOpOutlook does not have a Visible property to set;

Outlook always runs in the background. The calendar information stored in Outlook is extracted in the Populate method (of the form). The code for the Populate method is shown here:

```

procedure TForm1.Populate;
var
  i : Integer;
  oMAPIFolder : MAPIFolder;
  oItems : Items;
  oAppt : _AppointmentItem;
  {calendar folder contains these}
  oApptItem : TOpAppointmentItem;
  {Delphi wrapper object for _AppointmentItem}

begin
  ListBox1.Clear;
  Memo1.Clear;
  with OpOutlook1 do begin
    if not Connected then
      Connected := True;

    oMAPIFolder := OpOutlook1.
    MAPINamespace.GetDefaultFolder(olFolderCalendar);
    if assigned( oMAPIFolder ) then begin
      oItems := oMAPIFolder.Items;
      if oItems.Count > 0 then
        for i := 1 to oItems.Count do
          {Items uses a 1 based index}
          if oItems.Item(i).QueryInterface(
            _AppointmentItem, oAppt ) = s_OK then begin
            oApptItem := TOpAppointmentItem.
              Create( oAppt );
            ListBox1.Items.AddObject(
              oApptItem.Subject, oApptItem )
          end
        end
      end
    end;
  end;
end;
```

As usual, setting the Connected property of the Office component to True establishes the connection with the associated Office application. This is accomplished in the following code:

```

with OpOutlook1 do begin
  if not Connected then
    Connected := True;
```

Access to one of Outlook's folders is provided by the following line of code:

```
OpOutlook1.  
  MAPINamespace.GetDefaultFolder (olFolderCalendar);
```

OfficePartner exposes an interface called MAPINamespace as a member of TOpOutlook. GetDefaultFolder is a public method of this interface. The GetDefaultFolder method takes as a parameter an integer constant that defines which folder you want to access. The constants specify one of the ten default folders in Outlook as described earlier.

In this example, we have told Outlook to give us the folder containing appointments (the Calendar folder).

Next, we confirm that the folder requested does in fact exist, and if so, we assign the items of this folder to the local variable oItems.

```
if assigned(oMAPIFolder) then begin  
  oItems := oMAPIFolder.Items;
```

We are now free to iterate the list of calendar items using the same approach used for iterating any VCL Items property.

```
for i := 1 to oItems.Count do  
  {Items uses a 1 based index}  
  if oItems.Item(i).QueryInterface(   
    AppointmentItem, oAppt ) = s_OK then begin  
    oApptItem := TOpAppointmentItem.Create( oAppt );  
    ListBox1.Items.AddObject(   
      oApptItem.Subject, oApptItem )  
  end
```

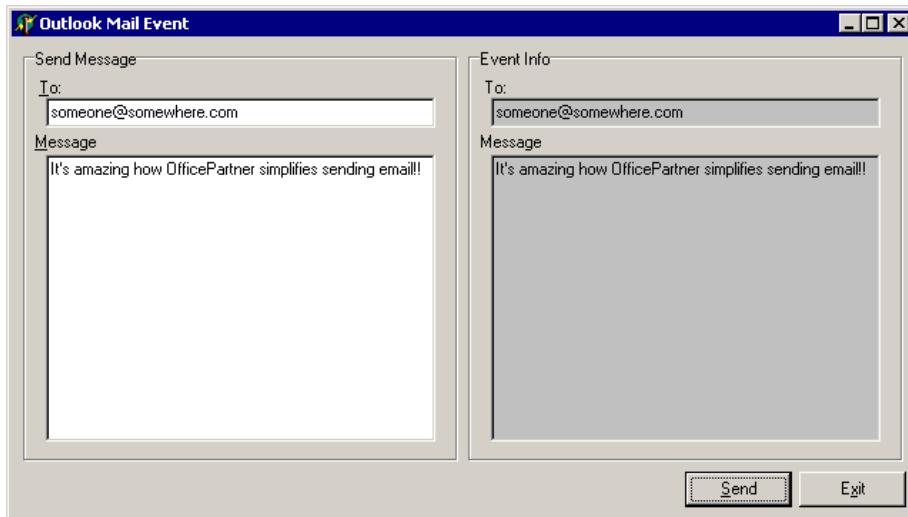
The preceding code requires explanation. For components that support interfaces, QueryInterface calls the QueryInterface method of the interface supported by the component. This function returns a reference to the interface specified by the IID parameter, as the out parameter (the first parameter). If the component does not support the interface, the out parameter is nil.

Essentially, this code determines whether the item referred to by oItems.Item(i) implements the \_AppointmentItem interface. If it does implement the interface, then a reference (pointer to the interface) is returned in the out parameter .

Next, we create an instance of TOpAppointmentItem. This object is then added (using the AddObject function) to the Items property of the list box. We are then able to extract specifics about the appointment (e.g. dates and times) by referring to this object in the ListBox1Click event.

## Example 2: Sending an e-mail

This demonstration introduces composing and sending an e-mail message as well as responding to the OnItemSend event of the TOpOutlook component. The program is located in the “examples” folder. The following figure shows how the example appears on the desk top.



Consider the following code:

```
procedure TFrmMain.BtnSendClick(Sender : TObject);
begin
  with OpOutlook.CreateMailItem do
    try
      MsgTo := EdtTo.Text;
      Subject := 'OpOutlook Event Test';
      HTMLBody := MemMsg.Lines.Text;
      Send;
    finally
      Free;
    end;
end;
```

The TOpOutlook component contains a member function CreateMailItem that returns an instance of TOpMailItem class. The TOpMailItem has properties corresponding to the recipient's address (MsgTo), the actual message (HTMLBody), and the subject (Subject) among others. These are all simple string properties that determine the composition and routing of your mail message. The TOpMailItem.Send procedure actually does the work of

sending the mail item to the recipient. This example also demonstrates how to respond to the OnItemSend event, which OfficePartner has exposed as a standard VCL event. Here we simply respond to the event by confirming the mail item entries in the Event Info fields on the right side of the screen. Here is the code that accomplishes this:

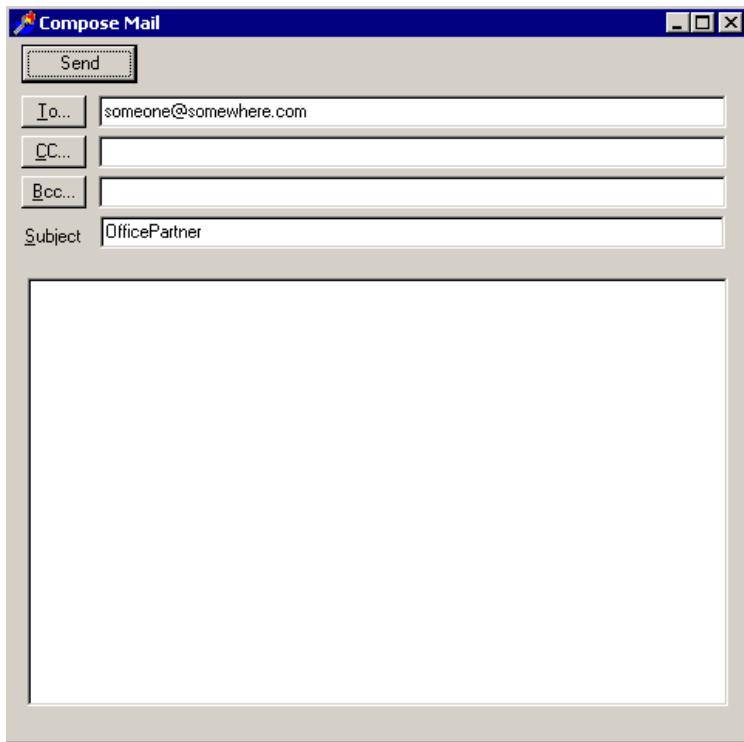
```
procedure TFrmMain.OpOutlookItemSend(
  const Item : IDispatch; var Cancel : WordBool);
var
  Mail : TOpMailItem;
begin
  if Item <> nil then begin
    Mail := TOpMailItem.Create(Item as _MailItem);
    try
      EdtEventTo.Text := Mail.MsgTo;
      MemEventMsg.Lines.Text := Mail.Body;
    finally
      Mail.Free;
    end;
  end;
end;
```

As you can see, there is very little programming intervention required of you to send e-mail and respond to Outlook events.

### Example 3: Accessing address lists

This demonstration builds on the simple mailer from Example 2 and allows selection of recipients from your address lists. The code for this example is contained in the “examples” folder and includes a fair amount of user-interface code. In the explanation that follows, we have left the user-interface code for you to decipher. Instead, we’ll focus on the procedures and methods that harness the power of Outlook in the client program.

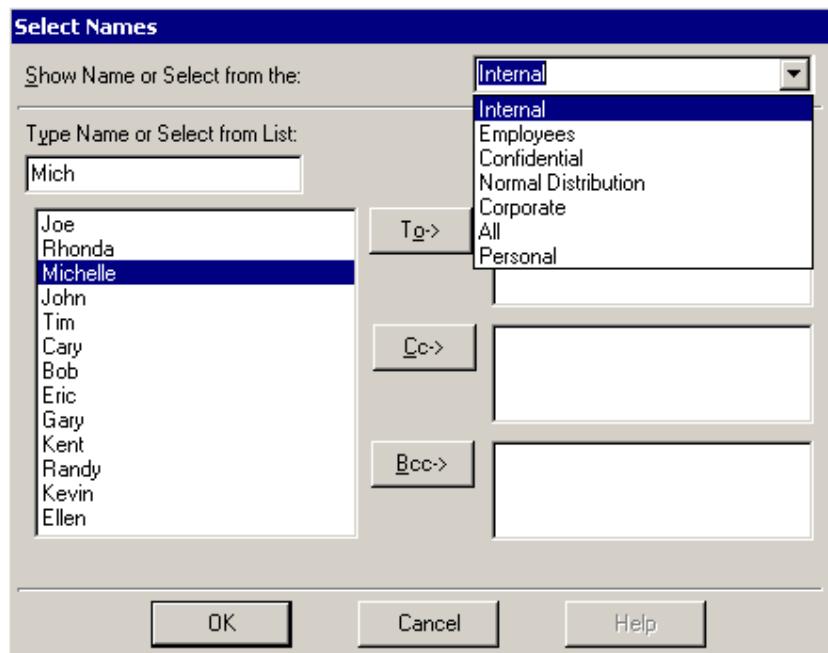
The first major difference in this demonstration is that the user no longer has to manually enter the recipient's address. To achieve this, the demonstration acquires the names of the address lists and the name of the items contained within each list. The following figure shows example 3 running:



Clicking the To button will execute the following code:

```
if fOutlook.MapiNamespace.AddressLists.Count > 0 then
  for i := 1 to fOutlook.
    MapiNamespace.AddressLists.Count do
      cbAddressLists.Items.Add(fOutlook.
        MapiNamespace.AddressLists.Item(i).Name );
```

This code iterates through all of your existing address lists and populates a combo box with the name of each list as shown in the following figure:



Now that we have the names of the address lists, how do we discover the names (and addresses) of each list's members? Selecting an address list name in the combo box shown in the preceding figure will execute the following code (procedure TDlgSelectNames.cbAddressListsChange):

```

  Sender : TObject);
var
  idx : integer;
  oAddressList : AddressList;
  i : Integer;
  oAddress : TAddressEntryWrapper;
begin
  cbAddressLists.Update;
  FlushNamesListBox;
  idx := cbAddressLists.Items.IndexOf(
    cbAddressLists.Text );
  if idx >= 0 then begin
    oAddressList := fOutlook.
      MapiNamespace.AddressLists.Item(idx + 1);
    if assigned( oAddressList ) then
      if oAddressList.AddressEntries.
        Count > 0 then begin
        lbNames.Items.BeginUpdate;
        try
          for i := 1 to
            oAddressList.AddressEntries.Count do begin
              oAddress := TAddressEntryWrapper.Create;
              oAddress.Address := oAddressList.
                AddressEntries.Item(i);
              lbNames.Items.AddObject(
                oAddress.Address.Name, oAddress )
            end
        finally
          lbNames.Items.EndUpdate;
        end;
      end;
    end;
  end;
end;

```

The execution of the preceding code populates the list box of the running example (shown in preceding figure) with the names and addresses in the selected address list. The local variable oAddressList is assigned to the address list selected (from a combo box) by the user. (The index of the selected address list is determined by its index in the combo box):

```

oAddressList := fOutlook.
  MapiNamespace.AddressLists.Item(idx + 1);

```

## A word of caution

Interfaces cannot be reliably stored in a pointer list without directly handling the \_AddRef and \_Release methods of the IUnknown interface. We ultimately want to add each AddressEntry to a list box (using the AddObject procedure of the list box). To allow this, we must either implement the \_AddRef and \_Release methods directly, or create a wrapper class around the AddressEntry interface. For simplicity, we have chosen the latter approach. We have defined a class called TAddressEntryWrapper, which simply wraps the AddressEntry interface, thus enabling us to add the entry to a list box. The TAddressEntryWrapper is defined and implemented as follows:

```
TAddressEntryWrapper = class( TObject )
private
  fAddress : AddressEntry;
public
  {:: Return a duplicate of self with Address assigned }
  function Clone : TAddressEntryWrapper;
  property Address :
    AddressEntry read fAddress write fAddress;
end;

{implementation}
function
  TAddressEntryWrapper.Clone : TAddressEntryWrapper;
begin
  result := TAddressEntryWrapper.Create;
  result.Address := Address;
end;
```

## Continuing onward

Given the reference to the address list, we can now iterate through all of the member names belonging to this list and add them to our list box (lbNames).

```
for i := 1 to oAddressList.
  AddressEntries.Count do begin
  oAddress := TAddressEntryWrapper.Create;
  oAddress.Address :=
    oAddressList.AddressEntries.Item(i);
  lbNames.Items.AddObject(
    oAddress.Address.Name, oAddress)
end;
```

For each address entry associated with the selected address list, we create the wrapper class and add the name (a string) and wrapper class to the list box's items property.

The preceding code allows the user to select the address list as well as the recipient(s) from those already defined in Outlook. All that is left to do is let the user compose and send the message exactly as was done in the second demonstration.

## Example 4: Sharing Outlook information with Excel

This example, ExO2XL.dpr, ties a few of the topics discussed in the Excel demonstration “Exporting all customers to Excel” on page 57 with those discussed in the Outlook demonstrations. Additionally, we’ll introduce a new component—the TOpContactsDataSet.

The TOpContactsDataSet is a component which is connected to a TOpOutlook component. TOpContactsDataSet provides convenient access to Outlook’s contacts database.

The TOpContactsDataSet is a direct descendant of the VCL’s TDataSet, and, as such, is able to provide a conduit between the data stored in Outlook’s contacts database and a range object. For this example, we’ll use an Excel range, but this could just as easily be a range within a Word document.

First, drop a TOpContactsDataSet component on the form and set its Outlook property to the TOpOutlook component. Then, drop a TOpDataSetModel component on the form and set its Dataset property to the TOpContactsDataSet above.

The entire source code for this is contained in the following procedure:

```
procedure TForm1.btnExitClick(Sender : TObject);
var
  oCursor : TCursor;
  oWorkBook : TOpExcelWorkbook;
  oSheet : TOpExcelWorksheet;
  oRange : TOpExcelRange;
begin
  oCursor := Screen.Cursor;
  Screen.Cursor := crHourglass;
  try
    { check all connections }
    if not OpOutlook1.Connected then
      OpOutlook1.Connected := True;
    if not OpExcel1.Connected then
      OpExcel1.Connected := True;
    if not OpContactsDataSet1.Active then
      OpContactsDataSet1.Active := True;

    with OpExcel1 do begin
      oWorkBook := Workbooks.Add;
      {create a workbook}
      oSheet := oWorkBook.Worksheets.Add;
      {add a worksheet}
      oRange := oSheet.Ranges.Add;
      {create range for output}
      oRange.Address := 'A1';
      {locate range}
      oRange.OfficeModel := OpDatasetModell1;
      {assign the model to the range}
      oRange.Populate;
      {fill the range from the model}
      Visible := True;
      {show it}
      Interactive := True
      {Let the user access it}
    end;
  finally
    Screen.Cursor := oCursor;
  end;
end;
```

We first launch both Excel and Outlook, and open the contacts dataset with the following lines:

```
if not OpOutlook1.Connected then
    OpOutlook1.Connected := True;
if not OpExcel1.Connected then
    OpExcel1.Connected := True;
if not OpContactsDataSet1.Active then
    OpContactsDataSet1.Active := True;
```

We then add a workbook, worksheet, and a range to the Excel application object as follows:

```
oWorkBook := Workbooks.Add;
    {create a workbook}
oSheet := oWorkBook.Worksheets.Add;
    {add a worksheet}
oRange := oSheet.Ranges.Add;
    {create range for output}
```

The anchor cell of the range is then set to reference cell A1 and the OfficeModel property of the range is pointed to the TOpDataSetModel component.

```
oRange.Address := 'A1';
    {locate range}
oRange.OfficeModel := OpDatasetModel1;
    {assign the model}
```

Finally, the Populate method of the range is called and the Excel application object made visible as follows:

```
oRange.Populate;           {fill the range}
OpExcel1.Visible := True;  {show it}
```

The effect of all of this is that now Outlook's contact database now appears in an Excel worksheet as show in the following figure:

A screenshot of Microsoft Excel showing a contact list imported from Outlook. The Excel window title is "Microsoft Excel - Book1". The menu bar includes File, Edit, View, Insert, Format, Tools, Data, Window, and Help. The toolbar contains various icons for file operations, search, and data manipulation. The spreadsheet has columns labeled A1 through Ma. Row 1 contains column headers: FirstName, LastName, MiddleName, Suffix, MailingAddressCity, MailingAddressCountry, and Ma. Rows 2 through 9 contain data for US Presidents: Bill Clinton, George Bush, Ronald Reagan, Jimmy Carter, Gerald Ford, Lyndon Johnson, Dwight Eisenhower, and John Kennedy. Row 10 is blank. The status bar at the bottom shows "Ready". A small "Outlook to Excel" dialog box is overlaid on the Excel window, containing a single button labeled "Contacts to Excel".

	A	B	C	D	E	F	Ma
1	FirstName	LastName	MiddleName	Suffix	MailingAddressCity	MailingAddressCountry	Ma
2	Bill	Clinton					
3	George	Bush					
4	Ronald	Reagan					
5	Jimmy	Carter					
6	Gerald	Ford					
7	Lyndon	Johnson					
8	Dwight	Eisenhower					
9	John	Kennedy					
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

---

## TOpOutlookBase Class

TOpOutlookBase class is the immediate ancestor of the TOpOutlook component. It implements all of the methods and properties used by the TOpOutlook component and is identical to the TOpOutlook component except that no properties are published.

TOpOutlookBase is provided to facilitate creation of descendent components. For property and method descriptons, see “TOpOutlook Component” on page 143.

### Hierarchy

TComponent (VCL)

TOpOfficeComponent (OpShared)

TOpOutlookBase (OpOlk2K)





# TOpOutlook Component

4

The TOpOutlook component represents an instance of Microsoft Outlook. This class contains a collection representing all folders in Outlook and each folder contains a collection of items of the same type. Folders are accessed using the MAPINamespace property. Items within folders are accessed by indexing into the folder's collection of items.

The PropDirection property determines whether data is sent from your application to Outlook or whether data is being acquired from an existing Outlook folder.

PropDirection controls what happens to the component properties when the component is initially connected. If PropDirection is pdToServer, the streamed design-time properties are pushed to Outlook immediately after Outlook is launched. If PropDirection is set to pdFromServer, the streamed properties are ignored and the component properties represent the state of Outlook when it is launched.

## Hierarchy

TComponent (VCL)

① TOpOfficeComponent (OpShared)443

TOpOutlookBase (OpOlk2K)

TOpOutlook (OpOutlk)

## Properties

① ClientState	NewSession	PropDirection
Connected	① OfficeVersion	Server
MachineName	① OfficeVersionStr	ShowLoginDialog
MapiNamespace	OutlookVersion	

## Methods

① AddConnectListener	CreateJournalItem	① GetFileInfo
① CoCreate	CreateMailItem	HandleEvent
① Create	CreateNoteItem	Login
CreateAppointmentItem	CreatePostItem	Logoff
CreateContactItem	CreateTaskItem	Quit
CreateInstance	① Destroy	① RemoveConnectListener
① CreateItem	GetAppInfo	SendMailMessage

## Events

OnItemSend	OnQuit
OnNewMail	OnReminder
OnOptionsPagesAdd	OnStartup

## Reference Section

### Connected

property

4

property Connected : Boolean

Default: False

↳ Indicates whether the Outlook application is connected to your application.

Outlook differs from Excel, Word, and PowerPoint in that contains no Visible property. When you set the Connected property to True, Outlook is hidden. To show Outlook's user interface, you must invoke the Display method of a folder or item.

Invoking the Display method of a folder displays the Explorer window for the folder. This window allows you to view the contents of the folder.

Invoking the Display method of an item displays the appropriate inspector window for that item type.

See also: MachineName, OnStartup, PropDirection, ShowLoginDialog

### CreateAppointmentItem

method

function CreateAppointmentItem : TOpAppointmentItem;

↳ Creates and returns a new TOpAppointmentItem instance.

This method creates appointment items in

Outlook. After creating the appointment item, you can set the parameters of the appointment item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ai : TOpAppointmentItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  ai := outlook1.CreateAppointmentItem;
  ai.Subject := 'Team meeting';
  ai.Save;
  ai.Free;
end;
```

The instance obtained from this method must be disposed of using the `TOpAppointmentItem.Free` method. To persist the appointment information in Outlook, you must call the `Save` method of the `TOpAppointmentItem` class.

See also: `CreateContactItem`, `CreateJournalItem`, `CreateMailItem`, `CreateNoteItem`, `CreatePostItem`, `CreateTaskItem`

### **CreateContactItem**

### **method**

---

```
function CreateContactItem : TOpContactItem;
```

Creates and returns a new `TOpContactItem` instance.

This method creates contact items in Outlook. After creating the contact item, you can set the parameters of the contact item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ci : TOpContactItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  ci := outlook1.CreateContactItem;
  ci.LastName := 'Smith';
  {...}
  ci.Save;
  ci.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpContactItem.Free method. To persist the appointment information in Outlook, you must call the Save method of the TOpContactItem class.

See also: CreateAppointmentItem, CreateJournalItem, CreateMailItem, CreateNoteItem, CreatePostItem, CreateTaskItem

---

<b>CreateInstance</b>	<b>method</b>
-----------------------	---------------

---

```
function CreateInstance : _Application;
```

 Returns the automation interface for the Outlook Server.

The majority of the functionality of this interface has been exposed in the form of methods and properties within OfficePartner. There may be times however, that you need to extend the functionality of OfficePartner beyond what is already encapsulated for you. This interface allows you to do just that. Unfortunately, it is impractical to document all of the methods and properties of this interface. You should consult the source code and any additional information from Microsoft (e.g. MSDN) prior to using this interface.

See also: Server

---

<b>CreateJournalItem</b>	<b>method</b>
--------------------------	---------------

---

```
function CreateJournalItem : TOpJournalItem;
```

 Creates and returns a new TOpJournalItem instance.

This method creates journal entrys in Outlook. After creating the journal entry, you can set the parameters of the contact item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ji : TOpJournalItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  ji := outlook1.CreateJournalItem;
  ji.Subject := 'Dear Diary';
  {...}
  ji.Save;
  ji.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpJournalItem.Free method. To persist the appointment information in Outlook, you must call the Save method of the TOpJournalItem class.

See also: CreateAppointmentItem, CreateContactItem, CreateMailItem, CreateNoteItem, CreatePostItem, CreateTaskItem

## CreateMailItem

## method

4

```
function CreateMailItem : TOpMailItem;
```

↳ Creates and returns a new TOpMailItem instance.

This method creates mail items in Outlook. After creating the mail item, you can set the parameters of the contact item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  mi : TOpmailItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  mi := outlook1.CreateMailItem;
  mi.Recipients.Add('xyz@turbopower.com');
  mi.Subject := 'Test';
  mi.Body := 'Testing...';
  mi.Send;
  mi.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpMailItem.Free method.

See also: CreateAppointmentItem, CreateContactItem, CreateJournalItem, CreateNoteItem, CreatePostItem, CreateTaskItem

```
function CreateNoteItem : TOpNoteItem;
```

↳ Creates and returns a new TOpNoteItem instance.

This method creates Note items in Outlook. After creating the Note item, you can set the parameters of the Note item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ni : TOpNoteItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  ni := outlook1.CreateNoteItem;
  ni.Body := 'Note to self...';
  {...}
  ni.Save;
  ni.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpNoteItem.Free method. To persist the appointment information in Outlook, you must call the Save method of the TOpNoteItem class.

See also: [CreateAppointmentItem](#), [CreateContactItem](#), [CreateJournalItem](#), [CreateMailItem](#), [CreatePostItem](#), [CreateTaskItem](#)

```
function CreatePostItem : TOpPostItem;
```

- ↳ Creates and returns a new TOpPostItem instance.

This method creates Post items in Outlook. After creating the Post item, you can set the parameters of the contact item as required. The following example demonstrates the proper approach to using this function:

4

```
procedure TForm1.Button1Click(Sender : TObject);
var
  pi : TOpPostItem;
begin
  if outlook1.Connected = False then
    outlook1.Connected := True;
  pi := outlook1.CreatePostItem;
  pi.Body := 'post test...';
  {...}
  pi.Save;
  pi.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpPostItem.Free method. To persist the information in Outlook, you must call the Save method of the TOpPostItem class.

See also: [CreateAppointmentItem](#), [CreateContactItem](#), [CreateJournalItem](#), [CreateMailItem](#), [CreateNoteItem](#), [CreateTaskItem](#)

## CreateTaskItem

method

```
function CreateTaskItem : TOpTaskItem;
```

↳ Creates and returns a new TOpTaskItem instance.

This method creates Task items in Outlook. After creating the Task item, you can set the parameters of the contact item as required. The following example demonstrates the proper approach to using this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ti : TOpTaskItem;
begin
  if outlook1.Connected = False then
    outlook1.connected := True;
  ti := outlook1.CreateTaskItem;
  ti.Subject := 'Test';
  {...}
  ti.Save;
  ti.Free;
end;
```

The instance obtained from this method must be disposed of using the TOpTaskItem.Free method. To persist the task information in Outlook, you must call the Save method of the TOpTaskItem class.

See also: CreateAppointmentItem, CreateContactItem, CreateJournalItem, CreateMailItem, CreateNoteItem, and CreatePostItem

## GetAppInfo

method

```
procedure GetAppInfo(Info : TStrings);
```

↳ Returns the product and version information from the Outlook application server.

```
Outlook1.GetAppInfo(Memo1.lines);
```

The preceding code adds lines to Memo1 similar to the following:

```
Product=Outlook
Version=9.0.0.3821
```

**HandleEvent****method**

```
procedure HandleEvent(
  const IID: TIID; DispId: Integer;
  const Params: TVariantArgList);
```

↳ Overridden in each TOpOfficeComponent subclass in order to correctly dispatch automation events to VCL Event Handlers.

IID contains the interface identifier common to all TOpOutlook events.

DispId contains an integer that maps a general TOpOutlook event to the specific event. For example, a DispId value of 61443 maps an event to the OnNewMail event.

Params contains an array of variants whose values depend on the specific event being triggered.

This method is called internally.

See also: [OnItemSend](#), [OnNewMail](#), [OnOptionsPagesAdd](#), [OnReminder](#), [OnStartup](#), [OnQuit](#)

**Login****method**

```
procedure Login(const Profile, Password : string);
```

↳ Logs on to Outlook with the specified profile name and password.

Profile is an optional string that identifies the name of the Outlook user profile you are attempting to log on to.

Password is the Outlook user password.

If ShowLoginDialog is set to True, then a dialog is presented to the user where they may log on manually. If ShowLoginDialog is False, the user is logged on transparently using the Profile and Password strings passed to the procedure.

See also: [ShowLoginDialog](#)

**Logoff****method**

```
procedure Logoff;
```

↳ Logs off from Outlook.

Setting the Connected property to False also causes this method to be called.

See also: [Connected](#), [Login](#), [ShowLoginDialog](#)

---

<b>MachineName</b>	<b>property</b>
--------------------	-----------------

---

**property MachineName : String;**

>Returns or sets the name of the server that Outlook is running on if DCOM is configured.

DCOM must be properly configured to control Outlook across machine boundaries. If DCOM is not properly configured, then any attempt to connect to the Outlook server displays the message “The Remote Procedure Call Server is Unavailable” and the connection fails.

See also: Connected, PropDirection

---

<b>MapINamespace</b>	<b>property</b>
----------------------	-----------------

---

**property MapINamespace : NameSpace**

Contains an interface to all of the Outlook folders.

This interface property provides access to any of Outlook’s default folders. For example, the following code assigns the default Calendar folder to the MAPIFolder variable Folder:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  Folder : MapIFolder;
begin
  if Outlook1.Connected = False then
    Outlook1.Connected := True;
  Folder := Outlook1.
    MapINamespace.GetDefaultFolder(olFolderCalendar);
  Folder.Display;
end;
```

The following table defines the Outlook folders and OfficePartner's associated constants. Each constant is a valid parameter that can be passed to the GetDefaultFolder method of the MAPINamespace object:

**Table 3:**

Default Outlook folder name	OfficePartner constant
Calendar	olFolderCalendar
Contacts	olFolderContacts
Deleted Items	olFolderDeletedItems
Drafts	olFolderDrafts
Inbox	olFolderInbox
Journal	olFolderJournal
Notes	olFolderNotes
Outbox	olFolderOutbox
Sent Items	olFolderSentMail
Tasks	olFolderTasks

See also: Connected

---

**NewSession** **property**

**property** NewSession : Boolean

Default: False

- ↳ Dictates whether a new session is started or the existing session is used when logging on via the Login method.

By default, if an instance of Outlook is already running, setting the Connected property to True connects to the existing instance of Outlook. A new instance of Outlook is not started. Setting the NewSession property to True guarantees that a new instance of Outlook is instantiated when the Connected property is set to True.

See also: Connected, ShowLoginDialog

## OnItemSend

event

```
property OnItemSend : TItemSendEvent  
  TItemSendEvent = procedure(  
    const Item : IDispatch; var Cancel : WordBool);
```

Defines an event handler that is called just before a TOpMailItem is sent.

Item is the IDispatch interface to the Outlook item being sent.

Setting the value of Cancel allows you to abort the send or proceed. Setting Cancel to True aborts the send.

See also: SendMailMessage

## OnNewMail

event

```
property OnNewMail : TOfficeEvent  
  TOfficeEvent = procedure;
```

Defines an event handler that is called when Outlook has received new mail.

See also: Connected, CreateMailItem,

## OnOptionsPagesAdd

event

```
property OnOptionsPagesAdd : TOptionsPagesAddEvent  
  TOptionsPagesAddEvent = procedure (  
    const Pages : PropertyPages);
```

Defines an event handler that is called whenever the Options dialog box (available through the Tools menu) or a folder Properties dialog box is opened.

Pages contains a collection of custom property pages to which you can add.

This event allows you to add a custom property page. A custom property page is an ActiveX control that is displayed by Outlook in the Options dialog box or in the folder Properties dialog box when the user clicks on the custom property page's tab.

See also: Connected

**OnQuit****event**

```
property OnQuit : TOfficeEvent  
TOfficeEvent = procedure;
```

↳ Defines an event handler that is called when the Outlook application server is terminated.

The Outlook application server is terminated by setting the Connected property to False.

See also: Connected, Login, OnStartup

**OnReminder****event**

```
property OnReminder : TReminderEvent  
TReminderEvent = procedure (const Item : IDispatch);
```

↳ Defines an event handler that is called when a reminder is due.

Item contains the IDispatch interface to the Outlook item that is due.

See also: OnStartUp

**OnStartup****event**

```
property OnStartup : TOfficeEvent  
TOfficeEvent = procedure;
```

↳ Defines an event handler that is called when the Outlook server is started.

The Outlook server is started by setting the Connected property to True.

See also: Connected, Login, OnQuit

## **OutlookVersion**

**read-only property**

```
property OutlookVersion : TOfficeVersion
```

↳ Returns the version of Outlook with which the component is currently communicating.

The following table depicts the possible values for this property:

**Table 4:**

<b>Value</b>	<b>Meaning</b>
ov97	Office 97
ov98	Office 98
ov2000	Office 2000
ovUnknown	Office version could not be determined

OutlookVersion always returns ovUnknown if the Connected property is False.

See also: Connected, Version

## **PropDirection**

**property**

```
property PropDirection : TPropDirection
```

```
TPropDirection = (pdToServer, pdFromServer);
```

Default: pdToServer

↳ Controls what happens to the component properties when the component is initially connected.

If PropDirection is pdToServer, the streamed design-time properties are pushed to Outlook immediately after Outlook is launched. If PropDirection is set to pdFromServer, the streamed properties are ignored and the component properties represent the state of Outlook when it is launched.

See also: Connected, MachineName

**Quit****method**

```
procedure Quit;
```

- ↳ Terminates the Outlook session and logs the current user off.

Any changes made to items that have not been saved are discarded. The Outlook component remains connected after calling this method.

See also: Connected, Login, Logoff, ShowLoginDialog

**SendMailMessage****method**

```
procedure SendMailMessage(
  const ToAddrs, CcAddrs, BccAddrs, MsgSubject: string;
  MsgText, MsgAttachments: TStrings);
```

- ↳ Provides a means for easily sending an Outlook mail message with one method call.

The ToAddrs, CcAddrs, BccAddrs parameters contains semicolon-delimited lists of, primary, carbon copy, and blind carbon copy recipients respectively. These can either be Internet e-mail addresses or names that can be resolved from the Outlook address book.

MsgSubject contains the subject of the message. MsgText parameter holds the text of the message.

The MsgAttachments parameter is a list of file names that serves as message attachments. You may pass nil in the MsgAttachments parameters if there are no attachments.

See also: CreateMailItem, TOpMailItem.Send

**Server****read-only property**

```
property Server : _Application
```

- ↳ Represents the Automation interface for the Outlook Server.

The majority of the functionality of this interface has been exposed in the form of methods and properties within the OfficePartner framework. It is recommended that you use these methods instead of accessing this interface. The interface should be utilized only after consulting the source code or Microsoft documentation.

See also: CreateInstance

---

**ShowLoginDialog****property**

property ShowLoginDialog : Boolean

Default: False

↳ Controls whether a log on dialog is displayed when logging on via the Login method.

You must be connected (i.e. Connected must be set to True) to Outlook prior to calling the Login method. If ShowLoginDialog is set to False, calling the Login method transparently logs on to Outlook. If ShowLoginDialog is True, calling the Login method displays a log on dialog box to the user.

See also: Connected, Login, Logoff

# TOpRecipientList Class

The TTopRecipientList class provides an interface to a list of names associated with an Outlook item, (a mail item for example). Add individual recipients to this list using the Add method of this class. Individual recipients may be removed from the list using the Remove method.

The TTopRecipient class encapsulates the functionality of an Outlook recipient. The TTopRecipientList class is a collection of TTopRecipient objects. The TTopRecipientList.RecipientsIntf property provides access to and modification of individual instances of the TTopRecipient class. Many Outlook item classes surface a recipients property of this type, enabling you to gain access to the recipients of an item.

## Hierarchy

TObject (VCL)

    TTopMailListProp (OpOutlk)

        TTopRecipientList (OpOutlk)

## Properties

Count

Items

RecipientsIntf

## Methods

Add

Remove

Create

ResolveAll

## Reference Section

Add	method
-----	--------

function Add(const Name : string) : TTopRecipient;

↳ Creates an instance of TTopRecipient and adds it to the collection of recipients.

Name contains the name of the TTopRecipient being added.

Use this function to add a recipient to the list. The return value of this function is a TOpRecipient whose properties may be set as needed. The following code demonstrates the proper way to add a recipient to the recipient list associated with an e-mail message:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  EMail : TOpMailItem;
begin
  EMail := OpOutlook1.CreateMailItem;
  EMail.Recipients.Add('abcxyz@turbopower.com');
  EMail.Send;
end;
```

See also: Remove, Count, Items

### **Count**

**read-only property**

---

```
property Count : Integer
```

↳ Returns the number of Recipient objects contained in the recipient list.

See also: Add, Remove

### **Create**

**constructor**

---

```
constructor Create(
  AOwner : TOpOutlookItem; Intf : Recipients);
```

↳ Creates a TOpRecipientList that is owned by a TOpOutlook item.

AOwner can be one of the following: TOpAppointmentItem, TOpMailItem, or TOpTaskItem.

Intf is an interface to the Recipients collection newly created.

Direct use of this interface is not recommended. Instead, you should simply access the properties and methods of the TOpRecipientList class. The source code and additional Microsoft documentation should be consulted prior to using this interface.

See also: Add, Create, Count, Items

### **Items**

**property**

---

```
property Items[Index : Integer] : TOpRecipient
```

↳ Returns the specified TOpRecipient object.

Index is a 1-based index of a TOpRecipient object.

See also: Add, Count, Remove

**RecipientsIntf****property**

```
property RecipientsIntf : Recipients
```

- ↳ Returns an IDispatch interface for the TOpRecipientList class.

Most of the functionality of this IDispatch interface has already been encapsulated within the TOpRecipientList class. This interface allows you to expand the functionality of Outlook beyond that which has been encapsulated for you in OfficePartner. It is impractical to document every facet of this interface. You should refer to the source code and any additional information provided by Microsoft (for example, MSDN) prior to using this interface.

See also: Create

**Remove****method**

```
procedure Remove(Index : Integer);
```

- ↳ Removes the recipient referenced by Index from the collection of recipients.

Index is one-based.

See also: Add, Count, Items

**ResolveAll****method**

```
function ResolveAll : Boolean;
```

- ↳ Attempts to resolve all the recipient objects in the collection of recipients against the AddressBook.

This method returns True if all of the objects were resolved and False if one or more were not successfully resolved.

See also: Items

# TOpRecipient Class

The `TOpRecipient` class encapsulates the Outlook Recipient interface, which is a representation of one recipient for a given Outlook item (such as an e-mail message). Typically, you would access instances of this class via a `TOpRecipientList` object.

# Hierarchy

## TObject (VCL)

### TOpMailListItem (OpOutlk)

### TOpRecipient (OpOutlk)

# Properties

Address	Index	Resolved
AddressEntry	Name	TrackingStatus
DisplayType	RecipientIntf	TrackingStatusTime
EntryID	RecipientType	

## Methods

## Create

## Destroy

## Resolve

## Reference Section

**Address** \_\_\_\_\_ **property** \_\_\_\_\_

property Address : string

 Specifies the messaging address of an address entry or message recipient.

See also: AddressEntry

**AddressEntry****property**

```
property AddressEntry : AddressEntry
```

- ↳ Defines addressing information valid for a given messaging system.

An address usually represents a person or process to which the messaging system can deliver messages. AddressEntry is an interface that can access various methods and properties of the recipient class. This interface can extend the functionality of OfficePartner beyond that which it already encapsulates. You should consult the source code and any additional information from Microsoft (for example, MSDN) prior to using this interface.

See also: Address

**Create****constructor**

```
constructor Create(  
    AOwner : TOpRecipientList; Intf : Recipient);
```

- ↳ Creates an instance of the TOpRecipient class.

AOwner is the TOpRecipientList object that the newly created TOpRecipient is added to. Intf contains an interface to the newly created TOpRecipient.

Instances of the TOpRecipient class are created automatically when you call the parent RecipientList.Add method.

See also: Destroy, Resolve

**Destroy****destructor**

```
destructor Destroy;
```

- ↳ Releases the memory held by the Recipient object.

Destroy is called automatically when the parent RecipientList is destroyed.

See also: Create

<b>DisplayType</b>	<b>property</b>
--------------------	-----------------

```
property DisplayType : TOpOldDisplayType
TOpOldDisplayType = (
    olUser, olDistList, olForum, olAgent, olOrganization,
    olSensPrivateDistList, olRemoteUser);
```

↳ Defines the display type of the address entry.

The **DisplayType** property enables special processing based on its value, such as displaying an associated icon. You can also use the display type to sort or filter address entries.

These types are defined as follows:

**Table 5:**

<b>Value</b>	<b>Meaning</b>
olUser	Local messaging user
olDistList	Public distribution list
olForum	Forum, such as a bulletin board or a public folder
olAgent	Automated agent, such as Quote-of-the-Day
olOrganization	Special address entry defined for large groups, such as a helpdesk
olPrivateDistList	Private, personally administered distribution list
olRemoteUser	Messaging user in a remote messaging system

See also: [RecipientType](#)

<b>EntryID</b>	<b>read-only property</b>
----------------	---------------------------

```
property EntryID : string
```

↳ Returns the unique identifier of the AddressEntry object as a string.

The **EntryID** is automatically assigned whenever a recipient is saved or is moved to a different folder.

See also: [Index](#)

```
property Index : Integer
```

- >Returns the index number of the TOpRecipient object within the Recipients collection.

The Index property indicates this object's position within the parent Recipients collection. It can be saved and used later with the collection's Item property to reselect the same recipient in the collection.

4

The first object in the collection has an Index value of 1.

An index value should not be considered constant for the duration of a session. It can be affected when other recipients are added and deleted. The index value is changed following an update to the Outlook item associated with the TOpRecipient object.

See also: EntryID

---

Name	property
------	----------

---

```
property Name : string
```

- >Returns or sets the name of the recipient object as a string.

This property is passed as a parameter to the Recipients.Add function and is not generally set directly.

See also: Address, EntryID, Index

---

RecipientIntf	property
---------------	----------

---

```
property RecipientIntf : Recipient
```

- >Returns an interface pointer to the recipient object.

The Recipient interface exposes additional functionality of the TOpRecipient object. This interface extends the functionality beyond what has already been encapsulated for you within OfficePartner. You should consult the source code and any additional information from Microsoft (for example, MSDN) prior to using this interface.

See also: AddressEntry

---

**RecipientType** property

```
property RecipientType : TOpOlMailRecipientType
TOpOlMailRecipientType = (
    olmrtOriginator, olmrtTo, olmrtCC, olmrtBCC);
```

Default: olmrtOriginator

↳ Specifies the recipient type of the Recipient object.

The following table describes the possible values:

**Table 6:**

Value	Description
olOriginator	Originator recipient
olTo	To recipient
olCC	CC recipient
olBCC	BCC recipient

See also: Recipient, DisplayType, RecipientIntf

---

**Resolve** method

```
function Resolve : Boolean;
```

↳ Resolves a recipient's address information into a full messaging address.

The Resolve method validates the Recipient object's [Type](#) property and returns False if it is not one of the defined recipient types.

You can call the [Recipients](#) collection's [Resolve](#) method to resolve every object in the collection and force an update to the collection's [Count](#) property.

The Resolve method uses the address book or books specified in the profile, such as the global address list (GAL) and the personal address book (PAB).

See also: Resolved, RecipientType

## Resolved

## property

property Resolved : Boolean

Default: False

- ↳ Determines whether a recipient's address information has been resolved to a full messaging address.

4

A recipient is considered resolved when a matching address is found in the address list or the user has selected an address from a dialog box.

When you supply the Name property, Resolve looks it up in the address book. When a recipient is resolved, the Recipient object's Address property is set to the full address and its AddressEntry property is set to the child AddressEntry object that represents a copy of information in the address book.

When you specify a custom address by supplying only the Recipient object's Address property, the Resolve method does not attempt to compare the address against the address book.

To avoid delivery errors, clients should always ensure that recipients are resolved recipients before submitting a message to the MAPI system.

The Recipients collection's Resolved property is set to True when every recipient in the collection has its address resolved.

See also: Address, AddressEntry, Name, Resolve

**TrackingStatus****property**

```
property TrackingStatus : TOpOlTrackingStatus
TOpOlTrackingStatus = (
    oltsNone, oltsDelivered,
    oltsNotDelivered, oltsNotRead,
    oltsRecallFailure, oltsRecallSuccess,
    oltsRead, oltsReplied);
```

Default: oltsNone

↳ Defines the tracking status of the recipient.

TrackingStatus can be set to any of the following values:

**Table 7:**

<b>Value</b>	<b>Description</b>
olTrackingNone	Not tracked
olTrackingDelivered	Delivered
olTrackingNotDelivered	Not delivered
olTrackingNotRead	Unread
olTrackingRecallFailure	Recall failure
olTrackingRecallSuccess	Recall success
olTrackingRead	Read
olTrackingReplied	Read and replied

See also: TrackingStatusTime

**TrackingStatusTime****property**

```
property TrackingStatusTime : TDateTime
```

↳ Returns or sets the tracking status date and time for the recipient.

See also: TrackingStatus

---

# TOpAttachmentList Class

TOpAttachmentList encapsulates the Outlook Attachments interface. Attachments in Outlook can be separate files, or additional Outlook items such as mail items, appointment items, etc. These Outlook item classes surface an Attachments property of this type, enabling you to gain access to the attachments for a given item.

Attachments are added to this list by calling the Add method. Attachments are deleted from this list by calling the Remove method. The Count property maintains at all times an accurate count of the attachments in the collection. Individual attachments are accessed via the Items property.

Additionally, the TOpAttachmentList class provides an interface property AttachmentsIntf. This property allows you to exploit the functionality of the TOpAttachmentList class beyond that which OfficePartner exposes.

## Hierarchy

TObject (VCL)

TOpMailListProp (OpOutlk)

TOpAttachmentList (OpOutlk)

## Properties

AttachmentsIntf

Count

Items

## Methods

Add

Create

Remove

## Reference Section

### Add

### method

function Add(const FileName: string): TOpAttachment;

Creates an instance of TOpAttachment and adds it to the collection of attachments.

FileName contains the disk file name of the attachment being added.

The return value of this function is a newly created TOpAttachment instance whose properties can be set as needed.

The following code provides an example of how to use this function:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  EMail : TOpMailItem;
begin
  OpOutlook1.Connected := True;
  EMail := OpOutlook1.CreateMailItem;
  EMail.Attachments.Add('Somefile.txt');
  ...
end;
```

See also: Remove, Count, Items

---

<b>AttachmentsIntf</b>	<b>property</b>
------------------------	-----------------

---

**property AttachmentIntf : Attachments**

↳ Returns an automation interface for the TOpAttachmentList class.

Most of the functionality of this interface has already been encapsulated within the TOpAttachmentList class. This interface provides the properties and methods that allow you to expose more functionality than what has been encapsulated by OfficePartner.

It is recommended that you consult the OfficePartner source code and any additional information from Microsoft (such as MSDN) prior to using this interface.

See also: Add, Create

---

<b>Count</b>	<b>read-only property</b>
--------------	---------------------------

---

**property Count : Integer**

↳ Returns the number of Attachment objects contained in the attachment list.

This property is automatically updated as attachments are added or removed from the attachment list.

See also: Add, Items, Remove

**Create****method**


---

```
constructor Create(
  AOwner : TOpOutlookItem; Intf : Attachments);
```

↳ Creates a TOpAttachmentList that is owned by a TOpOutlook item.

AOwner can be any descendant of TOpOutlookItem. (eg TOpAppointmentItem, TOpMailItem, or TOpTaskItem).

Intf is an interface to the Attachments collection newly created.

The Intf interface can be used to extend the functionality of the attachment beyond what OfficePartner already encapsulates for you. You should consult the OfficePartner source code and any additional Microsoft information (such as MSDN) prior to using this interface.

See also: Add, Count, Items, Remove

**Items****property**


---

```
property Items[Index : Integer] : TOpAttachment
```

↳ Returns the TOpAttachment object whose position in the Attachment list is defined by Index.

Index is 1-based.

The item read when accessing this property is a TOpAttachment instance whose properties can be set as needed.

See also: Add, Count, Remove

**Remove****method**


---

```
procedure Remove(Index : Integer);
```

↳ Removes the attachment referenced by Index from the collection of attachments.

The index into the AttachmentList is 1-based.

See also: Add, Count, Items

# TOpAttachment Class

TOpAttachment encapsulates the Outlook Attachment interface. The Outlook Attachment interface represents disk files that have been attached to Outlook mail items or other Outlook items attached to mail items. Typically, you would access instances of this class via a TOpAttachmentList object.

## Hierarchy

TObject (VCL)

TOpMailListItem (OpOutlk)

TOpAttachment (OpOutlk)

## Properties

AttachmentIntf	FileName
DisplayName	PathName

## Methods

Create	Destroy
--------	---------

## Reference Section

AttachmentIntf	property
----------------	----------

`property AttachmentIntf : Attachment`

↳ Returns an interface pointer to the Attachment IDispatch interface.

Most of the functionality Oof the Attachment interface has already been encapsulated within the TOpAttachment class. This interface allows you to expand on the functionality already encapsulated for you within the OfficePartner framework. It is recommended that you consult the OfficePartner source code and any additional information from Microsoft (such as MSDN) prior to utilizing the properties and methods of this interface.

See also: Create

---

**Create****constructor**

---

```
constructor Create(  
  AOwner : TOpAttachmentList; Intf : Attachment);
```

↳ Creates an instance of the TOpAttachment class.

AOwner defines the TOpAttachmentList object that owns the newly created TOpAttachment.

Intf contains an interface to the newly created TOpAttachment.

This method is called internally and should not be called directly. Rather, Attachments should be created as in the following example:

```
procedure TForm1.Button1Click(Sender : TObject);  
var  
  mi : TOpMailItem;  
begin  
  mi := OpOutlook1.CreateMailItem;  
  mi.Attachments.Add('test.txt');  
end;
```

The above code implicitly calls this constructor for you.

See also: Destroy

---

**Destroy****destructor**

---

```
destructor Destroy;
```

↳ Destroys this instance of TOpAttachment and removes it from the collection of attachments.

Automatic destruction of Attachment items is provided for you. Do not call this method directly.

See also: Create

---

**DisplayName** property

```
property DisplayName : string
```

↳ Returns or sets the name displayed below the icon representing the embedded attachment.

This property may differ from the actual filename of the attachment.

See also: [FileName](#), [PathName](#)

---

**FileName** property

```
property FileName : string
```

↳ Returns the file name of the attachment.

Use this property in conjunction with the [PathName](#) property to determine the fully qualified path of the attachment.

See also: [DisplayName](#), [PathName](#)

---

**PathName** property

```
property PathName : string
```

↳ Returns the full path to the linked attached file.

This property is only valid for linked files.

See also: [DisplayName](#), [FileName](#)

# TOpAppointmentItem Class

The TOpAppointmentItem class represents an appointment in the Calendar folder. An AppointmentItem object can represent a meeting, a one-time appointment, or a recurring appointment or meeting.

TOpAppointmentItem encapsulates the Outlook \_AppointmentItem interface. You most commonly create an instance of this class using the TOpOutlook.CreateAppointmentItem method. This class contains numerous properties that correspond to the various fields found in Outlook's appointment item inspector window.

## Hierarchy

TObject (VCL)

TOpOutlookItem (OpOutlk)

TOpAppointmentItem (OpOutlk)

## Properties

AllDayEvent	IsRecurring	RecurrenceState
AppointmentItem	ItemInspector	ReminderMinutesBeforeS...
Attachments	LastModificationTime	ReminderOverrideDefault
BillingInformation	Location	ReminderPlaySound
Body	MeetingStatus	ReminderSet
BusyStatus	MessageClass	ReminderSoundFile
Categories	Mileage	ReplyTime
Companies	NetMeetingAutoStart	RequiredAttendees
ConversationIndex	NetMeetingOrganizerAlias	Resources
ConversationTopic	NetMeetingServer	ResponseRequested
CreationTime	NetMeetingType	ResponseStatus
Duration	NoAging	Saved
End_	OptionalAttendees	Sensitivity
EntryID	Organizer	Size
FormDescription	OutlookInternalVersion	Start
Importance	OutlookVersion	Subject
IsOnlineMeeting	Recipients	UnRead

---

## Methods

ClearRecurrencePattern	Destroy	Respond
Close	Display	Save
Copy	ForwardAsVcal	SaveAs
Create	GetRecurrencePattern	Send
Delete	PrintOut	

## Reference Section

---

### AllDayEvent

property

property AllDayEvent : Boolean

Default: False

↳ Indicates whether this appointment is an all-day or multiple-day event.

The AllDayEvent property contains True if the appointment takes up one or more entire days in the calendar without any free blocks. It contains False otherwise.

Changes to this property do not take effect until you call the Send method.

See also: Send

---

### AppointmentItem

property

property AppointmentItem : \_AppointmentItem

↳ Interface to the methods and properties of Outlook's appointment item interface.

OfficePartner encapsulates most of the functionality of this interface. This property allows you to expand the functionality beyond the encapsulation that OfficePartner provides. It is recommended that you consult the OfficePartner source code and any additional information provided by Microsoft (for example, MSDN) prior to using this interface.

See also: Create

---

### Attachments

property

property Attachments : TOpAttachmentList

↳ Maintains a list of attachments associated with the appointment item.

You should add, remove, and otherwise access the properties of individual attachments by indexing into this property.

For additional information, see “[TOpAttachmentList](#)” on page 170.

See also: [AppointmentItem](#)

---

### **BillingInformation** property

`property BillingInformation : string`

↳ Sets or returns a string containing the billing information associated with the appointment item.

Based on user preferences, this field may be hidden in Outlook’s Appointment inspector window.

---

### **Body** property

`property Body : string`

↳ Sets or returns the body of the message associated with this appointment item.

This property corresponds to the text that appears in the main memo of Outlook’s Appointment inspector window.

See also: [Subject](#)

---

### **BusyStatus** property

`property BusyStatus : TOpOlBusyStatus`

`TOpOlBusyStatus = (`  
    `olbsFree, olbsTentative, olbsBusy, olbsOutOfOffice);`

Default: `olbsFree`

↳ Returns or sets the busy status of this messaging user for this appointment.

If two or more conflicting commitments are present during the time span of this appointment, the highest applicable value is returned, representing the most committed state. For example, if the user has one tentative commitment and one confirmed commitment during the time span, the `BusyStatus` property returns `olbsBusy`, that is, the highest level of commitment.

Changes to properties on an `AppointmentItem` object take effect when you call the `Send` method.

See also: [ResponseStatus](#)

---

**Categories** property

```
property Categories : string
```

↳ Specifies the categories assigned to the message.

A category is a keyword or phrase that allows you to find, sort, filter, and group Outlook items. For example, contact items can be categorized as “Business” or “Personal”. Outlook provides a list of general categories, but this list can be modified as necessary.

The Categories property is defined by the application. The Categories property is commonly used to hold a set of keywords that can be used to access messages in a folder.

See also: AppointmentItem

---

**ClearRecurrencePattern** method

```
procedure ClearRecurrencePattern;
```

↳ Removes any recurrence settings from this appointment.

The ClearRecurrencePattern method sets the IsRecurring property to False and disassociates this AppointmentItem object from any recurrence pattern previously assigned to it.

See also: IsRecurring

---

**Close** **method**


---

```
procedure Close(SaveMode : TOpO1InspectorClose);
TOpO1InspectorClose = (olicSave, olicDiscard, olicPromptForSave);
Default: olicPromptForSave
```

- ↳ Closes the appointment item.

SaveMode determines whether the appointment is saved upon closing. The following table defines the possible values:

**Table 8:**

<b>Value</b>	<b>Description</b>
olImportanceLow	Lowest Importance
olImportanceNormal	Normal Importance
olImportanceHigh	Highest Importance

See also: Save, SaveAs, Saved

---

**Companies** **property**


---

```
property Companies : string
```

- ↳ Sets or returns a semicolon delimited string describing how the companies are associated with the appointment item.

---

**ConversationIndex** **property**


---

```
property ConversationIndex : string
```

- ↳ Specifies the index to the conversation thread of the message.

The ConversationIndex property is a string that represents a hexadecimal number. Valid characters within the string include the numbers 0 through 9 and the letters A through F (uppercase or lowercase).

See also: ConversationTopic

**ConversationTopic****property**


---

```
property ConversationTopic : string
```

↳ Specifies the subject of the conversation thread of the message.

A conversation is a group of related messages. The ConversationTopic property is the string that describes the overall topic of the conversation. To be considered as messages within the same conversation, the messages must have the same value in their ConversationTopic property.

The ConversationIndex property represents an index that indicates a sequence of messages within that conversation.

See also: ConversationIndex

**Copy****method**


---

```
function Copy : TOpAppointmentItem;
```

↳ Creates a duplicate of the appointment item and returns a reference to it.

See also: Create, Delete

**Create****method**


---

```
constructor Create(APptItem : _AppointmentItem);
```

↳ Creates an instance of TOpAppointmentItem and associated collections.

This method creates an attachments collection and a recipients collection and associates these collections with the appointment item.

You should not use this method to create an AppointmentItem. Instead, you should use the CreateAppointmentItem function of the TOpOutlook component as follows:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ai : TOpAppointmentItem;
begin
  ai := OpOutlook1.CreateAppointmentItem;
  ai.{methods..}
  ...
end;
```

See also: Destroy, Save, SaveAs

**CreationTime****property**


---

```
property CreationTime : TDateTime
```

- ↳ Specifies the date and time the message was first saved.

See also: End\_, Start

**Delete****method**


---

```
procedure Delete;
```

- ↳ Removes an AppointmentItem from the Appointments collection.

Before calling Delete, your application should prompt the user to verify whether the message should be permanently deleted.

When you delete a member of a collection, the collection is immediately refreshed. That is, its Count property is reduced by one and its members are re-indexed. To access a member following the deleted member, you must use its new index value.

See also: Close, EntryID, Save, SaveAs

**Destroy****destructor**


---

```
destructor Destroy;
```

- ↳ Destroys this instance of the appointment item.

This method is called automatically when the parent collection is cleared. It should not be called directly.

See also: Close, Create, Delete

**Display****method**


---

```
procedure Display(Modal : Boolean);
```

- ↳ Displays Outlook's Appointment inspector window depicting the details of the appointment item.

Modal determines the window's modality. If Modal is True, the window is displayed modally. Otherwise, the window is displayed non-modally.

This is Outlook's default user-interface window for editing appointment items.

See also: ItemInspector

---

**Duration** property

```
property Duration : Integer
```

Default: 30

↳ Returns the duration of this appointment in minutes.

The Duration property contains the number of minutes the appointment is to last. The minimum value is 0 and the maximum value is 1,490,000, which is treated as infinity.

See also: CreationTime, End\_, Start

---

**End\_** property

```
property End_ : TDateTime
```

↳ Returns or sets the ending date and time of this appointment.

The End\_ property defaults to the current time rounded up to the next half hour. The End\_ property ignores seconds and truncates the time to the minute.

The value of End\_ must be greater than that of the Start property.

See also: Start, CreationTime

---

**EntryID** property

```
property EntryID : string
```

↳ Returns the unique identifier of the Message object as a string.

The EntryID property can be used to retrieve this message later.

See also: EntryID

---

**FormDescription** property

```
property FormDescription : FormDescription
```

↳ Contains an interface to the general properties of an Outlook form.

The properties of an Outlook form are displayed on the Properties page of a form in design mode.

See also: ItemInspector

**ForwardAsVcal****method**

```
function ForwardAsVcal : TOpMailItem;
```

- ↳ Forwards the AppointmentItem as a virtual calendar (\*.vcs) item.

Virtual calendar files exchange information about appointments and schedules with other users.

The ForwardAsVcal method returns a MailItem with the virtual calendar file attached.

Refer to Microsoft's documentation for further details.

See also: Send

**GetRecurrencePattern****method**

```
function GetRecurrencePattern : RecurrencePattern;
```

- ↳ Returns a RecurrencePattern interface defining the recurrence attributes of an appointment.

If the appointment was previously non-recurring, calling this method makes it recurring and returns a new default RecurrencePattern interface. You can then use this interface to define the properties and methods of the recurrence pattern.

Note: You cannot use GetRecurrencePattern to test whether an appointment is recurring, because it always returns a RecurrencePattern object and forces the appointment to be recurring. Instead, you should test for recurrence using the IsRecurring property.

You can use the ClearRecurrencePattern method to return the appointment to non-recurring status.

See also: ClearRecurrencePattern, IsRecurring

---

<b>Importance</b>	<b>property</b>
-------------------	-----------------

---

```
property Importance : TOp01Importance
TOp01Importance = (olImpLow, olImpNormal, olImpHigh);
Default: olImpNormal
```

↳ Returns or sets the importance of the message.

The possible values are defined in the following table:

**Table 9:**

Value	Description
olImportanceLow	Lowest Importance
olImportanceNormal	Normal Importance
olImportanceHigh	Highest Importance

---



---

<b>IsOnlineMeeting</b>	<b>property</b>
------------------------	-----------------

---

```
property IsOnlineMeeting : Boolean
```

Default: False

↳ Indicates whether the appointment is to be conducted online.

If IsOnlineMeeting is True then the meeting is an online meeting. Otherwise, the meeting is not online.

See also: NetMeetingAutoStart, NetMeetingOrganizerAlias, NetMeetingServer, NetMeetingType

---

<b>IsRecurring</b>	<b>read-only property</b>
--------------------	---------------------------

---

```
property IsRecurring : Boolean
```

Default: False

↳ Indicates whether this appointment is specified as recurring.

The IsRecurring property contains True if the appointment is recurring and False if it is not. IsRecurring defaults to False in a newly created AppointmentItem object.

IsRecurring is set to True when the GetRecurrencePattern method is called and to False when the ClearRecurrencePattern method is called.

See also: GetRecurrencePattern

---

<b>ItemInspector</b>	<b>property</b>
----------------------	-----------------

---

**property ItemInspector : Inspector**

↳ Displays the window in which the appointment item is displayed.

You must call the Display method of this interface to show the inspector window as shown in the following code:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  mi : TOpAppointmentItem;
begin
  mi := OpOutlook1.CreateAppointmentItem;
  mi.ItemInspector.Display(False); {not modal}
  ...
end;
```

See also: FormDescription

---

<b>LastModificationTime</b>	<b>property</b>
-----------------------------	-----------------

---

**property LastModificationTime : TDateTime**

↳ Returns the date and time that the appointment item was last modified.

See also: CreationTime

---

<b>Location</b>	<b>property</b>
-----------------	-----------------

---

**property Location : string**

↳ Returns or sets the location of this appointment.

The Location property contains a string representing the specific location in which this appointment is scheduled, such as an office or conference room. It is typically set by the user that initiates a meeting. The initiating user is available through the Organizer property.

Changes you make to properties on an AppointmentItem object take effect when you call the Send method.

See also: Organizer, Send

MeetingStatus	property
---------------	----------

```
property MeetingStatus : TOpOlMeetingStatus  
TOpOlMeetingStatus = (  
    olmsNonMeeting, olmsMeeting,  
    olmsMeetingReceived, olmsMeetingCancelled);
```

Default: olmsNonMeeting

↳ Returns or sets the overall meeting status of this appointment.

The MeetingStatus property is used to indicate whether this appointment represents a meeting, and if so, what its status is. MeetingStatus can be assigned any one of the following values:

**Table 10:**

Value	Appointment item type
olNonMeeting	Not a meeting
olMeeting	A normal meeting
olMeetingReceived	Received
olMeetingCancelled	Cancelled

Only the organizer of a meeting can cancel or release it.

Changes you make to properties on an AppointmentItem object take effect when you call the Send method.

See also: Send

---

**MessageClass** **property**

```
property MessageClass : string
```

- ↳ Sets or returns a string defining the message class of the appointment item.

A message class is a keyword or phrase that allows you to find, sort, filter, and group appointment items.

---

**Mileage** **property**

```
property Mileage : string
```

- ↳ Sets or returns a string defining the distance to the appointment.

---

**NetMeetingAutoStart** **property**

```
property NetMeetingAutoStart : Boolean
```

Default: False

- ↳ Determines whether the meeting starts automatically if it is an online meeting.

If the meeting is not an online meeting, this property has no effect.

See also: IsOnlineMeeting

---

**NetMeetingOrganizerAlias** **property**

```
property NetMeetingOrganizerAlias : string
```

- ↳ Sets or returns a string describing the computer alias name of the person who organized the appointment.

See also: Organizer, IsOnLineMeeting

---

**NetMeetingServer** **property**

```
property NetMeetingServer : string
```

- ↳ Sets or returns the name of the host computer server if the appointment is online.

This property is ignored if the appointment is not online.

See also: IsOnlineMeeting

---

**NetMeetingType** property

```
property NetMeetingType : TOpOlNetMeetingType
TOpOlNetMeetingType = (OlChat, olnmtNetMeeting, olnmtNetShow);
Default: olnmtNetMeeting
```

↳ Returns or sets the type of online meeting.

The property corresponds to the NetMeeting type drop-down list box on the Online page of an online meeting item. The following are the OlNetMeetingType constants:

**Table 11:**

Value	Description
olChat	Interactive chat meeting
olNetMeeting	Normal network meeting
olNetShow	Network presentation

Refer to Microsoft's documentation for more information regarding meeting types.

See also: IsOnLineMeeting

---

**NoAging** property

```
property NoAging : Boolean
```

Default: False

↳ Indicates whether the appointment item is aged.

Outlook users may take action on an item based on its age. For instance, e-mail messages may be deleted or archived once they reach a certain age.

If NoAging is set to True, then the appointment item is not aged.

See also: CreationTime, Duration

---

**OptionalAttendees** **property**

```
property OptionalAttendees : string
```

- ↳ Sets or returns a string that lists the optional attendees for the appointment.

This property is a semicolon-delimited string of attendees.

See also: RequiredAttendees

---

**Organizer** **property**

```
property Organizer : string
```

- ↳ Sets or returns the name of the person responsible for organizing the appointment.

See also: NetMeetingOrganizerAlias

---

**OutlookInternalVersion** **property**

```
property OutlookInternalVersion : Integer
```

- ↳ Returns an integer corresponding to the build number of the Outlook application to which this appointment item applies.

See also: OutlookVersion

---

**OutlookVersion** **property**

```
property OutlookVersion : string
```

- ↳ Returns a string containing the major version number of the Outlook application to which this appointment item applies.

See also: OutlookInternalVersion

---

**PrintOut** **method**

```
procedure PrintOut;
```

- ↳ Immediately sends the details of the appointment item to the default printer.

See also: Save, SaveAs

---

**Recipients** property

```
property Recipients : TOpRecipientList
```

>Returns the TOpRecipientList collection associated with this appointment item.

The TOpRecipientList is a collection of one or more TOpRecipient objects. Use this property to iterate through and gain access to the individual TOpRecipient objects contained in this collection.

See also: RequiredAttendees

---

**RecurrenceState** property

```
property RecurrenceState : TOpOlRecurrenceState
TOpOlRecurrenceState = (
    olrsApptNotRecurring, olrsApptMaster,
    olrsApptOccurrence, olrsApptException);
```

Default: olrsApptNotRecurring

↳ Determines how Outlook handles recurrence of this appointment item.

Refer to Microsoft's documentation for further information regarding recurrence states.

See also: IsRecurring

---

**ReminderMinutesBeforeStart** property

```
property ReminderMinutesBeforeStart : Integer
```

Default: 15

↳ Indicates how many minutes before the start of this appointment a reminder should be issued.

The ReminderMinutesBeforeStart property must contain a positive integer. It is not enabled unless the ReminderSet property is True.

Changes you make to properties on an AppointmentItem object take effect when you call the Send method.

See also: Send

**ReminderOverrideDefault****property**

```
property ReminderOverrideDefault : Boolean
```

Default: False

- ↳ Determines whether the reminder overrides the default reminder behavior for this appointment item.

This property is True if the reminder overrides the default reminder behavior for the appointment. Otherwise, ReminderOverrideDefault is False.

See also: [ReminderPlaySound](#), [ReminderSet](#), [ReminderSoundFile](#)

**ReminderPlaySound****property**

```
property ReminderPlaySound : Boolean
```

Default: False

- ↳ Determines whether an audio file is played when the reminder for this appointment item occurs.

Set this property to True and supply a file name in the ReminderSoundFile property to play the sound file when this reminder is due. If ReminderPlaySound is False or ReminderSoundFile does not contain a valid sound (e.g.: \*.wav) file, then no sound plays when the reminder is due.

See also: [ReminderSet](#), [ReminderSoundFile](#)

**ReminderSet****property**

```
property ReminderSet : Boolean
```

Default: True

- ↳ Indicates whether this messaging user is to be reminded of this appointment.

The ReminderSet property contains True if a reminder has been set for this appointment and False otherwise.

The ReminderMinutesBeforeStart property is not enabled unless the ReminderSet property is True.

See also: ReminderMinutesBeforeStart

### **ReminderSoundFile**

**property**

**property** ReminderSoundFile : string

↳ Sets or returns the file name of an audio file that is to be played when the reminder for this appointment occurs.

The file type can be any audio file type registered to Windows.

See also: ReminderMinutesBeforeStart, ReminderPlaySound

### **ReplyTime**

**property**

**property** ReplyTime : TDateTime

↳ Returns or sets the date and time a recipient replies to the meeting request associated with this appointment.

The ReplyTime property is not meaningful on the original `AppointmentItem object held in the calendar folder of the meeting's Organizer. Instead, ReplyTime is used on the AppointmentItem object created in the calendar folder of each meeting request recipient.

See also: Send

### **RequiredAttendees**

**property**

**property** RequiredAttendees : string

↳ Sets or returns a string listing all of the required attendees for this appointment.

If the string returned by the RequiredAttendees property contains more than one item, it is a semi-colon delimited string of attendees.

See also: OptionalAttendees

## Resources property

---

`property Resources : string`

- ↳ Sets or returns a string describing the resources required to conduct this appointment.

## Respond method

---

```
function Respond(
    Response : TOpOlMeetingResponse;
    NoUI, AdditionalTextDialog : Boolean) : MeetingItem;

TOpOlMeetingResponse = (
    olmsMeetingolmrTentative, olmsMeetingolmrAccepted,
    olmsMeetingolmrDeclined);
```

- ↳ Prepares a meeting response.

The returned `MeetingItem` provides access to virtually all properties associated with the appointment item.

Note: The only `AppointmentItem` objects on which you can call the `Respond` method are those returned by the `GetAssociatedAppointment` method of a `MeetingItem` object.

The `Respond` method prepares a meeting response, which can be sent in answer to a meeting request using the `Send` method.

If you respond to a meeting request with `olmsMeetingolmrDeclined`, no `AppointmentItem` object is created, but any `AppointmentItem` already in the folder is left undeleted. Therefore, if you accept a request and subsequently decline it, you must either delete the associated `AppointmentItem` object yourself or leave it in the folder.

If you have declined a meeting request and subsequently wish to accept it, you cannot use any associated `AppointmentItem` object for your new response. However, if you have retained the requesting `MeetingItem` object, you can use its `Respond` method to accept the request.

See also: `ResponseRequested`, `ResponseStatus`

---

**ResponseRequested** property

```
property ResponseRequested : Boolean
```

Default: True

↳ Indicates whether a response is requested for this appointment.

If the meeting organizer wishes a response, this value is True.

See also: Organizer, ResponseStatus, Send

---

**ResponseStatus** property

```
property ResponseStatus : T0lResponseStatus
```

```
T0lResponseStatus = (  
    olrespstNone, olrespstOrganized,  
    olrespstTentative, olrespstAccepted,  
    olrespstDeclined, olrespstNotResponded);
```

Default: olrespstNone

↳ Indicates the response received from attendee(s) regarding this appointment.

Refer to Microsoft's documentation for further explanations.

See also: ResponseRequested

---

**Save** method

```
procedure Save;
```

↳ Saves the Outlook item to the current folder or, if this is a new item, to the Outlook default folder for the item type.

See also: SaveAs

## SaveAs

## method

```
procedure SaveAs(
  const Path : string; SaveAsType : TOpOlSaveAsType);
```

TOpOlSaveAsType = (  
 olsatDoc, olsatHTML, olsatMSG, olsatRTF, olsatTemplate, olsatTXT,  
 olsatVCal, olsatVCard);

4

↳ Saves the Outlook item to the specified path and in the format of the specified file type.

If the file type is not specified, the MSG (message) format is used. Path defines the path on which to save the item. SaveAsType determines the file format in which to save the item.

TOpOlSaveAsType is defined as follows:

**Table 12:**

<b>TOlSaveAsType</b>	<b>Description</b>
olDoc	Document file
olHTML	HTML file
olMSG	Message file
olRTF	Rich text file
olTemplate	Template
olTXT	Text file
olVCal	Virtual calendar item
olvCard	Virtual card item

See also: Save

## Saved

## property

```
property Saved : Boolean
```

Default: False

↳ Indicates whether this appointment has changed since it was last saved.

This property is True if the appointment has not changed since it was last saved. Otherwise, it is False.

See also: Save, SaveAs

**Send****method**


---

```
procedure Send;
```

- ↳ Sends the message to the recipients through the MAPI system.

The Send method saves all changes to the message in the MAPI system. It also moves the message to the current user's Outbox folder.

Messaging systems retrieve messages from the Outbox and transports them to the recipients. After it is transported, the message is removed from the Outbox and deleted.

See also: Respond

**Sensitivity****property**


---

```
property Sensitivity : TOpOlSensitivity
TOpOlSensitivity = (
    olSensNormal, olSensPersonal, olSensPrivate, olSensConfidential);
```

Default: olSensNormal

- ↳ Specifies the sensitivity of the message.

Sensitivity can be set to any of the following:

**Table 13:**

<b>Value</b>	<b>Description</b>
olNormal	No sensitive content
olPersonal	Sensitive
olPrivate	More sensitive
olConfidential	Highest sensitivity

**Size****read-only property**


---

```
property Size : Integer
```

- ↳ Returns the approximate size in bytes of the message.

The Size property contains the sum, in bytes, of the sizes of all properties on this Message object, including the Attachments property. It can be considerably greater than the size of the Text property alone.

The Size property is not computed until after the first Send operation.

See also: Send

---

**Start** **property**

`property Start : TDateTime`

↳ Sets or returns the date and time that the appointment is scheduled to begin.

See also: Duration

---

**Subject** **property**

`property Subject : string`

↳ The Subject property returns or sets the subject of the message as a string.

In a conversation thread, the Subject property is often used to set the ConversationTopic property.

See also: Body, ConversationTopic

---

**UnRead** **property**

`property UnRead : Boolean`

Default: True

↳ Determines whether the message has been read by the current user.

---

# TOpContactItem Class

The TOpContactItem class represents a contact in the Contacts folder.

TOpContactItem encapsulates the Outlook \_ContactItem interface. You will most commonly create an instance of this class using the TOpOutlook.CreateContactItem method. This class contains numerous properties that correspond to the various fields found in Outlook's contact item inspector window.

## Hierarchy

TObject (VCL)

TOpOutlookItem (OpOutlk)

TOpContactItem (OpOutlk)

## Properties

Account	Children	Email3Address
Anniversary	Companies	Email3AddressType
AssistantName	CompanyAndFullName	Email3DisplayName
AssistantTelephoneNumber	CompanyLastFirstNoSpace	Email3EntryID
BillingInformation	CompanyLastFirstSpace...	EntryID
Birthday	CompanyMainTelephone...	FileAs
Business2TelephoneNumber...	CompanyName	FirstName
BusinessAddress	ComputerNetworkName	FTPSite
BusinessAddressCity	ContactItem	FullName
BusinessAddressCountry	CreationTime	FullNameAndCompany
BusinessAddressPostalCo...	CustomerID	Gender
BusinessAddressPostOffi...	Department	GovernmentIDNumber
BusinessAddressState	Email1Address	Hobby
BusinessAddressStreet	Email1AddressType	Home2TelephoneNumber
BusinessFaxNumber	Email1DisplayName	HomeAddress
BusinessHomePage	Email1EntryID	HomeAddressCity
BusinessTelephoneNumber	Email2Address	HomeAddressCountry
CallbackTelephoneNumber	Email2AddressType	HomeAddressPostalCode
CarTelephoneNumber	Email2DisplayName	HomeAddressPostOffice...
Categories	Email2EntryID	HomeAddressState

HomeAddressStreet	MailingAddressPostOffic...	PrimaryTelephoneNumber
HomeFaxNumber	MailingAddressState	Profession
HomeTelephoneNumber	MailingAddressStreet	RadioTelephoneNumber
Importance	ManagerName	ReferredBy
Initials	MiddleName	Saved
InternetFreeBusyAddress	MobileTelephoneNumber	SelectedMailingAddress
ISDNNumber	NetMeetingAlias	Spouse
ItemInspector	NetMeetingServer	Suffix
JobTitle	NickName	TelexNumber
Journal	NoAging	Title
Language	OfficeLocation	TTYTDDTelephoneNum...
LastFirstAndSuffix	OrganizationalIDNumber	User1
LastFirstNoSpace	OtherAddress	User2
LastFirstNoSpaceCompany	OtherAddressCity	User3
LastFirstSpaceOnly	OtherAddressCountry	User4
LastFirstSpaceOnlyComp...	OtherAddressPostalCode	UserCertificate
LastModificationTime	OtherAddressPostOffice...	UserProperties
LastName	OtherAddressState	WebPage
LastNameAndFirstName	OtherAddressStreet	YomiCompanyName
MailingAddress	OtherFaxNumber	YomiFirstName
MailingAddressCity	OtherTelephoneNumber	YomiLastName
MailingAddressCountry	PagerNumber	
MailingAddressPostalCode	PersonalHomePage	

## Methods

Close	Display	Save
Copy	ForwardAsVcard	SaveAs
Create	Move	
Delete	PrintOut	

## Reference Section

<b>Account</b>	<b>property</b>
----------------	-----------------

property Account : string

>Returns or sets the account for the contact.

<b>Anniversary</b>	<b>property</b>
--------------------	-----------------

property Anniversary : TDateTime

>Returns or sets the anniversary date for the contact.

<b>AssistantName</b>	<b>property</b>
----------------------	-----------------

property AssistantName : string

>Returns or sets the name of the person who is the assistant for the contact.

See also: AssistantTelephoneNumber

<b>AssistantTelephoneNumber</b>	<b>property</b>
---------------------------------	-----------------

property AssistantTelephoneNumber : string

>Returns or sets the telephone number of the person who is the assistant for the contact.

See also: AssistantName

<b>BillingInformation</b>	<b>property</b>
---------------------------	-----------------

property BillingInformation : string

>Returns or sets the billing information associated with the Outlook item.

This is a free-form text field.

**Birthday** property


---

`property Birthday : TDateTime`

- ↳ Returns or sets the birthday for the contact.

**Business2TelephoneNumber** property


---

`property Business2TelephoneNumber : string`

- ↳ Returns or sets the second business telephone number for the contact.

**BusinessAddress** property


---

`property BusinessAddress : string`

- ↳ Returns or sets the whole, unparsed business address for the contact.

**BusinessAddressCity** property


---

`property BusinessAddressCity : string`

- ↳ Returns or sets the city name portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessAddressCountry** property


---

`property BusinessAddressCountry : string`

- ↳ Returns or sets the country code portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessAddressPostalCode****property****property BusinessAddressPostalCode : string**

>Returns or sets the postal code portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessAddressPostOfficeBox****property****property BusinessAddressPostOfficeBox : string**

>Returns or sets the post office box number portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessAddressState****property****property BusinessAddressState : string**

>Returns or sets the state code portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessAddressStreet****property**

```
property BusinessAddressStreet : string
```

- >Returns or sets the street address portion of the business address for the contact.

This property is parsed from the BusinessAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to the BusinessAddress property.

See also: BusinessAddress

**BusinessFaxNumber****property**

```
property BusinessFaxNumber : string
```

- >Returns or sets the business fax number for the contact.

See also: BusinessHomePage, BusinessTelephoneNumber

**BusinessHomePage****property**

```
property BusinessHomePage : string
```

- >Returns or sets the URL of the business Web page for the contact.

See also: BusinessFaxNumber, BusinessTelephoneNumber

**BusinessTelephoneNumber****property**

```
property BusinessTelephoneNumber : string
```

- >Returns or sets the first business telephone number for the contact.

See also: BusinessFaxNumber, BusinessHomePage

---

**CallbackTelephoneNumber** property

```
property CallbackTelephoneNumber : string
```

- ↳ Returns or sets the callback telephone number for the contact.

---

**CarTelephoneNumber** property

```
property CarTelephoneNumber : string
```

- ↳ Returns or sets the car telephone number for the contact.

---

**Categories** property

```
property Categories : string
```

- ↳ Returns or sets the categories assigned to the Contact item.

The Categories property is defined by the application. A category is a keyword or phrase that allows you to find, sort, filter, and group Outlook items. For example, contact items can be categorized as “Business” or “Personal.” Outlook provides a list of general categories, but this list can be modified as necessary.

---

<b>Children</b>	<b>property</b>
-----------------	-----------------

---

`property Children : string`

↳ Returns a semicolon delimited list of the contact's children.

---

<b>Close</b>	<b>method</b>
--------------	---------------

---

`procedure Close(SaveMode : TOpOlInspectorClose);`

`TOpOlInspectorClose = (olicSave, olicDiscard, olicPromptForSave);`

Default: `olicPromptForSave`

↳ Closes the inspector or item and optionally saves changes to the displayed Outlook item.

**Table 14:**

<b>Value</b>	<b>Description</b>
<code>olSave</code>	Save all changes without prompting.
<code>oldiscard</code>	Discard all changes without prompting.
<code>olPromptForSave</code>	Prompt to save or discard all changes.

See also: `Save`, `SaveAs`

---

<b>Companies</b>	<b>property</b>
------------------	-----------------

---

`property Companies : string`

↳ Returns or sets the names of the companies associated with the Contact item.

This is a free-form text field.

See also: `CompanyAndFullName`, `CompanyLastFirstNoSpace`,  
`CompanyLastFirstSpaceOnly`, `CompanyMainTelephoneNumber`, `CompanyName`

---

<b>CompanyAndFullName</b>	<b>property</b>
---------------------------	-----------------

---

`property CompanyAndFullName : string`

↳ Returns the concatenated company name and full name for the contact.

See also: `Companies`, `CompanyLastFirstNoSpace`, `CompanyLastFirstSpaceOnly`,  
`CompanyMainTelephoneNumber`, `CompanyName`

**CompanyLastFirstNoSpace****property****property CompanyLastFirstNoSpace : string**

- >Returns the company name for the contact followed by the concatenated last name, first name, and middle name with no space between the last and first names.

This property is parsed from the CompanyName, LastName, FirstName, and MiddleName properties. The LastName, FirstName, and MiddleName properties are themselves parsed from the FullName property.

See also: Companies, CompanyAndFullName, CompanyLastFirstSpaceOnly, CompanyMainTelephoneNumber, CompanyName

**CompanyLastFirstSpaceOnly****property****property CompanyLastFirstSpaceOnly : string**

- >Returns the company name for the contact followed by the concatenated last name, first name, and middle name with spaces between the last, first, and middle names.

This property is parsed from the CompanyName, LastName, FirstName, and MiddleName properties. The LastName, FirstName, and MiddleName properties are themselves parsed from the FullName property.

See also: Companies, CompanyAndFullName, CompanyLastFirstNoSpace, CompanyMainTelephoneNumber, CompanyName

**CompanyMainTelephoneNumber****property****property CompanyMainTelephoneNumber : string**

- >Returns or sets the company main telephone number for the contact.

See also: Companies, CompanyAndFullName, CompanyLastFirstNoSpace, CompanyLastFirstSpaceOnly, CompanyName

**CompanyName****property****property CompanyName : string**

- >Returns or sets the company name for the contact.

See also: Companies, CompanyAndFullName, CompanyLastFirstNoSpace, CompanyLastFirstSpaceOnly, CompanyMainTelephoneNumber

**ComputerNetworkName****property**

```
property ComputerNetworkName : string
```

- ↳ Returns or sets the name of the computer network for the contact.

**ContactItem****read-only property**

```
property ContactItem : _ContactItem
```

- ↳ Provides access to the automation interface for this contact item.

The majority of the functionality of this interface has already been encapsulated within the framework of OfficePartner. This property allows you to expand on this functionality. Consult the OfficePartner source code and any additional documentation from Microsoft (such as MSDN) prior to using this interface.

See also: Create

**Copy****method**

```
function Copy : IDispatch;
```

- ↳ Creates a new TOpContactItem object and returns an IDispatch interface to the new object.

See also: ContactItem

**Create****constructor**

```
constructor Create(AContactItem : _ContactItem);
```

- ↳ Creates an instance of the TOpContactItem class.

AContactItem contains an interface to the newly created TOpContactItem object.

Do not call this function directly. Instead, create instances of Contact items by calling the CreateContactItem method of the TOpOutlook component as follows:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ai : TOpContactItem;
begin
  ai := OpOutlook1.CreateContactItem;
  ai.{methods..}
  ...
end;
```

See also: ContactItem

**CreationTime****property****property CreationTime : TDateTime**

>Returns the creation time for the Contact item.

**CustomerID****property****property CustomerID : string**

>Returns or sets the customer ID for the contact.

**Delete****method****procedure Delete;**

Deletes this TOpContactItem object and removes it from the parent collection.

See also: Save, SaveAs

**Department****property****property Department : string**

>Returns or sets the department name for the contact.

**Display****method****procedure Display(Modal : Boolean);**

Displays Outlook's native contact inspector window.

Modal defines the modality of the inspector window. If Modal is True then the inspector window will be displayed modally. Otherwise, the inspector window will be displayed non-modally.

The following code illustrates how to display the contact inspector window:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ai : TOpContactItem;
begin
  ai := OpOutlook1.CreateContactItem;
  ai.FirstName := 'New';
  {show inspector modally}
  ai.Display(True);
end;
```

See also: ItemInspector

**Email1Address****property**

```
property Email1Address : string
```

>Returns or sets the e-mail address of the first e-mail entry for the contact.

See also: [Email1AddressType](#), [EMail1DisplayName](#), [EMail1EntryID](#)

**Email1AddressType****property**

```
property Email1AddressType : string
```

>Returns or sets the address type (such as EX or SMTP) of the first e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: [EMail1Address](#), [EMail1DisplayName](#), [EMail1EntryID](#)

**Email1DisplayName****property**

```
property Email1DisplayName : string
```

>Returns the display name of the first e-mail address for the contact.

This property is set to the value of the [FullName](#) property by default.

See also: [EMail1Address](#), [Email1AddressType](#), [EMail1EntryID](#)

**Email1EntryID****read-only property**

```
property Email1EntryID : string
```

>Returns the entry ID of the first e-mail address for the contact.

See also: [EMail1Address](#), [Email1AddressType](#), [EMail1DisplayName](#)

**Email2Address****property**

```
property Email2Address : string
```

>Returns or sets the e-mail address of the second e-mail entry for the contact.

See also: [Email2AddressType](#), [EMail2DisplayName](#), [EMail2EntryID](#)

**Email2AddressType****property****property Email2AddressType : string**

- >Returns or sets the address type (such as EX or SMTP) of the second e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: EMail2Address, EMail2DisplayName, EMail2EntryID

**Email2DisplayName****property****property Email2DisplayName : string**

- >Returns the display name of the second e-mail address for the contact.

This property is set to the value of the FullName property by default.

See also: EMail2Address, Email2AddressType, EMail2EntryID

**Email2EntryID****read-only property****property Email2EntryID : string**

- >Returns the entry ID of the second e-mail address for the contact.

See also: EMail2Address, Email2AddressType, EMail2DisplayName

**Email3Address****property****property Email3Address : string**

- >Returns or sets the e-mail address of the third e-mail entry for the contact.

See also: Email3AddressType, EMail3DisplayName, EMail3EntryID

**Email3AddressType****property****property Email3AddressType : string**

- >Returns or sets the address type (such as EX or SMTP) of the third e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: EMail3Address, EMail3DisplayName, EMail3EntryID

---

**Email3DisplayName** property

property Email3DisplayName : string

>Returns the display name of the third e-mail address for the contact.

This property is set to the value of the FullName property by default.

See also: EMail3Address, Email3AddressType, EMail3EntryID

---

**Email3EntryID** read-only property

property Email3EntryID : string

>Returns the entry ID of the third e-mail address for the contact.

See also: EMail3Address, Email3AddressType, EMail3DisplayName

---

**EntryID** read-only property

property EntryID : string

>Returns the unique entry ID of the Contact item.

This property corresponds to the index of the contact within the contacts collection.

---

**FileAs** property

property FileAs : string

>Returns or sets the default keyword string assigned to the contact when it is filed.

See also: Save, SaveAs

---

**FirstName** property

property FirstName : string

>Returns or sets the first name for the contact.

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to FullName.

See also: FullName

**ForwardAsVcard****method**


---

```
function ForwardAsVcard : TOpMailItem;
```

↳ Forwards the ContactItem as a virtual card.

A virtual card is the Internet standard for creating and sharing virtual business cards. The ForwardAsVcard method returns a TOpMailItem with the virtual card file attached.

**FTPSite****property**


---

```
property FTPSite : string
```

↳ Returns the FTP site entry for the contact.

**FullName****property**


---

```
property FullName : string
```

↳ Returns or sets the whole, unparsed full name for the contact.

This property is parsed into the FirstName, LastName, MiddleName, and Suffix properties, which may be changed or entered independently should they be parsed incorrectly. Note that any such changes or entries to the FirstName, LastName, MiddleName, or Suffix properties are overwritten by any subsequent changes or entries to FullName.

See also: FirstName, LastName, MiddleName, Suffix

**FullNameAndCompany****property**


---

```
property FullNameAndCompany : string
```

↳ Returns the full name and company of the contact by concatenating the values of the FullName and CompanyName properties.

See also: CompanyName, FullName

---

**Gender** **property**

`property Gender : TOp01Gender`

Default: olGendUnspecified

>Returns or sets the gender of the contact.

**Table 15:**

<b>Value</b>	<b>Description</b>
olFemale	The contact item is a female.
olMale	The contact item is a male.
olUnspecified	The gender of the contact item is unspecified.

---

**GovernmentIDNumber** **property**

`property GovernmentIDNumber : string`

>Returns or sets the government ID number for the contact.

---

**Hobby** **property**

`property Hobby : string`

>Returns or sets the hobby for the contact.

---

**Home2TelephoneNumber** **property**

`property Home2TelephoneNumber : string`

>Returns or sets the second home telephone number for the contact.

---

**HomeAddress** **property**

`property HomeAddress : string`

>Returns or sets the full, unparsed text of the home address for the contact.

---

**HomeAddressCity** **property**

`property HomeAddressCity : string`

>Returns or sets the city portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this

property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

### **HomeAddressCountry**

**property**

`property HomeAddressCountry : string`

↳ Returns or sets the country portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

### **HomeAddressPostalCode**

**property**

`property HomeAddressPostalCode : string`

↳ Returns or sets the postal code portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

### **HomeAddressPostOfficeBox**

**property**

`property HomeAddressPostOfficeBox : string`

↳ Returns or sets the post office box number portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

**HomeAddressState****property**

```
property HomeAddressState : string
```

- >Returns or sets the state portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

**HomeAddressStreet****property**

```
property HomeAddressStreet : string
```

- >Returns or sets the street portion of the home address for the contact.

This property is parsed from the HomeAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to HomeAddress.

See also: HomeAddress

**HomeFaxNumber****property**

```
property HomeFaxNumber : string
```

- >Returns or sets the home fax number for the contact.

See also: HomePhoneNumber

**HomePhoneNumber****property**

```
property HomePhoneNumber : string
```

- >Returns or sets the first home telephone number for the contact.

See also: HomeFaxNumber

---

<b>Importance</b>	<b>property</b>
-------------------	-----------------

---

property Importance : OlImportance  
 T0pOlImportance = (olImpLow, olImpNormal, olImpHigh);  
 Default: olImpNormal

- ↳ Returns or sets the relative importance level for the Outlook item.

---

<b>Initials</b>	<b>property</b>
-----------------	-----------------

---

property Initials : string

- ↳ Returns or sets the initials for the contact.

---

<b>InternetFreeBusyAddress</b>	<b>property</b>
--------------------------------	-----------------

---

property InternetFreeBusyAddress : string

- ↳ Returns or sets the URL location of the user's free/busy information in vCard Free/Busy standard format.

The InternetFreeBusyAddress property refers to the reading and publishing of a calendar user's free/busy map of events.

The map is retrieved when a user plans a meeting. This property corresponds to a field in the details page for an Internet-Only contact.

See also: ISDNNumber

---

<b>ISDNNumber</b>	<b>property</b>
-------------------	-----------------

---

property ISDNNumber : string

- ↳ Returns or sets the ISDN number for the contact.

See also: InternetFreeBusyAddress

---

<b>ItemInspector</b>	<b>property</b>
----------------------	-----------------

---

property ItemInspector : Inspector

- ↳ Maintains an interface to Outlook's native contact inspector window.

You can call the Display method of this interface to present the user with Outlook's familiar contact inspector window.

See also: ContactItem

---

**JobTitle** property

```
property JobTitle : string
```

>Returns or sets the job title for the contact.

---

**Journal** property

```
property Journal : Boolean
```

Default: False

>Returns or sets the journalization status for the contact.

---

**Language** property

```
property Language : string
```

>Returns or sets the language for the contact.

---

**LastFirstAndSuffix** property

```
property LastFirstAndSuffix : string
```

>Returns the last name, first name, middle name, and suffix of the contact.

There is a comma between the last and first names and spaces between all the names and the suffix. This property is parsed from the LastName, FirstName, MiddleName, and Suffix properties. The LastName, FirstName, MiddleName, and Suffix properties are themselves parsed from the FullName property.

See also: FirstName, LastName, MiddleName, Suffix

---

**LastFirstNoSpace** property

```
property LastFirstNoSpace : string
```

>Returns the concatenated last name, first name, and middle name of the contact with no space between the last name and the first name.

This property is parsed from the LastName, FirstName, and MiddleName properties. The LastName, FirstName, and MiddleName properties are parsed from the FullName property.

See also: FirstName, FullName, LastName, MiddleName

**LastFirstNoSpaceCompany****property****property** `LastFirstNoSpaceCompany : string`

- ↳ Returns the concatenated last name, first name, and middle name, and company of the contact with no space between the last name and the first name.

The company name for the contact is included after the middle name. This property is parsed from the LastName, FirstName, MiddleName, and CompanyName properties. The LastName, FirstName, and MiddleName are themselves parsed from the FullName property.

See also: FirstName, FullName, CompanyName, LastName, MiddleName

**LastFirstSpaceOnly****property****property** `LastFirstSpaceOnly : string`

- ↳ Returns the concatenated last name, first name, and middle name of the contact with spaces between them.

This property is parsed from the LastName, FirstName, and MiddleName properties. The LastName, FirstName, and MiddleName properties are parsed from the FullName property.

See also: FirstName, FullName, LastName, MiddleName

**LastFirstSpaceOnlyCompany****property****property** `LastFirstSpaceOnlyCompany : string`

- ↳ Returns the concatenated last name, first name, middle name and company name of the contact with spaces between them.

The company name for the contact is after the middle name. This property is parsed from the LastName, FirstName, MiddleName, and CompanyName properties. The LastName, FirstName, and MiddleName properties are parsed from the FullName property.

See also: FirstName, FullName, CompanyName, LastName, MiddleName

**LastModificationTime****property****property** `LastModificationTime : TDateTime`

- ↳ Returns the time that the Contact item was last modified.

See also: CreationTime

**LastName****property**


---

```
property LastName : string
```

- ↳ Returns or sets the last name for the contact.

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to FullName.

See also: FullName

**LastNameAndFirstName****property**


---

```
property LastNameAndFirstName : string
```

- ↳ Returns the concatenated last name and first name for the contact.

This property is parsed from the FirstName and LastName properties for the contact. The FirstName and LastName properties are parsed from the FullName property.

See also: FirstName, FullName, LastName

**MailingAddress****property**


---

```
property MailingAddress : string
```

- ↳ Returns or sets the full, unparsed selected mailing address for the contact.

This property replicates the property indicated by the SelectedMailingAddress property. The possible values are listed below:

**Table 16:**

<b>Value</b>	<b>Description</b>
olNone	No address
olHome	Home address
olBusiness	Business address
olOther	Other address

See also: SelectedMailingAddress

**MailingAddressCity****property****property MailingAddressCity : string**

>Returns or sets the city name portion of the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

**MailingAddressCountry****property****property MailingAddressCountry : string**

>Returns or sets the country code portion of the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

**MailingAddressPostalCode****property****property MailingAddressPostalCode : string**

>Returns or sets the postal code portion of the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

**MailingAddressPostOfficeBox****property**

`property MailingAddressPostOfficeBox : string`

- ↳ Returns or sets the post office box number portion of the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

**MailingAddressState****property**

`property MailingAddressState : string`

- ↳ Returns or sets the state code portion for the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

**MailingAddressStreet****property**

`property MailingAddressStreet : string`

- ↳ Returns or sets the street address portion of the selected mailing address of the contact.

This property is parsed from the SelectedMailingAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to SelectedMailingAddress.

See also: SelectedMailingAddress

---

**ManagerName** property

```
property ManagerName : string
```

↳ Returns or sets the manager name for the contact.

---

**MiddleName** property

```
property MiddleName : string
```

↳ Returns or sets the middle name for the contact.

---

**MobileTelephoneNumber** property

```
property MobileTelephoneNumber : string
```

↳ Returns or sets the mobile telephone number for the contact.

---

**Move** method

```
function Move(const DestFldr : MAPIFolder) : IDispatch;
```

↳ Moves the contact item to a new folder.

DestFldr is the Destination folder. Move returns an IDispatch interface to the destination folder object.

See also: Save, SaveAs

---

**NetMeetingAlias** property

```
property NetMeetingAlias : string
```

↳ Indicates the user's NetMeeting ID, or alias.

See also: NetMeetingServer

---

**NetMeetingServer** property

```
property NetMeetingServer : string
```

↳ Contains the name of the NetMeeting server being used for the Online meeting.

This property corresponds to the Directory Server field on the Online page of an online meeting item.

See also: NetMeetingAlias

**NickName** **property**

---

property NickName : string

>Returns or sets the nickname for the contact.

**NoAging** **property**

---

property NoAging : Boolean

Default: False

>Returns determines whether the contact item is aged.

Set this property to False to age the contact item.

**OfficeLocation** **property**

---

property OfficeLocation : string

>Returns or sets the specific office location for the contact.

This property returns more specific information than the BusinessAddressStreet property; for example, Building 6, Room 3, or Suite 354.

**OrganizationalIDNumber** **property**

---

property OrganizationalIDNumber : string

>Returns or sets the organizational ID number for the contact.

**OtherAddress** **property**

---

property OtherAddress : string

>Returns or sets the other address for the contact.

This property contains the full, unparsed other address for the contact.

**OtherAddressCity** **property**

---

property OtherAddressCity : string

>Returns or sets the city portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

See also: OtherAddress

**OtherAddressCountry****property**`property OtherAddressCountry : string`

>Returns or sets the country portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

See also: OtherAddress

**OtherAddressPostalCode****property**`property OtherAddressPostalCode : string`

>Returns or sets the postal code portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

See also: OtherAddress

**OtherAddressPostOfficeBox****property**`property OtherAddressPostOfficeBox : string`

>Returns or sets the post office box portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

See also: OtherAddress

**OtherAddressState****property**`property OtherAddressState : string`

>Returns or sets the state portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

See also: OtherAddress

`property OtherAddressStreet : string`

- ↳ Returns or sets the street portion of the other address for the contact.

This property is parsed from the OtherAddress property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes or entries to OtherAddress.

---

**OtherFaxNumber** property

```
property OtherFaxNumber : string
```

>Returns or sets the other fax number for the contact.

---

**OtherTelephoneNumber** property

```
property OtherTelephoneNumber : string
```

>Returns or sets the other telephone number for the contact.

---

**PagerNumber** property

```
property PagerNumber : string
```

>Returns or sets the pager number for the contact.

---

**PersonalHomePage** property

```
property PersonalHomePage : string
```

>Returns or sets the URL of the personal Web page for the contact.

---

**PrimaryTelephoneNumber** property

```
property PrimaryTelephoneNumber : string
```

>Returns or sets the primary telephone number for the contact.

---

**PrintOut** method

```
procedure PrintOut;
```

Prints the Outlook item using all default printer settings.

The PrintOut method is the only Outlook method that can be used for printing.

See also: Save, SaveAs

---

**Profession** property

```
property Profession : string
```

- ↳ Returns or sets the profession for the contact.

---

**RadioTelephoneNumber** property

```
property RadioTelephoneNumber : string
```

- ↳ Returns or sets the radio telephone number for the contact.

---

**ReferredBy** property

```
property ReferredBy : string
```

- ↳ Returns or sets the referral name entry for the contact.

---

**Save** method

```
procedure Save;
```

- ↳ Saves the Contact item to the current folder or, if this is a new item, to the Outlook default folder for the item type.

See also: SaveAs

**SaveAs****method**

```
procedure SaveAs(
    const Path : string; SaveAsType : TOpOlSaveAsType);
TOpOlSaveAsType = (
    olsatTXT, olsatRTF, olsatTemplate, olsatMSG, olsatDoc,
    olsatHTML, olsatVCard, olsatVCal);
```

↳ Saves the contact item under a new name.

Path defines the disk file name to save the contact item as.

SaveAsType defines the file type to save the contact item as.

If the file type is not specified, the MSG format is used. The following table defines the valid values:

**Table 17:**

T0lSaveAsType	Description
olDoc	Document file
olHTML	HTML file
olMSG	Message file
olRTF	Rich text file
olTemplate	Template
olTXT	Text file
olvCal	Virtual calendar item
olvCard	Virtual card item

See also: Save

**Saved****property**

property Saved : Boolean

Default: False

↳ Indicates whether the contact item has been modified since the last save.

Saved is True if the Outlook item has not been modified since the last save.

See also: Save, SaveAs

---

**SelectedMailingAddress** property

```
property SelectedMailingAddress : TOpOlMailingAddress
TOpOlMailingAddress = (
    olmaNone, olmaHome, olmaBusiness, olmaOther);
```

Default: olmaNone

↳ Returns or sets the type of the mailing address for the contact.

The following are the possible values for TOlMailing Address:

**Table 18:**

Value	Description
olNone	No address
olHome	Home address
olBusiness	Business address
olOther	Other address

---

**Size** read-only property

```
property Size : Integer
```

↳ Returns the size (in bytes) of the Outlook item.

---

**Spouse** property

```
property Spouse : string
```

↳ Returns or sets the spouse name entry for the contact.

---

**Suffix** property

```
property Suffix : string
```

↳ Returns or sets the name suffix (such as Jr., III, or Ph.D.) for the contact.

This property is parsed from the FullName property, but may be changed or entered independently should it be parsed incorrectly. Note that any such changes or entries to this property are overwritten by any subsequent changes of entries to FullName.

See also: FirstName, LastName, MiddleName, FullName

---

**TelexNumber** **property**

---

property TelexNumber : string

>Returns or sets the telex number for the contact.

**Title** **property**

---

property Title : string

>Returns or sets the title for the contact.

**TTYTDDTelephoneNumber** **property**

---

property TTYTDDTelephoneNumber : string

>Returns or sets the TTY/TDD telephone number for the contact.

**User1** **property**

---

property User1 : string

>Returns or sets the first Microsoft Schedule+ user for the contact.

**User2** **property**

---

property User2 : string

>Returns or sets the second Microsoft Schedule+ user for the contact.

**User3** **property**

---

property User3 : string

>Returns or sets the third Microsoft Schedule+ user for the contact.

**User4** **property**

---

property User4 : string

>Returns or sets the fourth Microsoft Schedule+ user for the contact.

**UserCertificate** **property**

---

property UserCertificate : string

>Returns or sets the string containing the user's authentication certificate for the contact.

---

**UserProperties** **property**

```
property UserProperties : UserProperties
```

- >Returns an interface to the UserProperties collection that represents all the user properties for the Outlook item.

UserProperties is an IDispatch interface through which ContactItem methods can be invoked and properties modified. Most of the functionality of this interface has already been encapsulated within the TOpContactItem class. This interface property allows you to expand on this functionality. Consult the OfficePartner source code and any additional Microsoft documentation (MSDN for example) prior to using this interface.

See also: ContactItem

---

**WebPage** **property**

```
property WebPage : string
```

- >Returns or sets the URL of the Web page for the contact.

---

**YomiCompanyName** **property**

```
property YomiCompanyName : string
```

- >Returns or sets the Japanese phonetic rendering (yomigana) of the company name for the contact.

---

**YomiFirstName** **property**

```
property YomiFirstName : string
```

- >Returns or sets the Japanese phonetic rendering (yomigana) of the first name for the contact.

---

**YomiLastName** **property**

```
property YomiLastName : string
```

- >Returns or sets the Japanese phonetic rendering (yomigana) of the last name for the contact.

# TOpJournalItem Class

The TOpJournalItem class represents a contact in the Journal folder.

TOpJournalItem class encapsulates the Outlook \_JournalItem interface. You will most commonly create an instance of this class using the TOpOutlook.CreateJournalItem method.

## Hierarchy

TObject (VCL)

TOpOutlookItem (OpOutlk)

TOpJournalItem (OpOutlk)

## Properties

Attachments	DocRouted	NoAging
BillingInformation	DocSaved	OutlookInternalVersion
Body	Duration	OutlookVersion
Categories	End_	Saved
Companies	EntryID	Sensitivity
ContactNames	Importance	Size
ConversationIndex	ItemInspector	Start
ConversationTopic	JournalItem	Subject
CreationTime	LastModificationTime	Type_
DocPosted	MessageClass	UnRead
DocPrinted	Mileage	

## Methods

Close	Forward	SaveAs
Copy	PrintOut	StartTimer
Create	Reply	StopTimer
Delete	ReplyAll	
Display	Save	

## Reference Section

4

---

### Attachments property

---

`property Attachments : TOpAttachmentList`

- ↳ Returns an AttachmentList object that represents all the attachments for the journal entry.  
Individual attachments can be accessed by indexing into this list.

---

### BillingInformation property

---

`property BillingInformation : string`

- ↳ Returns or sets the billing information associated with the journal entry.

---

### Body property

---

`property Body : string`

- ↳ Returns or sets the body of the journal entry.  
This property corresponds to the main text (body) of the journal entry.

See also: Subject

---

**Categories** property


---

`property Categories : string`

>Returns or sets the categories assigned to the journal entry.

**Close**


---

method


---

`procedure Close(SaveMode : TOpOIInspectorClose);`

`TOpOIInspectorClose = (olicSave, olicDiscard, olicPromptForSave);`

Default: olicPromptForSave

>Returns the Explorer or the Inspector window.

For the explorer object, the Close method closes the explorer. No information is saved. For an inspector or Outlook item object, the Close method closes the inspector or item and optionally saves changes to the displayed Outlook item.

**Table 19:**

Value	Description
olicSave	Save all changes without prompting.
olicDiscard	Discard all changes without prompting.
olicPromptForSave	Prompt to save or discard all changes.

If the item displayed within the inspector has not been changed, this argument has no effect.

See also: Save, SaveAs

---

**Companies** property


---

`property Companies : string`

>Returns or sets the names of the companies associated with the journal entry.

**ContactNames**


---

property


---

`property ContactNames : string`

>Returns the contact names associated with the journal entry.

This property contains the display names for the contacts only. Use the Recipients collection to modify the contents of this string.

**ConversationIndex****property**

```
property ConversationIndex : string
```

>Returns the index of the conversation thread of the item.

See also: ConversationTopic

**ConversationTopic****read-only property**

```
property ConversationTopic : string
```

>Returns the topic of the conversation thread of the item.

See also: ConversationIndex

**Copy****method**

```
function Copy : TOpJournalItem;
```

Creates and returns a copy of the object.

This method is supported by all descendants of TOpOutlook items.

See also: Create, Delete

**Create****constructor**

```
constructor Create(AJournalItem : _JournalItem);
```

Creates an instance of a TOpJournalItem.

AJournalItem contains an interface to the newly created journal entry.

Rather than using this method to create a journal entry, you should call the CreateJournalItem method associated with the TOpOutlook component. The following code illustrates:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ji : TOpJournalItem;
begin
  ji := OpOutlook1.CreateJournalItem;
  {show inspector modally}
  ji.Display(True);
end;
```

See also: Delete

**CreationTime****property**


---

```
property CreationTime : TDateTime
```

>Returns the creation time for the journal entry.

See also: Duration

**Delete****method**


---

```
procedure Delete;
```

Deletes the Journal entry from the parent collection.

See also Copy, Create

**Display****method**


---

```
procedure Display(Modal : Boolean);
```

Displays a new journal entry inspector window.

Modal determines the modality of the item inspector window. If Modal is True, then the item inspector window will be displayed modally. Otherwise, the item inspector window will be displayed non-modally.

The property values of this instance of the journal entry are reflected within the inspector window.

See also: ItemInspector

**DocPosted****property**


---

```
property DocPosted : Boolean
```

Default: False

Indicates whether the journalized item was posted as part of the journalized session.

DocPosted is True if the journalized item was posted. If the item is not posted, this property returns False.

See also: DocPrinted, DocRouted, DocSaved

**DocPrinted****property**

property DocPrinted : Boolean

Default: False

↳ Indicates whether the journalized item was printed as part of the journalized session.

DocPrinted is True if the journalized item was printed. If the item is not printed, this property returns False.

See also: DocPosted, DocRouted, DocSaved

**DocRouted****property**

property DocRouted : Boolean

Default: False

↳ Indicates whether the journalized item was routed as part of the journalized session.

DocRouted is True if the journalized item was routed. If the item is not routed, this property returns False.

See also: DocPosted, DocPrinted, DocSaved

**DocSaved****property**

property DocSaved : Boolean

Default: False

↳ Indicates whether the journalized item was saved as part of the journalized session.

DocSaved is True if the journalized item is saved. If the item is not saved, this property returns False.

See also: DocPosted, DocPrinted, DocRouted

**Duration****property**

property Duration : Integer

↳ Returns or sets the duration (in minutes) of the appointment, journal entry, or recurrence pattern.

For recurrences, this property is only valid for appointments.

See also: CreationTime, End\_

---

**End\_** property**property** End\_ : TDateTime

↳ Returns or sets the end date and time of an appointment or journal entry.

See also: CreationTime, Duration

---

**EntryID** property**property** EntryID : string

↳ Returns the unique entry ID of the journal entry.

The EntryID property is not set for an Outlook item until it is saved or sent. The EntryID changes when an item is moved into another folder.

See also: Forward, Save, SaveAs

---

**Forward** method**function** Forward : TOpMailItem;

↳ Executes the Forward action for a journal entry.

The resulting copy is returned as a new object.

See also: Reply, PrintOut

---

**Importance** property**property** Importance : TOpOImportance

TOpOImportance = (olImpLow, olImpNormal, olImpHigh);

Default: olImpNormal

↳ Returns or sets the relative importance level for the journal entry.

Can be one of the following values:

**Table 20:**

<b>Value</b>	<b>Description</b>
olImportanceLow	Lowest importance
olImportanceNormal	Normal importance
olImportanceHigh	Highest importance

---

**ItemInspector** **property**

`property ItemInspector : Inspector`

- ↳ Provides an interface through which TOpJournalItem inspector window objects can be manipulated.

This interface allows you to expand on the functionality of the inspector window. It is recommended that you consult the OfficePartner source code and any additional Microsoft documentation (such as MSDN) prior to using this interface.

See also: Display

---

**JournalItem** **property**

`property JournalItem : _JournalItem`

- ↳ Provides an Interface through which TOpJournalItem objects can be manipulated.

The functionality of this interface has already been largely encapsulated for you within the OfficePartner framework. This interface allows you to expand on this functionality. It is recommended that you consult the OfficePartner source code and any additional Microsoft documentation (such as MSDN) prior to using this interface.

See also: Attachments

---

**LastModificationTime** **property**

`property LastModificationTime : TDateTime`

- ↳ Returns the time that the Outlook item was last modified.

See also: Save, SaveAs, CreationTime

---

**MessageClass** **property**

`property MessageClass : string`

- ↳ Returns or sets the message class for the Outlook item or Action.

The MessageClass property is used to link the journal entry to the ItemInspector form on which it is based.

See also: ItemInspector

---

**Mileage** property

```
property Mileage : string
```

>Returns or sets the mileage for an item.

This is a free-form string field and can be used to store mileage information associated with the item for purposes of reimbursement.

---

**NoAging** property

```
property NoAging : Boolean
```

Default: False

↳ Determines whether the Journal entry is aged or not.

Set this property to False to age the journal entry.

---

**OutlookInternalVersion** property

```
property OutlookInternalVersion : Integer
```

↳ Returns the build number of the Outlook application for an Outlook item.

See also: OutlookVersion

---

**OutlookVersion** property

```
property OutlookVersion : string
```

↳ Returns the major and minor version number of the Outlook application for a journal entry.

See also: OutlookInternalVersion

---

**PrintOut** method

```
procedure PrintOut;
```

↳ Prints the Outlook item using all default settings.

The PrintOut method is the only Outlook method that can be used for printing.

---

**Reply** method

```
function Reply : TOpMailItem;
```

Creates a reply, pre-addressed to the original sender, from the original message.

Returns the reply as a MailItem object.

See also: Forward, ReplyAll

---

**ReplyAll** method

```
function ReplyAll : TOpMailItem;
```

Creates a reply to all original recipients from the original message.

Returns the reply as a MailItem object.

See also: Reply

---

**Save** method

```
procedure Save;
```

Saves the Outlook item.

If this item exists, it is saved to the current folder. If this is a new item, it is saved to the Outlook default folder for the item type.

See also: SaveAs

---

**SaveAs** method

```
procedure SaveAs(
  const Path : string; SaveAsType : TOpOlSaveAsType);

TOpOlSaveAsType = (
  olsatTXT, olsatRTF, olsatTemplate, olsatMSG, olsatDoc,
  olsatHTML, olsatVCard, olsatVCal);
```

Saves the journal entry under a new name.

Path is a string representation of the fully qualified disk path in which to save the file.

SaveAsType determines the file type/extension and can be any of the following:

**Table 21:**

<b>Value</b>	<b>Description</b>
olDoc	Document file
olHTML	HTML file
olMSG	Message file
olRTF	Rich text file
olTemplate	Template
olTXT	Text file
olVCal	Virtual calendar item
olVCard	Virtual card item

See also: Save

If the file type is not specified, the MSG format is used.

---

<b>Saved</b>	<b>property</b>
--------------	-----------------

---

property Saved : Boolean

Default: False

↳ Indicates whether the Journal entry has been modified since the last save.

Saved is True if the Journal entry **Save**has not been modified since the last save.

See also: Save, SaveAs

---

<b>Sensitivity</b>	<b>property</b>
--------------------	-----------------

---

property Sensitivity : TOpOlSensitivity

```
TOpOlSensitivity = (
    olSensNormal, olSensPersonal, olSensPrivate, olSensConfidential);
```

Default: olSensNormal

↳ Returns or sets the sensitivity for the journal entry.

Refer to Microsoft's documentation for further details.

---

**Size****read-only property**

```
property Size : Integer
```

>Returns the size, in bytes, of the Outlook item.

---

**Start****property**

```
property Start : TDateTime
```

>Returns or sets the starting date and time for the appointment or journal entry.

See also: CreationTime, Duration, LastModificationTime

---

**StartTimer****method**

```
procedure StartTimer;
```

Starts the timer on the journal entry.

This method allows programmatic control of the timer function. The Duration, End, and Start properties are automatically updated when the timer is stopped.

See also: Duration, End, Start, StopTimer

---

**StopTimer****method**

```
procedure StopTimer;
```

Stops the timer on the journal entry.

This method allows programmatic control of the timer function. The Duration, End, and Start properties are automatically updated when the timer is stopped.

See also: Duration, End, Start, StartTimer

---

**Subject****property**

```
property Subject : string
```

Returns or sets the subject for the Outlook item.

This property corresponds to the main body of the journal entry.

See also: Body

---

**Type\_** property

```
property Type_ : string
```

>Returns or sets the type of the object.

Type\_ is a free-form string field, usually containing the display name of the journalizing application (for example, "MSWord").

---

**UnRead** property

```
property UnRead : Boolean
```

Default: True

↳ Indicates whether the Outlook item has been opened.

Unread is True if the Outlook item has not yet been opened (read).

# TOpMailItem Class

TOpMailItem encapsulates the Outlook `_MailItem` interface. Instances of this class are created to facilitate composing and sending mail messages. The primary properties of this class correspond to those typical of e-mail. For instance, this class contains the properties `MsgTo`, `MsgCC`, and `MsgBCC`.

Instances of TOpMailItem are typically created by calling the `CreateMailItem` method of the associated TOpOutlook component.

## Hierarchy

TObject (VCL)

    TOpOutlookItem (OpOutlk)

        TOpMailItem (OpOutlk)

## Properties

Attachments	MailItem	SenderName
AutoForwarded	MsgBCC	Sensitivity
Body	MsgCC	Sent
DeferredDeliveryTime	MsgTo	Size
HTMLBody	Recipients	Subject
Importance	Saved	UnRead

## Methods

ClearConversationIndex	Display	ReplyAll
Create	Forward	Save
Delete	PrintOut	Send
Destroy	Reply	

# Reference Section

## Attachments

property

```
property Attachments : TOpAttachmentList
```

↳ Returns a TOpAttachmentList object that represents all the attachments for the mail item.

Individual attachments are accessed by indexing into this list.

See also: Recipients

4

## AutoForwarded

property

```
property AutoForwarded : Boolean
```

Default: False

↳ Indicates whether the mail message was automatically forwarded.

AutoForwarded is True if the mail message was automatically forwarded, otherwise it is False.

See also: Forward

## Body

property

```
property Body : string
```

↳ Returns or sets the text body of the Outlook item.

This property corresponds to the main text (body) of the mail item.

See also: HTMLBody

## ClearConversationIndex

method

```
procedure ClearConversationIndex;
```

↳ Clears the index of the conversation thread for the mail message or post.

## Create

method

```
constructor Create(AMailItem : _MailItem);
```

↳ Creates an instance of a TOpMailItem and additionally creates an attachments and recipients collection and associates these collections with the newly created mail item.

AMailItem contains an interface to the newly created mail item.

You will generally create mail items by calling the CreateMailItem function of the TOpOutlook component as follows:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  mi : TOpMailItem;
begin
  mi := OpOutlook1.CreateMailItem;
  mi.MsgTo := 'someone@somewhere.com';
  mi.Subject := 'Test';
  mi.Body := 'Hello...';
  mi.Send;
end;
```

See also: Delete

### **DeferredDeliveryTime**

**property**

**property DeferredDeliveryTime : TDateTime**

↳ Returns or sets the date and time the mail message is to be delivered.

### **Delete**

**method**

**procedure Delete;**

↳ Deletes the mail item object from the collection.

See also: Create, Destroy

### **Destroy**

**destructor**

**destructor Destroy;**

↳ Frees all attachments to the mail item and destroys the mail item.

See also: Delete, Create

### **Display**

**method**

**procedure Display(Modal : Boolean);**

↳ Displays a new mail item inspector window for the mail item.

Modal defines the modality of the item inspector window. If Modal is true then the item inspector window will be displayed modally. Otherwise, the item inspector window will be displayed non-modally.

## **Forward**

**method**

```
function Forward : TOpMailItem;
```

↳ Executes the Forward action for a mail item.

Returns the resulting copy as a new object.

See also: Send

## **HTMLBody**

**property**

```
property HTMLBody : string
```

↳ Sets the HTML body of the item.

The HTMLBody property should be an HTML syntax string. This property corresponds to the main text (body) of the mail item in HTML format. Setting this property always updates the Body property immediately and clears the contents of the HTMLBody property on HTML aware stores (Internet-Only mode).

See also: Body

## **Importance**

**property**

```
property Importance : TOpO1Importance
```

```
TOpO1Importance = (olImpLow, olImpNormal, olImpHigh);
```

Default: olImpNormal

↳ Returns or sets the relative importance level for the mail item.

See also: Sensitivity

## **MailItem**

**property**

```
property MailItem : _MailItem
```

↳ MailItem contains an interface to the TOpMailItem object.

The OfficePartner framework has already encapsulated most of the functionality of this interface. This interface property allows you to expand this functionality. It is recommended that you consult the OfficePartner source code as well as additional information from Microsoft (MSDN, for example) prior to using the properties and methods of this interface.

See also: Attachments

**MsgBCC****property****property** `MsgBCC : string`

- >Returns the display list of names to receive a blind carbon copy of the MailItem.

This property contains the display names only. The TOpRecipientList class should be used to modify the BCC recipients.

See also: `MsgTo`, `MsgCC`

**MsgCC****property****property** `MsgCC : string`

- >Returns the display list of names to receive a carbon copy of the MailItem.

This property contains the display names only. The TOpRecipientList class should be used to modify the CC recipients.

See also: `MsgTo`, `MsgBcc`

**MsgTo****property****property** `MsgTo : string`

- >Returns the display list of names to receive the MailItem.

This list is a semicolon-delimited string of recipients.

See also: `MsgCC`, `MsgBcc`

**PrintOut****method****procedure** `PrintOut;`

- Prints the Outlook item using the default printer settings.

The PrintOut method is the only Outlook method that can be used for printing.

See also: `Forward`, `Send`

**Recipients****property****property** `Recipients : TOpRecipientList`

- >Returns a TOpRecipientList object that represents all the recipients for the mail item.

Individual recipients can be accessed by indexing into this list.

See also: `Attachments`

**Reply****method**

```
function Reply : TOpMailItem;
```

↳ Creates a reply which is pre-addressed to the original sender.

Reply the reply as a MailItem object.

See also: ReplyAll

**ReplyAll****method**

```
function ReplyAll : TOpMailItem;
```

↳ Creates a reply to all original recipients from the original message.

ReplyAll returns the reply as a TOpMailItem object.

See also: Reply

**Save****method**

```
procedure Save;
```

↳ Saves the Outlook item.

If the item already exists, it is saved to the current folder. If this is a new item, it is saved to the Outlook default folder for the item type.

See also: Saved, Send

**Saved****read-only property**

```
property Saved : Boolean
```

Default: False

↳ Indicates whether the mail item has been modified since the last save.

Saved is True if the Outlook item has not been modified since the last save.

See also: Save

**Send****method**

```
procedure Send;
```

↳ Sends the mail message to all recipients.

See also: Body, MsgTo, MsgCC, MsgBCC

See also: Forward, PrintOut

---

<b>SenderName</b>	<b>property</b>
-------------------	-----------------

---

property SenderName : string

↳ Returns the display name of the sender for the mail message.

See also: Body, MsgTo, MsgCC, MsgBCC

---

<b>Sensitivity</b>	<b>property</b>
--------------------	-----------------

---

property Sensitivity : TOpOlSensitivity

TOpOlSensitivity = (  
    olSensNormal, olSensPersonal, olSensPrivate, olSensConfidential);

Default: olSensNormal

↳ Returns or sets the sensitivity for the mail item.

Refer to Microsoft's documentation for further details.

See also: Importance

---

<b>Sent</b>	<b>read-only property</b>
-------------	---------------------------

---

property Sent : Boolean

Default: False

↳ Indicates whether a message has been sent.

Sent is True if the mail item has been sent.

In general, there are three different kinds of messages: sent, posted, and saved. Sent messages are traditional e-mail messages or meeting items sent to a recipient or public folder. Posted messages are created in a public folder. Saved messages are created and saved without either sending or posting.

See also: Send, Forward

---

**Size** read-only property

`property Size : Integer`

↳ Returns the size, in bytes, of the mail item.

---

**Subject** property

`property Subject : string`

↳ Returns or sets the subject for the Outlook item.

This property corresponds to the subject line of the mail item.

See also: Body, HTMLBody, MsgTo, MsgCC, MsgBcc

---

**UnRead** property

`property UnRead : Boolean`

Default: True

↳ Indicates whether the mail item has been opened.

Unread is be True if the mail item has not yet been opened.

# TOpNoteItem Class

TOpNoteItem encapsulates the Outlook `_NoteItem` interface. You most commonly create an instance of this class using the `TOpOutlook.CreateNoteItem` method.

## Hierarchy

`TObject (VCL)`

`TOpOutlookItem (OpOutlk)`

`TOpNoteItem (OpOutlk)`

## Properties

<code>Body</code>	<code>Height</code>	<code>Saved</code>
<code>Categories</code>	<code>LastModificationTime</code>	<code>Size</code>
<code>Color</code>	<code>Left</code>	<code>Subject</code>
<code>CreationTime</code>	<code>MessageClass</code>	<code>Top</code>
<code>EntryID</code>	<code>NoteItem</code>	<code>Width</code>

## Methods

<code>Close</code>	<code>Delete</code>	<code>Save</code>
<code>Copy</code>	<code>Display</code>	<code>SaveAs</code>
<code>Create</code>	<code>PrintOut</code>	

## Reference Section

### Body `property`

`property Body : string`

↳ Returns or sets the text body of the note item.

The `Body` property corresponds to the main text (body) of the NoteItem.

See also: `Subject`

---

<b>Categories</b>	<b>property</b>
-------------------	-----------------

---

`property Categories : string`

>Returns or sets the categories assigned to the note item.

See also: [MessageClass](#)

---

<b>Close</b>	<b>method</b>
--------------	---------------

---

`procedure Close(SaveMode : TOpOlInspectorClose);`

`TOpOlInspectorClose = (olicSave, olicDiscard, olicPromptForSave);`

>Returns the inspector or item and optionally saves changes to the displayed note item.

SaveMode determines the close behavior. It can be set to any of the following values:

**Table 22:**

Value	Description
<code>olicSave</code>	Save all changes without prompting.
<code>olicDiscard</code>	Discard all changes without prompting.
<code>olicPromptForSave</code>	Prompt to save or discard all changes.

---

If the item displayed within the inspector has not been changed, this argument has no effect.

See also: [Save](#), [SaveAs](#)

---

<b>Color</b>	<b>property</b>
--------------	-----------------

---

`property Color : TOpOlNoteColor`

`TOpOlNoteColor = (`  
`olncBlue, olncGreen, olncPink, olncYellow, olncWhite);`

Default: `olncYellow`

>Returns or sets the background color of the note.

---

<b>Copy</b>	<b>method</b>
-------------	---------------

---

`function Copy : TOpNoteItem;`

Creates and returns a copy of the note item.

This method is supported by all Outlook items.

See also: [Create](#), [Save](#), [SaveAs](#)

**Create****constructor**

```
constructor Create(ANoteItem : _NoteItem);
```

- ↳ Creates an instance of TOpNoteItem.

ANoteItem contains an interface to the newly created note item.

You will generally create a NoteItem by calling the CreateNoteItem of the associated TOpOutlook component rather than calling this method directly. The following example illustrates:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ni : TOpNoteItem;
begin
  ni := OpOutlook1.CreateNoteItem;
  {Set the properties of the note item...}
end;
```

See also: Copy, Delete

**CreationTime****property**

```
property CreationTime : TDateTime
```

- ↳ Returns the creation time for the note item.

See also: LastModificationTime

**Delete****method**

```
procedure Delete;
```

- ↳ Frees the Note item.

See also: Close, Copy, Create, Save, SaveAs

**Display****method**

```
procedure Display(Modal : Boolean);
```

- ↳ Displays a new note item inspector window for the note item.

Modal defines the modality of the item inspector window. If Modal is True, then the item inspector window will be displayed modally. Otherwise, the item inspector window will be displayed non modally.

See also: PrintOut

**EntryID****read-only property****property EntryID : string**

>Returns the unique entry ID of the object.

The EntryID property is not set for a mail item until it is saved. The EntryID changes when a mail item is moved into another folder.

**Height****property****property Height : Integer**

>Returns or sets the height, in pixels, of the note window.

See also: Left, Top, Width

**LastModificationTime****property****property LastModificationTime : TDateTime**

>Returns the time that the note item was last modified.

See also: CreationTime

**Left****property****property Left : Integer**

>Returns or sets the position, in pixels, of the left vertical edge of the note window from the edge of the screen.

See also, Top, Height, Width

**MessageClass****property****property MessageClass : string**

>Returns or sets the message class for the note item.

When an item is selected, Outlook uses the message class to locate the form and expose its properties, such as Reply commands.

See also: Categories

**NoteItem****property**

```
property NoteItem : _NoteItem
```

- ↳ NoteItem returns an interface to the TOpNoteItem class.

The TOpNoteItem class already encapsulates most of the functionality available in this interface. This property allows you to increase the functionality of the NoteItem class beyond that which is already encapsulated by OfficePartner. It is recommended that you consult the OfficePartner source code as well as any additional information from Microsoft (such as MSDN) prior to using this interface.

**PrintOut****method**

```
procedure PrintOut;
```

- ↳ Prints the Outlook item using the default printer settings.

The PrintOut method is the only Outlook method that can be used for printing.

See also: Display

**Save****method**

```
procedure Save;
```

- ↳ Saves the note item.

If the item already exists, it is saved to the current folder. If this is a new item, it is saved to the Outlook default folder for the item type.

See also: SaveAs, Saved

**SaveAs****method**

```
procedure SaveAs(
  const Path : string; SaveAsType : TOpOISaveAsType);

TOpOISaveAsType = (
  olsatTXT, olsatRTF, olsatTemplate, olsatMSG, olsatDoc,
  olsatHTML, olsatVCard, olsatVCal);
```

Default: olTxt

- ↳ Saves the Outlook item to the specified path and in the format of the specified file type.

Path defines the disk file name under which to save the note item.

---

`SaveAsType` defines the file type under which to save the note item. It can be set to any of the following:

**Table 23:**

<b>Value</b>	<b>Description</b>
<code>olDoc</code>	Document file
<code>olHTML</code>	HTML file
<code>olMSG</code>	Message file
<code>olRTF</code>	Rich text file
<code>olTemplate</code>	Template
<code>olTXT</code>	Text file
<code>olvCal</code>	Virtual calendar item
<code>olvCard</code>	Virtual card item

If the file type is not specified, the MSG format is used.

See also: [Save](#), [Saved](#)

---

**Saved** **read-only property**

`property Saved : Boolean`

Default: False

↳ Indicates whether the note item has been modified since the last save.

`Saved` is True if the note item has not been modified since the last save.

See also: [Save](#), [SaveAs](#)

---

**Size** **property**

```
property Size : Integer
```

- ↳ Returns the size, in bytes, of the note item.

---

**Subject** **read-only property**

```
property Subject : string
```

- ↳ Returns or sets the subject for the note item.

For a NoteItem object, the Subject property is a read-only string that is calculated from the body text of the note.

See also: Body

---

**Top** **property**

```
property Top : Integer
```

- ↳ Returns or sets the position, in pixels, of the top horizontal edge of the note window from the edge of the screen.

See also: Left, Height, Width

---

**Width** **property**

```
property Width : Integer
```

- ↳ Returns or sets the width, in pixels, for the note window.

See also: Top, Left, Height

# TOpPostItem Class

TOpPostItem encapsulates the Outlook \_PostItem interface. You will most commonly create an instance of this class using the TOpOutlook.CreatePostItem method.

The TOpPostItem class represents a post in a public folder that others may browse. Unlike a MailItem object, a PostItem object is not sent to a recipient. You use the Post method, which is analogous to the Send method for the MailItem object, to save the PostItem to the target public folder instead of mailing it.

## Hierarchy

TObject (VCL)

TOpOutlookItem (OpOutlk)

TOpPostItem (OpOutlk)

## Properties

Attachments	HTMLBody	ReceivedTime
BillingInformation	Importance	Saved
Body	ItemInspector	SenderName
Categories	LastModificationTime	Sensitivity
Companies	MessageClass	SentOn
ConversationIndex	Mileage	Size
ConversationTopic	NoAging	Subject
CreationTime	OutlookInternalVersion	UnRead
EntryID	OutlookVersion	
ExpiryTime	PostItem	

## Methods

ClearConversationIndex	Destroy	Reply
Close	Display	Save
Copy	Forward	SaveAs
Create	Post	
Delete	PrintOut	

## Reference Section

4

---

### Attachments property

---

property Attachments : TOpAttachmentList

↳ Returns a TOpAttachmentList object that represents all the attachments for the post item.

Individual attachments can be accessed by indexing into this list.

---

### BillingInformation property

---

property BillingInformation : string

↳ Returns or sets the billing information associated with the post item.

This is a free-form text field.

---

### Body property

---

property Body : string

↳ Returns or sets the text body of the post item.

This property corresponds to the main text (body) of the PostItem.

See also: HTMLBody, Subject

---

### Categories property

---

property Categories : string

↳ Returns or sets the categories assigned to the post item.

Multiple items in this list should be separated by semicolons.

---

### ClearConversationIndex method

---

procedure ClearConversationIndex;

↳ Clears the index of the conversation thread for the post.

See also: ConversationTopic, Post, Reply

---

**Close** **method**

```
procedure Close(SaveMode : TOpO1InspectorClose);
TOpO1InspectorClose = (olicSave, olicDiscard, olicPromptForSave);
Default: olicPromptForSave
```

↳ Closes the inspector or item and optionally saves changes to the displayed Outlook item.

SaveMode can be set to any of the following values:

**Table 24:**

<b>Value</b>	<b>Description</b>
olSave	The appointment item is saved without user intervention.
oldDiscard	The appointment item is discarded, not saved.
olPromptForSave	A dialog opens prompting the user to save the item or close without saving..

If the item displayed within the inspector has not been changed, this argument has no effect.

See also: Save, SaveAs

---

**Companies** **property**

```
property Companies : string
```

↳ Returns or sets the names of the companies associated with the post item.

This is a free-form text field.

---

**ConversationIndex** **property**

```
property ConversationIndex : string
```

↳ Returns the index of the conversation thread of the post item.

See also: ConversationTopic

---

**ConversationTopic** **property**

```
property ConversationTopic : string
```

↳ Returns the topic of the conversation thread of the post item.

See also: ConversationIndex

**Copy****method**

```
function Copy : TOpPostItem;
```

- ↳ Creates and returns a copy of the post item.

This method is supported by all Outlook items.

See also: Create, Forward, Display, Forward, PrintOut

**Create****constructor**

```
constructor Create(APostItem : _PostItem);
```

- ↳ Creates an instance of the Post item class and additionally creates an associated attachments collection.

APostItem contains an interface to the newly created post item.

Instances of TOpPostItems are generally created by calling the CreatePostItem function of the associated TOpOutlook component rather than calling this method. The following example illustrates the preferred method:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  pi : TOpPostItem;
begin
  pi := OpOutlook1.CreatePostItem;
  {Set the properties of the post item...}
end;
```

See also: Close, Delete, Destroy

**CreationTime****read-only property**

```
property CreationTime : TDateTime
```

- ↳ Returns the creation time for the post item.

See also: LastModificationTime

**Delete****method**

```
procedure Delete;
```

- ↳ Deletes the post object from the `parent collection.

See also: Close, Destroy

**Destroy****destructor**

```
destructor Destroy;
```

- ↳ Clears the associated attachments collection and then frees this instance of the post item.

See also: Delete, Create

**Display****method**

```
procedure Display(Modal : Boolean);
```

- ↳ The Display method displays a new inspector window object for the post item.

Modal defines the modality of the inspector window. Set Modal to True to display the inspector window modally. Set Modal to False to display the window non-modally.

See also: ItemInspector

**EntryID****property**

```
property EntryID : string
```

- ↳ Returns the unique entry ID of the post object.

The EntryID property is not set for an Outlook item until it is saved. Also, the EntryID changes when an item is moved into another folder.

**ExpiryTime****property**

```
property ExpiryTime : TDateTime
```

- ↳ Returns or sets the date and time at which the item becomes invalid and can be deleted automatically.

If no value is set for this property, the item doesn't expire. Consequently, the item is not automatically deleted.

See also: CreationTime, LastModificationTime

**Forward****method**

```
function Forward : TOpMailItem;
```

- ↳ Executes the Forward action for the post item.

Forward returns the resulting copy as a new object.

See also: Post, Reply

**HTMLBody****property**


---

```
property HTMLBody : string
```

- ↳ Sets the HTML body of the item.

The HTMLBody property should be an HTML syntax string.

Setting the HTMLBody property sets the EditorType property of the item. Setting the HTMLBody property always updates the Body property immediately.

Setting the Body property clears the contents of the HTMLBody property.

See also: Body, Subject

**Importance****property**


---

```
property Importance : TOpOImportance
```

```
TOpOImportance = (olImpLow, olImpNormal, olImpHigh);
```

Default: olImpNormal

- ↳ Returns or sets the relative importance level for the post item.

See also: Sensitivity

**ItemInspector****property**


---

```
property ItemInspector : Inspector
```

- ↳ This property contains a reference to Outlook's native post item editor window.

This window is made visible by calling the Display method of this class.

See also: Display

**LastModificationTime****read-only property**


---

```
property LastModificationTime : TDateTime
```

- ↳ Returns the time that the Outlook item was last modified.

See also: CreationTime, ExpiryTime

**MessageClass****property****property MessageClass : string**

>Returns or sets the message class for the post item.

When an item is selected, Outlook uses the message class to locate the form and expose its properties, such as Reply commands.

See also: Categories

**Mileage****property****property Mileage : string**

>Returns or sets the mileage for a post item.

This is a free-form string field and can be used to store mileage information associated with the item.

**NoAging****property****property NoAging : Boolean**

Default: False

>Returns whether the post item is aged.

Set NoAging to False age the post item. Otherwise, no aging occurs.

See also: CreationTime

**OutlookInternalVersion****read-only property****property OutlookInternalVersion : Integer**

>Returns the build number of the Outlook application for an Outlook item.

See also: OutlookVersion

**OutlookVersion****read-only property****property OutlookVersion : string**

>Returns the major and minor version number of the Outlook application for the post item.

See also: OutlookInternalVersion

---

**Post** **method**

```
procedure Post;
```

- ↳ Sends (posts) the PostItem object.

The Post method, which is analogous to the Send method for the MailItem object, is used to save the post to the target public folder instead of mailing it.

See also: Forward, Reply

---

**PostItem** **property**

```
property PostItem : _PostItem
```

- ↳ Provides an Interface through which TOpPostItem objects can be manipulated.

The functionality of TOpPostItem has been encapsulated within the OfficePartner framework. This interface property allows you to extend this functionality beyond that which is already encapsulated for you. It is recommended that you consult the OfficePartner source code and any additional documentation by Microsoft (for example, MSDN) prior to using the properties and methods of this interface.

See also: Attachments

---

**PrintOut** **method**

```
procedure PrintOut;
```

- ↳ Prints the post item using the default printer settings.

The PrintOut method is the only Outlook method that can be used for printing.

See also: Display, Forward

---

**ReceivedTime** **read-only property**

```
property ReceivedTime : TDateTime
```

- ↳ Returns the date and time at which the post was received.

See also: ExpiryTime, SentOn

**Reply****method**

```
function Reply : TOpMailItem;
```

↳ Creates a reply which is pre-addressed to the original sender.

Reply returns the reply as a MailItem object.

See also: Forward, Post

**Save****method**

```
procedure Save;
```

↳ Saves the post item.

If the item already exists, it is saved to the current folder. If this is a new item, it is saved to the Outlook default folder for the item type.

See also: SaveAs

**SaveAs****method**

```
procedure SaveAs(
    const Path : string; SaveAsType : TOpOlSaveAsType);
TOpOlSaveAsType = (
    olsatTXT, olsatRTF, olsatTemplate, olsatMSG, olsatDoc,
    olsatHTML, olsatVCard, olsatVCal);
```

Default: olTxt

↳ Saves the post item to the specified path and in the format of the specified file type.

Path defines the disk file name under which the post item is saved.

SaveAsType determines the file extension / type in which to save the file.

SaveAsType can be set to any of the following:

**Table 25:**

<b>Value</b>	<b>Description</b>
OldDoc	Document file
olHTML	HTML file
olMSG	Message file
olRTF	Rich text file
olTemplate	Template

**Table 25:**

<b>Value</b>	<b>Description</b>
OlTXT	Text file
OlVCal	Virtual calendar item
OlVCard	Virtual card item

If the file type is not specified, the MSG format is used.

See also: Save, Saved

---

<b>Saved</b>	<b>property</b>
--------------	-----------------

---

property Saved : Boolean

Default: False

↳ Indicates whether the post item has been modified since the last save.

Saved is True if the Outlook item has not been modified since the last save.

See also: Save, SaveAs

---

<b>SenderName</b>	<b>property</b>
-------------------	-----------------

---

property SenderName : string

↳ Returns the display name of the sender for the post.

See also: Forward

---

<b>Sensitivity</b>	<b>property</b>
--------------------	-----------------

---

property Sensitivity : TOpOlSensitivity

TOpOlSensitivity = (  
    olSensNormal, olSensPersonal, olSensPrivate, olSensConfidential);

Default: olSensNormal

↳ Returns or sets the sensitivity for the post item.

Refer to Microsoft's documentation for further details.

See also: Importance

---

<b>SentOn</b>	<b>property</b>
---------------	-----------------

---

`property SentOn : TDateTime`

>Returns the date and time on which the post was sent.

See also: CreationTime, ReceivedTime

---

<b>Size</b>	<b>read-only property</b>
-------------	---------------------------

---

`property Size : Integer`

>Returns the size, in bytes, of the Outlook item.

---

<b>Subject</b>	<b>property</b>
----------------	-----------------

---

`property Subject : string`

>Returns or sets the subject for the post item.

The Subject property is the default property for Outlook items.

See also: Body, HTMLBody

---

<b>UnRead</b>	<b>property</b>
---------------	-----------------

---

`property UnRead : Boolean`

Default: True

>Returns whether the post item has been opened or not.

Unread is True if the post item has not been opened. Otherwise, Unread is False.

# TOpTaskItem Class

TOpTaskItem encapsulates the Outlook \_TaskItem interface. You will most commonly create an instance of this class using the TOpOutlook.CreateTaskItem method.

This class represents a task (an assigned, delegated, or self-imposed task to be performed within a specified time frame) in a Tasks folder.

## Hierarchy

TObject (VCL)

TOpOutlookItem (OpOutlk)

TOpTaskItem (OpOutlk)

## Properties

ActualWork	Importance	ReminderTime
Attachments	IsRecurring	ResponseState
BillingInformation	ItemInspector	Role
Body	LastModificationTime	Saved
CardData	MessageClass	SchedulePlusPriority
Categories	Mileage	Sensitivity
Companies	NoAging	Size
Complete	Ordinal	StartDate
ContactNames	OutlookInternalVersion	Status
Contacts	OutlookVersion	StatusOnCompletionReci...
ConversationIndex	Owner	StatusUpdateRecipients
ConversationTopic	Ownership	Subject
CreationTime	PercentComplete	TaskItem
DateCompleted	Recipients	TeamTask
DelegationState	ReminderOverrideDefault	TotalWork
Delegator	ReminderPlaySound	UnRead
DueDate	ReminderSet	
EntryID	ReminderSoundFile	

## Methods

Assign	CancelResponseState	ClearRecurrencePattern
--------	---------------------	------------------------

Close	Display	Save
Copy	GetRecurrencePattern	SaveAs
Create	MarkComplete	Send
Delete	PrintOut	SkipRecurrence
Destroy	Respond	StatusReport

## Reference Section

4

---

### ActualWork property

```
property ActualWork : Integer
```

↳ Returns or sets the actual effort, in minutes, spent on the task.

Corresponds to the Actual work text box on the Details page of a TaskItem.

See also: TotalWork

---

### Assign method

```
function Assign : TOpTaskItem;
```

↳ Assigns a task and returns a TaskItem object that represents it.

This method allows a task to be assigned (delegated) to another user. You must create a task before you can assign it, and you must assign a task before you can send it. An assigned task is sent as a TaskRequestItem object.

See also: ResponseState

---

### Attachments read-only property

```
property Attachments : TOpAttachmentList
```

↳ Returns a TOpAttachmentList object that represents all the attachments for the TaskItem.

Individual attachments can be accessed by indexing into this list.

See also: Recipients

---

### BillingInformation property

```
property BillingInformation : string
```

↳ Returns or sets the billing information associated with the task item.

This is a free-form text field.

**Body****property**

```
property Body : string
```

- ↳ Returns or sets the text body of the task item.

This property corresponds to the main text (body) of the task item.

See also: [Subject](#)

**CancelResponseState****method**

```
procedure CancelResponseState;
```

- ↳ Resets an unsent response to a task request back to a simple task.

After you receive a task request and respond to it, but before sending the response, you can use this method to revert the task to the state it was in before you responded.

See also: [ResponseStatus](#)

**CardData****property**

```
property CardData : string
```

- ↳ Returns or sets the text of the card data for the task.

**Categories****property**

```
property Categories : string
```

- ↳ Returns or sets the categories assigned to the task item.

Multiple items in this list are separated with semicolons.

**ClearRecurrencePattern****method**

```
procedure ClearRecurrencePattern;
```

- ↳ Removes the recurrence settings and restores the single-occurrence state for an appointment or ask.

See also: [GetRecurrencePattern](#)

**Close****method**

```
procedure Close(SaveMode : TOpOILInspectorClose);  
TOpOILInspectorClose = (olicSave, olicDiscard, olicPromptForSave);  
Default: olicPromptForSave
```

- ↳ The Close method closes the inspector

SaveMode defines the close behavior. SaveMode can be set to any of the following:

4

**Table 26:**

<b>Value</b>	<b>Description</b>
olicSave	Save all changes without prompting.
olicDiscard	Discard all changes without prompting.
olicPromptForSave	Prompt to save or discard all changes.

If the item displayed within the inspector has not been changed, this argument has no effect.

See also: Delete

**Companies****property**

```
property Companies : string
```

- ↳ Returns or sets the names of the companies associated with the task item.

This is a free-form text field.

**Complete****property**

```
property Complete : Boolean
```

Default: False

- ↳ Indicates whether the task has been completed.

Complete is True if the task is completed. Otherwise, Complete is False.

See also: DateCompleted

---

**ContactNames** property

```
property ContactNames : string
```

- ↳ Returns the contact names associated with the task entry.

This property contains the display names for the contacts only. Use the Recipients collection to modify the contents of this string.

See also: Contacts

---

**Contacts** property

```
property Contacts : string
```

- ↳ Returns the contact names associated with the task.

This property contains the display names for the contacts only. Use the Recipients collection to modify the contents of this string.

See also: ContactNames

---

**ConversationIndex** read-only property

```
property ConversationIndex : string
```

- ↳ Returns the index of the conversation thread of the item.

See also: ConversationTopic

---

**ConversationTopic** read-only property

```
property ConversationTopic : string
```

- ↳ Returns the index of the conversation thread of the task item.

See also: ConversationIndex

---

**Copy** method

```
function Copy : TOptTaskItem;
```

- ↳ Creates and returns a copy of the task item object.

This method is supported by all Outlook items.

See also: Create, Close, Delete

**Create****constructor**

```
constructor Create(ATaskItem : _TaskItem);
```

- ↳ Creates a task item and additionally creates associated attachments and recipients collections.

ATaskItem contains an interface to the newly created task item.

You will generally create task items by calling the CreateTaskItem method of the associated TOpOutlook component. The following code illustrates the proper method of creating task items:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  ti : TOpTaskItem;
begin
  ti := OpOutlook1.CreateTaskItem;
  {set properties as required...}
end;
```

See also: Copy, Close, Delete

**CreationTime****property**

```
property CreationTime : TDateTime
```

- ↳ Returns the creation time for the Outlook item.

See also: StartDate

**DateCompleted****property**

```
property DateCompleted : TDateTime
```

- ↳ Returns or sets the completion date of the task.

See also: StartDate

---

**DelegationState** **property**

```
property DelegationState : TOp01TaskDelegationState
TOp01TaskDelegationState = (
    oltdsNotDelegated, oltdsUnknown,
    oltdsAccepted, oltdsDeclined);
```

Default: oltdsNotDelegated

- ↳ Returns the delegation state of the task.

DelegationState can be set to any one of the following:

**Table 27:**

<b>Value</b>	<b>Description</b>
oltaskNotDelegated	Not delegated
oltaskDelegationUnknown	Unspecified delegation
oltaskDelegationAccepted	Delegation accepted
oltaskDelegationDeclined	Delegation declined

See also: Delegator

---

**Delegator** **property**

```
property Delegator : string
```

- ↳ Returns the display name of the delegator for the task.

See also: DelegationState

---

**Delete** **method**

```
procedure Delete;
```

- ↳ Deletes the task item object from the parent collection.

See also: Destroy

---

**Destroy** **destructor**

```
destructor Destroy;
```

- ↳ Clears the associated attachments and recipients collections and then frees the task item.

See also: Delete, Close

---

<b>Display</b>	<b>method</b>
----------------	---------------

---

```
procedure Display(Modal : Boolean);
```

↳ The Display method displays a new task item inspector window for the item.

Modal defines the modality of the task item inspector window. Set Modal to True to display the window modally. Otherwise, if Modal is False, the inspector window is non-modal.

See also: ItemInspector

---

<b>DueDate</b>	<b>property</b>
----------------	-----------------

---

```
property DueDate : TDateTime
```

↳ Returns or sets the due date for the task.

See also: CreationTime

---

<b>EntryID</b>	<b>read-only property</b>
----------------	---------------------------

---

```
property EntryID : string
```

↳ Returns the unique entry ID of the task item object.

The EntryID property is not set for a task item until it is saved or sent. In addition, the EntryID changes when an item is moved into another folder.

---

<b>GetRecurrencePattern</b>	<b>method</b>
-----------------------------	---------------

---

```
function GetRecurrencePattern : RecurrencePattern;
```

↳ Returns an interface to a RecurrencePattern object that represents the recurrence attributes of an appointment or task.

If there is no existing recurrence pattern, a new, empty RecurrencePattern object is returned.

See also: ClearRecurrencePattern, SkipRecurrence

---

<b>Importance</b>	<b>property</b>
-------------------	-----------------

---

```
property Importance : TOpO1Importance
```

```
TOpO1Importance = (olImpLow, olImpNormal, olImpHigh);
```

Default: olImpNormal

↳ Returns or sets the relative importance level for the task item.

See also: Sensitivity

**IsRecurring****read-only property**

```
property IsRecurring : Boolean
```

Default: False

- ↳ Indicates whether the task is recurring.

IsRecurring is True if the appointment or task is a recurring appointment or task. When the GetRecurrencePattern method is used with an AppointmentItem or TaskItem object, this property is set to True.

See also: GetRecurrencePattern

**ItemInspector****property**

```
property ItemInspector : Inspector
```

- ↳ This property maintains a reference to Outlook's native task-item editor window.

This interface property extends the functionality of the task item inspector window beyond that which OfficePartner has already encapsulated. It is recommended that you consult the OfficePartner source code and any additional information from Microsoft (for example, MSDN) prior to using the properties and methods of this interface.

See also: Display

**LastModificationTime****read-only property**

```
property LastModificationTime : TDateTime
```

- ↳ Returns the time that the task item was last modified.

See also: CreationTime

**MarkComplete****method**

```
procedure MarkComplete;
```

- ↳ Marks the task as completed.

MarkComplete sets PercentComplete to 100, Complete to True, and DateCompleted to the current date.

See also: PercentComplete, Complete, DateCompleted

**MessageClass****property****property MessageClass : string**

>Returns or sets the message class for the task item.

When an item is selected, Outlook uses the message class to locate the form and expose its properties.

See also: Categories

**Mileage****property****property Mileage : string**

>Returns or sets the mileage for a task item.

This is a free-form string field and can be used to store mileage information associated with the item for purposes of reimbursement.

**NoAging****property****property NoAging : Boolean**

Default: False

>Returns whether the task item is aged or not.

Set NoAging to False to age the task item.

**Ordinal****property****property Ordinal : Integer**

>Returns or sets the position (ordinal) in the task item inspector view for the task.

**OutlookInternalVersion****read-only property****property OutlookInternalVersion : Integer**

>Returns the build number of the Outlook application for a task item.

See also: OutlookVersion

**OutlookVersion****property**

```
property OutlookVersion : string
```

>Returns the major and minor version number of the Outlook application for a task item.

See also: [OutlookInternalVersion](#)

**Owner****property**

```
property Owner : string
```

>Returns or sets the owner for the task.

This is a free-form string field. Setting this property to a user other than the current user does not have the effect of delegating the task.

See also: [Ownership](#)

**Ownership****property**

```
property Ownership : TOpOlTaskOwnership
```

```
TOpOlTaskOwnership = (
    oltoNewTask, oltoDelegatedTask, oltoOwnTask);
```

Default: oltoNewTask

>Returns the ownership state of the task.

Ownership can be set to any one of the following:

**Table 28:**

<b>Value</b>	<b>Description</b>
olNewTask	This is a new task.
olDelegatedTask	This is a delegated task.
olOwnTask	You are the owner of this task.

See also: [Owner](#)

**PercentComplete****property**

```
property PercentComplete : Integer
```

>Returns or sets the percentage of the task completed at the current date and time.

See also: [Complete](#), [Status](#)

## PrintOut

method

```
procedure PrintOut;
```

↳ Prints the task item using all default settings.

The PrintOut method is the only Outlook method that can be used for printing.

See also: Display, Send

## Recipients

read-only property

```
property Recipients : TOpRecipientList
```

↳ Returns a TOpRecipientList collection that represents all the recipients for the task item.

Individual recipients can be accessed by indexing into this list.

See also: Attachments

## ReminderOverrideDefault

property

```
property ReminderOverrideDefault : Boolean
```

Default: False

↳ Determines whether the reminder overrides the default behavior of the task.

ReminderOverrideDefault is True if the reminder overrides the default reminder behavior for the task.

Note: You must set the ReminderOverrideDefault property to validate the ReminderPlaySound and the ReminderSoundFile properties.

See also: ReminderPlaySound, ReminderSoundFile, ReminderSet

## ReminderPlaySound

property

```
property ReminderPlaySound : Boolean
```

Default: False

↳ Determines whether a sound should be played when a reminder occurs.

The ReminderPlaySound property must be set in order to validate the ReminderSoundFile property. To play a sound when a reminder occurs for a task, set this property to True.

Note: This property is only valid if the ReminderOverrideDefault property is set to True.

See also: ReminderSoundFile, ReminderSet, ReminderOverrideDefault

**ReminderSet****property**

```
property ReminderSet : Boolean
```

Default: False

↳ Indicates whether a reminder has been set for this task.

ReminderSet is True if a reminder has been set for this task.

See also: ReminderSoundFile, ReminderPlaySound, ReminderOverrideDefault

**ReminderSoundFile****property**

```
property ReminderSoundFile : string
```

↳ Returns or sets the path and filename of the sound file to play when the reminder occurs for this task.

This property is only valid if the ReminderOverrideDefault and ReminderPlaySound properties are set to True.

See also: ReminderSet, ReminderPlaySound, ReminderOverrideDefault

**ReminderTime****property**

```
property ReminderTime : TDateTime
```

↳ Returns or sets the date and time at which the reminder should occur for this task.

See also: CreationTime

Respond	method
---------	--------

```
function Respond(
    Response : TOpOlTaskResponse; NoUI,
    AdditionalTextDialog : Boolean) : TOpTaskItem;

TOpOlTaskResponse = (
    oltrSimple, oltrAssign,
    oltrAccept, oltrDecline);
```

↳ Responds to a meeting request for the TaskItem object.

Response can be set to any of the following values:

4

**Table 29:**

Valuee	Description
olTaskSimple	Simple task
olTaskAssign	Task assigned
osTaskAccept	Task accepted
olTaskDecline	Task declined

If NoUI is set to False, the AdditionalTextDialog parameter is valid. If NoUI is set to False and AdditionalTextDialog is set to true, the user is prompted to add comments to the response and the response is displayed in the inspector for editing.

The following table clarifies the inter-relationships of the parameters for this method.

**Table 30:**

NoUI	AdditionalTextDialog	Dialog Box	User Prompted for Input
False	False	Displayed	No
False	True	Displayed	Yes
True	Invalid argument	Not displayed	Ignored

---

**ResponseState** property

```
property ResponseState : TOpOlTaskResponse  
TOpOlTaskResponse = (  
    oltrSimple, oltrAssign, oltrAccept, oltrDecline);
```

Default: oltrSimple

↳ Returns or sets the overall status of the response to the task request.

Refer to Microsoft's documentation for further details.

See also: Respond

---

**Role** property

```
property Role : string
```

↳ Returns or sets the free-form text string associating the owner of a task with a role for the task.

---

**Save** method

```
procedure Save;
```

↳ Saves the task item.

If the item already exists, it is saved to the current folder. If this is a new task item, it is saved to Outlook's default folder for task items.

See also: SaveAs, Saved

```
procedure SaveAs(
  const Path : string; SaveAsType : TOp01SaveAsType);
TOp01SaveAsType = (
  olsatTXT, olsatRTF, olsatTemplate, olsatMSG, olsatDoc, olsatHTML,
  olsatVCard, olsatVCal);
```

Default: olTxt

↳ Saves the task item to the specified path and in the format of the specified file type.

Path contains the fully qualified disk path in which to save the file.

SaveAsType determines the file extension/format to save the file in.

SaveAsType can be set to any one of the following values:

**Table 31:**

<b>Value</b>	<b>Description</b>
olDoc	Document file
olHTML	HTML file
olMSG	Message file
olRTF	Rich text file
olTemplate	Template
olTXT	Text file
olvCal	Virtual calendar item
olvCard	Virtual card item

If the file type is not specified, the MSG format is used.

See also: Save, Saved

---

## Saved

**read-only property**

property Saved : Boolean

Default: False

↳ Determines if the task item has been modified since the last save.

Save is True if the task item has not been modified since the last save.

See also: Save, SaveAs

---

**SchedulePlusPriority** property

```
property SchedulePlusPriority : string
```

- ↳ Returns or sets the Microsoft Schedule+ priority for the task.

Can be 1 through 9, A through Z, or A1 through Z9. Priority 1 is the highest.

---

**Send** method

```
procedure Send;
```

- ↳ Sends the task to all recipients.

See also: Recipients

---

**Sensitivity** property

```
property Sensitivity : TOpOlSensitivity
```

```
TOpOlSensitivity = (
    olSensNormal, olSensPersonal, olSensPrivate, olSensConfidential);
```

Default: olSensNormal

- ↳ Returns or sets the sensitivity for the task item.

See also: Importance

---

**Size** read-only property

```
property Size : Integer
```

- ↳ Returns the size, in bytes, of the task item.

---

**SkipRecurrence** method

```
function SkipRecurrence : Boolean;
```

- ↳ Clears the current instance of a recurring task and sets the recurrence to the next instance of that task.

See also: ClearRecurrencePattern, GetRecurrencePattern

---

<b>StartDate</b>	<b>property</b>
------------------	-----------------

---

property StartDate : TDateTime

>Returns or sets the starting date and time for the task.

See also: CreationTime, DueDate

---

<b>Status</b>	<b>property</b>
---------------	-----------------

---

property Status : TOpOlTaskStatus

```
TOpOlTaskStatus = (
    olTskStatNotStarted, olTskStatInProgress, olTskStatComplete,
    olTskStatWaiting, olTskStatDeferred);
```

Default: olTskStatNotStarted

>Returns or sets the status for the task.

Status can be set to any one of the following:

**Table 32:**

<b>Value</b>	<b>Description</b>
olTaskNotStarted	The task has yet to begin.
olTaskInProgress	The task is in progress.
olTaskComplete	The task is completed.
olTaskWaiting	The task is waiting for a contingency.
olTaskDeferred	The task has been deferred.

This property corresponds to the Status column of an Outlook task item.

See also: StatusOnCompletionRecipients, StatusUpdateRecipients

---

<b>StatusOnCompletionRecipients</b>	<b>read-only property</b>
-------------------------------------	---------------------------

---

property StatusOnCompletionRecipients : string

>Returns the semicolon-delimited string of display names for recipients who receive status upon completion of the task.

This property is calculated from the Recipients property. Recipients returned by the StatusOnCompletionRecipients property correspond to BCC recipients in the Recipients collection.

See also: Status, StatusUpdateRecipients

**StatusReport****method**

```
function StatusReport : IDispatch;
```

- ↳ Sends a status report to all CC recipients (recipients returned by the StatusUpdateRecipients property) with the current status for the task.

See also: Recipients, Status, StatusOnCompletionRecipients, StatusUpdateRecipients

**StatusUpdateRecipients****read-only property**

```
property StatusUpdateRecipients : string
```

- ↳ Returns the semicolon-delimited string of display names for recipients who receive status updates for the task.

This property is calculated from the Recipients property. Recipients returned by the StatusUpdateRecipients property correspond to CC recipients in the Recipients collection.

See also: StatusOnCompletionRecipients, Status

**Subject****property**

```
property Subject : string
```

- ↳ Returns or sets the subject for the task item.

This property corresponds to the subject of the task.

See also: Body

**TaskItem****property**

```
property TaskItem : _TaskItem
```

- ↳ Contains an automation interface to a TaskItem object.

TaskItem represents a task (an assigned, delegated, or self-imposed task to be performed within a specified time frame) in a Tasks folder.

OfficePartner encapsulates much of the functionality of the task item. This interface property allows you to expand the functionality beyond that which is encapsulated by OfficePartner. It is recommended that you consult the OfficePartner source code and any additional information from Microsoft (for example, MSDN) prior to using the properties and methods of this interface.

See also: Attachments, Recipients

**TeamTask****property**

property TeamTask : Boolean

Default: False

↳ Determines whether a task is a team task.

TeamTask is True if the task is assigned to more than one person. Otherwise, it is False.

**TotalWork****property**

property TotalWork : Integer

Default: 0 (zero)

↳ Returns or sets the total work for the task.

Corresponds to the Total work field on the Details page of a TaskItem.

See also: DueDate

**UnRead****property**

property UnRead : Boolean

Default: True

↳ Indicates whether the task has been opened.

Unread is True if the Outlook item has not been opened (read).

---

## TOpBaseData Class

The TOpBaseData class is the immediate ancestor of TOpBusinessData class, TOpPersonalData class, TOpNetworkData class, TOpMiscData class, and TOpDefaultData class. It implements methods and properties used by the data classes but has no published properties.

TOpBaseData class is provided to facilitate creation of descendent classes.

### Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

---

## TOpBusinessData Class

The data encapsulated within this class pertains only to contact items. Refer to TOpContactItem on page 199 for further reference.

The business data class encapsulates the data and events associated with business contacts. The properties and methods of this class correspond to the information found on the Contact's General tab of Microsoft Outlook.

### Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

TOpBusinessData (OpDbOlk)

### Properties

Account	BusinessAddressState	Department
AssistantName	BusinessAddressStreet	GovernmentIDNumber
AssistanTelephoneNumber	BusinessFaxNumber	JobTitle
BillingInformation	BusinessHomePage	ManagerName
Business2TelephoneNum...	BusinessTelephoneNumber	OfficeLocation
BusinessAddressCity	Companies	OrganizationalIDNumber
BusinessAddressCountry	CompanyMainTelephone...	Profession
BusinessAddressPostalCo...	CompanyName	ReferredBy
BusinessAddressPostOffi...	CustomerID	Title

## Reference Section

4

---

<b>Account</b>	<b>property</b>
----------------	-----------------

---

property Account : string

>Returns or sets the account number for the contact.

---

<b>BusinessTelephoneNumber</b>	<b>property</b>
--------------------------------	-----------------

---

property BusinessTelephoneNumber : string

>Returns or sets the primary business telephone number for the contact.

---

<b>AssistantName</b>	<b>property</b>
----------------------	-----------------

---

property AssistantName : string

>Returns or sets the name of the person who is the assistant for the contact.

This property corresponds to the Assistant's Name box on the Details page of a ContactItem.

See also: AssistantTelephoneNumber

---

<b>AssistantTelephoneNumber</b>	<b>property</b>
---------------------------------	-----------------

---

property AssistantTelephoneNumber : string

>Returns or sets the telephone number of the person who is the assistant for the contact.

See also: AssistantName

---

<b>Business2TelephoneNumber</b>	<b>property</b>
---------------------------------	-----------------

---

property Business2TelephoneNumber : string

>Returns or sets the second business telephone number for the contact.

See also: BusinessTelephoneNumber

---

**BusinessAddressCity** **property**

---

property BusinessAddressCity : string

↳ Returns or sets the city name portion of the business address for the contact.

---

**BusinessAddressCountry** **property**

---

property BusinessAddressCountry : string

↳ Returns or sets the country code portion of the business address for the contact.

---

**BusinessAddressPostalCode** **property**

---

property BusinessAddressPostalCode : string

↳ Returns or sets the postal code (zip code) portion of the business address for the contact.

---

**BusinessAddressPostOfficeBox** **property**

---

property BusinessAddressPostOfficeBox : string

↳ Returns or sets the post office box number portion of the business address for the contact.

---

**BusinessAddressState** **property**

---

property BusinessAddressState : string

↳ Returns or sets the state code portion of the business address for the contact.

---

**BusinessAddressStreet** **property**

---

property BusinessAddressStreet : string

↳ Returns or sets the street address portion of the business address for the contact.

---

**BusinessFaxNumber** **property**

---

property BusinessFaxNumber : string

↳ Returns or sets the business fax number for the contact.

---

**BusinessHomePage** **property**

---

property BusinessHomePage : string

↳ Returns or sets the business home pager number for the contact.

---

**BillingInformation** **property**

property BillingInformation : string

- ↳ Returns or sets the billing information associated with the Outlook item.

---

**Companies** **property**

property Companies : string

- ↳ Returns or sets the names of the companies associated with the contact.

---

**CompanyMainPhoneNumber** **property**

property CompanyMainPhoneNumber : string

- ↳ Returns or sets the main telephone number of the contact's business.

---

**CompanyName** **property**

property CompanyName : string

- ↳ Returns or sets the company name associated with the contact.

---

**CustomerID** **property**

property CustomerID : string

- ↳ Returns or sets the customer ID associated with the contact.

---

**Department** **property**

property Department : string

- ↳ Returns or sets the department to which the contact is assigned.

---

**GovernmentIDNumber** **property**

property GovernmentIDNumber : string

- ↳ Returns or sets the Government ID number of the contact.

---

**JobTitle** **property**

property JobTitle : string

- ↳ Returns or sets the job title of the contact.

---

**ManagerName** property

```
property ManagerName : string
```

↳ Returns or sets the manager name of the contact.

---

**OfficeLocation** property

```
property OfficeLocation : string
```

↳ Returns or sets the location of the contact's office.

---

**OrganizationalIDNumber** property

```
property OrganizationalIDNumber : string
```

↳ Returns or sets the Organizational ID of the contact.

---

**Profession** property

```
property Profession : string
```

↳ Returns or sets the profession of the contact.

---

**ReferredBy** property

```
property ReferredBy : string
```

↳ Sets or returns the name of the person by whom the contact was referred.

---

**Title** property

```
property Title : string
```

↳ Returns or sets the title of the contact.

---

# TOpDefaultData Class

The TOpDefaultData class encapsulates the most common string properties associated with a Microsoft Outlook contact.

Refer to the reference sections covering TOpPersonalData, TOpNetworkData, TOpMiscData, and TOpBusinessData for further classifications of contact-related data.

## Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

TOpDefaultData (OpDbOlk)

## Properties

FirstName	MailingAddressPostOffic...	PagerNumber
LastName	MailingAddressState	PrimaryTelephoneNumber
MailingAddressCity	MailingAddressStreet	Suffix
MailingAddressCountry	MiddleName	
MailingAddressPostalCode	MobileTelephoneNumber	

## Reference Section

<b>FirstName</b>	<b>property</b>
------------------	-----------------

property FirstName : string

>Returns or sets the first name of the contact.

<b>LastName</b>	<b>property</b>
-----------------	-----------------

property LastName : string

>Returns or sets the last name of the contact.

<b>MailingAddressCity</b>	<b>property</b>
---------------------------	-----------------

property MailingAddressCity : string

>Returns or sets the city of the primary mailing address of the contact.

<b>MailingAddressCountry</b>	<b>property</b>
------------------------------	-----------------

property MailingAddressCountry : string

>Returns or sets the country of the primary mailing address associated with the contact.

<b>MailingAddressPostalCode</b>	<b>property</b>
---------------------------------	-----------------

property MailingAddressPostalCode : string

>Returns or set the postal code (zip code) of the primary mailing address associated with the contact.

<b>MailingAddressPostOfficeBox</b>	<b>property</b>
------------------------------------	-----------------

property MailingAddressPostOfficeBox : string

>Returns or sets the post office box number of the primary mailing address associated with the contact.

<b>MailingAddressState</b>	<b>property</b>
----------------------------	-----------------

property MailingAddressState : string

>Returns or sets the state of the primary mailing address associated with the contact.

<b>MailingAddressStreet</b>	<b>property</b>
-----------------------------	-----------------

`property MailingAddressStreet : string`

- ↳ Returns or sets the street of the primary mailing address associated with the contact.

#### **MiddleName**

**property**

`property MiddleName : string`

- ↳ Returns or sets the middle name of the contact.

4

#### **MobileTelephoneNumber**

**property**

`property MobileTelephoneNumber : string`

- ↳ Returns or sets the mobile telephone number of the contact.

See also: PagerNumber, PrimaryTelephoneNumber

#### **PagerNumber**

**property**

`property PagerNumber : string`

- ↳ Returns or sets the pager number of the contact.

See also: MobileTelephoneNumber, PrimaryTelephoneNumber

#### **PrimaryTelephoneNumber**

**property**

`property PrimaryTelephoneNumber : string`

- ↳ Returns or sets the primary telephone number of the contact.

See also: MobileTelephoneNumber, PagerNumber

#### **Suffix**

**property**

`property Suffix : string`

- ↳ Returns or sets the suffix (e.g. Jr., Sr.) of the contact.

---

## TOpMiscData Class

The TOpMiscData class encapsulates the properties of a contact item that generally don't fit into any other category.

Refer to the reference sections covering TOpPersonalData, TOpNetworkData, TOpDefaultData, and TOpBusinessData for further classifications of contact-related data.

### Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

TOpMiscData (OpDbOlk)

### Properties

CallbackTelephoneNumber	OtherAddressPostOffice...	User3
CarTelephoneNumber	OtherAddressState	User4
Categories	OtherAddressStreet	UserCertificate
Language	OtherFaxNumber	YomiCompanyName
OtherAddressCity	OtherPhoneNumber	YomiFirstName
OtherAddressCountry	User1	YomiLastName
OtherAddressPostalCode	User2	

### Reference Section

---

Categories	property
------------	----------

property Categories : string

↳ Returns or sets the categories of the contact.

---

CallbackTelephoneNumber	property
-------------------------	----------

property CallbackTelephoneNumber : string

↳ Returns or sets the contact's callback telephone number.

See also: CarTelephoneNumber, OtherPhoneNumber, OtherFaxNumber

## **CarTelephoneNumber**

## **property**

property CarTelephoneNumber : string

>Returns or sets the contact's car telephone number.

See also: [CallbackTelephoneNumber](#), [OtherTelephoneNumber](#), [OtherFaxNumber](#)

<b>Language</b>	<b>property</b>
<b>property Language : string</b>	
↳ Returns or sets the language of the contact.	
<b>OtherAddressCity</b>	<b>property</b>
<b>property OtherAddressCity : string</b>	
↳ Returns or sets the city of the contact's second address.	
<b>OtherAddressCountry</b>	<b>property</b>
<b>property OtherAddressCountry : string</b>	
↳ Returns or sets the country of the contact's second address.	
<b>OtherAddressPostalCode</b>	<b>property</b>
<b>property OtherAddressPostalCode : string</b>	
↳ Returns or sets the postal (zip) code of the contact's second address.	
<b>OtherAddressPostOfficeBox</b>	<b>property</b>
<b>property OtherAddressPostOfficeBox : string</b>	
↳ Returns or sets the post office box number of the contact's second address.	
<b>OtherAddressState</b>	<b>property</b>
<b>property OtherAddressState : string</b>	
↳ Returns or sets the state of the contact's second address.	
<b>OtherAddressStreet</b>	<b>property</b>
<b>property OtherAddressStreet : string</b>	
↳ Returns or sets the street of the contact's second address.	
<b>OtherFaxNumber</b>	<b>property</b>
<b>property OtherFaxNumber : string</b>	
↳ Returns or sets the secondary fax number of the contact.	

---

**OtherTelephoneNumber** **property**

```
property OtherTelephoneNumber : string
```

>Returns or sets the secondary telephone number of the contact.

---

**User1** **property**

```
property User1 : string
```

>Returns or sets the first Microsoft Schedule+ user for the contact.

---

**User2** **property**

```
property User2 : string
```

>Returns or sets the second Microsoft Schedule+ user for the contact.

---

**User3** **property**

```
property User3 : string
```

>Returns or sets the third Microsoft Schedule+ user for the contact.

---

**User4** **property**

```
property User4 : string
```

>Returns or sets the fourth Microsoft Schedule+ user for the contact.

---

**UserCertificate** **property**

```
property UserCertificate : string
```

>Returns or sets the string containing the user's authentication certificate for the contact.

---

**YomiCompanyName** **property**

```
property YomiCompanyName : string
```

>Returns or sets the Japanese phonetic rendering (yomigana) of the company name for the contact.

---

**YomiFirstName** **property**

```
property YomiFirstName : string
```

>Returns or sets the Japanese phonetic rendering (yomigana) of the contact's first name.

---

**YomiLastName****property**

```
property YomiLastName : string
```

↳ Returns or sets the Japanese phonetic rendering (yomigana) of the contact's last name.

# TOpNetworkData Class

The TOpNetworkData class encapsulates the properties of a contact item that are related to networking.

Refer to the reference sections covering TOpPersonalData, TOpMiscData, TOpDefaultData, and TOpBusinessData for further classifications of contact-related data.

## Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

TOpNetworkData (OpDbOlk)

## Properties

ComputerNetworkName	Email3AddressType	PersonalHomePage
Email1Address	FTPSite	RadioTelephoneNumber
Email1AddressType	InternetFreeBusyAddress	TelexNumber
Email2Address	ISDNNumber	TTYTDDTelephoneNum...
Email2AddressType	NetMeetingAlias	WebPage
Email3Address	NetMeetingServer	

## Reference Section

### ComputerNetworkName property

`property ComputerNetworkName : string`

↳ Returns or sets the name of the computer network for the contact.

See also: FTPSite, ISDNNumber, PersonalHomePage

### Email1Address property

`property Email1Address : string`

↳ Returns or sets the first e-mail address of the contact.

See also: Email1AddressType

**Email1AddressType****property**

```
property Email1AddressType : string
```

- ↳ Returns or sets the address type (such as EX or SMTP) of the first e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: Email1AddressType

**Email2Address****property**

```
property Email2Address : string
```

- ↳ Returns or sets the second e-mail address of the contact.

See also: Email2AddressType

**Email2AddressType****property**

```
property Email2AddressType : string
```

- ↳ Returns or sets the address type (such as EX or SMTP) of the second e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: EMail2Address

**Email3Address****property**

```
property Email3Address : string
```

- ↳ Returns or sets the third e-mail address of the contact.

See also: Email3AddressType

**Email3AddressType****property**

```
property Email3AddressType : string
```

- ↳ Returns or sets the address type (such as EX or SMTP) of the third e-mail entry for the contact.

This is a free-form text field, but it must match the actual type of an existing mail transport.

See also: EMail3Address

---

**FTPSite** **property**

`property FTPSite : string`

>Returns or sets the FTP site entry for the contact.

See also: ComputerNetworkName, ISDNNumber, NetMeetingServer

---

**InternetFreeBusyAddress** **property**

`property InternetFreeBusyAddress : string`

Sets or returns a URL, FTP, or server pathname to the internet free/busy file.

The InternetFreeBusyAddress property refers to the reading and publishing of a calendar user's free/busy map of events.

The map is retrieved when a user plans a meeting. This property corresponds to a field in the details page for an Internet Only contact. Refer to Microsoft's documentation and the Outlook help file for additional information.

See also: ComputerNetworkName, ISDNNumber, NetMeetingServer

---

**ISDNNumber** **property**

`property ISDNNumber : string`

>Returns or sets the ISDN number for the contact.

See also: ComputerNetworkName, FTPSite, InternetFreeBusyAddress, NetMeetingServer, WebPage

---

**NetMeetingAlias** **property**

`property NetMeetingAlias : string`

↳ Indicates the user's NetMeeting ID, or alias.

See also: ComputerNetworkName, FTPSite, InternetFreeBusyAddress, NetMeetingServer, WebPage

---

**NetMeetingServer** **property**

`property NetMeetingServer : string`

↳ Contains the name of the NetMeeting server being used for an Online meeting.

This property corresponds to the Directory Server field on the Online page of an online meeting item.

See also: ComputerNetworkName, FTPSite, InternetFreeBusyAddress, WebPage, NetMeetingAlias

---

**PersonalHomePage****property**

`property PersonalHomePage : string`

>Returns or sets the URL of the personal Web page for the contact.

See also: WebPage

---

**RadioTelephoneNumber****property**

`property RadioTelephoneNumber : string`

>Returns or sets the radio telephone number for the contact.

---

**TelexNumber****property**

`property TelexNumber : string`

>Returns or sets the telex number for the contact.

---

**TTYTDDTelephoneNumber****property**

`property TTYTDDTelephoneNumber : string`

>Returns or sets the TTY/TDD telephone number for the contact.

---

**WebPage****property**

`property WebPage : string`

>Returns or sets the URL of the web page of the contact.

See also: PersonalHomePage

---

# TOpPersonalData Class

The TOpPersonalData class encapsulates the personal properties of a contact item. These properties generally correspond to the entry fields that appear when Personal Fields is selected in the Select From combo box within the All Fields tab of Outlook's contact form.

Refer to the reference sections covering TOpNetworkData, TOpMiscData, TOpDefaultData, and TOpBusinessData for further classifications of contact-related data.

## Hierarchy

TObject (VCL)

TOpBaseData (OpDbOlk)

TOpPersonalData (OpDbOlk)

## Properties

Children	HomeAddressPostalCode	HomeTelephoneNumber
Hobby	HomeAddressPostOffice...	NickName
Home2TelephoneNumber	HomeAddressState	Spouse
HomeAddressCity	HomeAddressStreet	
HomeAddressCountry	HomeFaxNumber	

## Reference Section

---

<u>Children</u>	<u>property</u>
-----------------	-----------------

property Children : string

↳ Returns or sets the names of the contact's children.

See also: Spouse

---

**Hobby** property

`property Hobby : string`

>Returns or sets the hobby of the contact.

---

**Home2TelephoneNumber** property

`property Home2TelephoneNumber : string`

>Returns or sets the secondary home telephone number of the contact.

---

**HomeAddressCity** property

`property HomeAddressCity : string`

>Returns or sets the city of the contact's home address.

---

**HomeAddressCountry** property

`property HomeAddressCountry : string`

>Returns or sets the country of the contact's home address.

---

**HomeAddressPostalCode** property

`property HomeAddressPostalCode : string`

>Returns or sets the postal (zip) code of the contact's home address.

---

**HomeAddressPostOfficeBox** property

`property HomeAddressPostOfficeBox : string`

>Returns or sets the post office box number associated with the contact's home address.

---

**HomeAddressState** property

`property HomeAddressState : string`

>Returns or sets the state of the contact's home address.

---

**HomeAddressStreet** property

`property HomeAddressStreet : string`

>Returns or sets the street name address of the contact's home address.

---

**HomeFaxNumber** **property**

`property HomeFaxNumber : string`

>Returns or sets the contact's home fax number.

See also: [HomeTelephoneNumber](#)

---

**HomeTelephoneNumber** **property**

`property HomeTelephoneNumber : string`

>Returns or sets the contact's primary home telephone.

See also: [HomeFaxNumber](#)

---

**NickName** **property**

`property NickName : string`

>Returns or sets the nickname of the contact.

---

**Spouse** **property**

`property Spouse : string`

>Returns or sets the name of the contact's spouse.

See also: [Children](#)

# Chapter 5: Working with Word

This chapter provides information on OfficePartner's ability to control Microsoft Word. The TOpWord component, and its various supporting classes, encapsulate and provide access to the functionality of the Word Automation objects they represent.

## The Application object

The Word Application object is the top-level object in Word's object model. The Application object can determine or specify application-level properties or execute application-level methods. The Application object is also the entry point into the rest of the Word object model.

Like other Office application object models, the Word Application object exposes several properties that can work with a currently active Word object. The Application object exposes Word documents, bookmarks, ranges, tables, shapes, and hyperlinks, each of which exposes its own functionality. For example, the Documents property returns the Documents collection that contains all the currently open Document objects.

In OfficePartner, the Word Application object is represented and controlled by the TOpWord component. Congruent to the Word Application Object, TOpWord is the primary component in the OfficePartner Word VCL class hierarchy.

## The Document object

In the Word object model, the Document object appears just below the Application object. The Document object represents an open document (\*.doc) file and provides control of the same. All Document objects are contained in the Application object's Documents collection. The Documents collection has a Count property which tells how many visible and hidden documents are open. The Word Document object is represented and controlled by the TOpWordDocument class. A collection of these is maintained in the TOpWordDocuments class.

Most of the work you will do in Word will be within the context of a document. A document contains words, sentences, paragraphs, sections, headers and footers, tables, fields, controls, images, shapes, hyperlinks, and more. The main idea is to specify a part of the document with a Range object, and then to do something with that range. This may involve, for example, adding or removing text or formatting words or characters. A new document can be opened and added to the collection with the Add method of TOpWordDocuments.

## The Document Range object

A Range object represents a contiguous area in a document, defined by a starting character position and an ending character position. The contiguous area can be as small as the insertion point or as large as the entire document. The characters in a Range object include

nonprinting characters, such as spaces, carriage returns, and paragraph marks.

It is probably easiest to think of the Range object as your handle to the Word element you want to work with. A Range object provides many methods to manipulate its text: Select, Insert, Move, Find, Copy, Paste, Replace, Delete, Format, CheckGrammar, CheckSpelling, etc. You can maintain as many Range objects as desired.

## The Document Bookmark Object

Bookmarks are used to mark a location in a document or as a container for text in a document. A Bookmark object is similar to a Range object in that it represents a contiguous area in a document. However, the Bookmark object is persistent. You can give the Bookmark object a name, which is saved with the document when the document is closed. The Document Bookmark object is represented and controlled by the TOpDocumentBookmark class. A collection of these is maintained in the TOpDocumentBookmarks class. All Bookmark objects are contained in the Document object's Bookmarks collection. A bookmark is added by using the Add method of TOpWordBookmarks.

## The Document Tables Object

Each Word document can contain a collection of tables. A Table object represents a grid that is made up of rows and columns of cells that can be filled with text and graphics. A Table object provides numerous methods to manipulate a table in a Word document such as, Format, Sort, Convert to text, and access cell data. The Document Table object is represented and controlled by the TOpDocumentTable class. A collection of these is maintained in the TOpDocumentTables class. All Table objects are contained in the Document object's Tables collection. A table is added by using the Add method of TOpWordTables.

## The Document Hyperlink Object

Each Word document can contain a collection of hyperlinks. A HyperLink object represents a hyperlink in a Word document. The Document Hyperlink object is represented and controlled by the TOpDocumentHyperlink class. A collection of these is maintained in the TOpDocumentHyperlinks class. All Hyperlink objects are contained in the Document object's Hyperlinks collection. A hyperlink is added by using the Add method of TOpWordHyperlinks.

---

# Microsoft Word Tutorial

The following sections will guide you through various steps in the process of opening and closing a Word document and manipulating the text within the document.

## Starting Word

Setting the TOpWord.Connected property to true starts the Word automation server. This can be done at run time or at design-time from the Delphi or C++Builder IDE object inspector. Setting the Visible property to true will activate the Word Application window.

```
procedure TForm1.btnStartWordClick(Sender: TObject);
begin
  OpWord1.Connected := True;
  OpWord1.Visible := True;
end;
```

## Opening a new Word document

Opening a new Word document is simply a matter of creating a Document object. This can be done in two ways: at run time by calling the TOpWord.Documents collection's Add method, or at design-time using the Documents property editor from the Delphi/C++Builder IDE object inspector.

The following code illustrates how to open a new document at run time and add some text:

```
procedure TForm1.btnNewDocClick(Sender: TObject);
var
  NewDoc : TOpWordDocument;
begin
  OpWord1.Connected := True;
  NewDoc := OpWord1.NewDocument;
  NewDoc.Insert('some text', True);
  OpWord1.Visible := True;
end;
```

## Opening an existing Word document

To open an existing Word document, first create a new Document object as outlined in the previous section. Then specify the document filename by setting the TOpWordDocument.DocFile property. If the Word application window is visible, a new window is created which contains the contents of the document.

The following code illustrates how to open an existing document at run time:

```
procedure TForm1.btnExistingDocClick(Sender: TObject);
var
  ExistingDoc : TOpWordDocument;
begin
  OpWord1.Connected := True;
  ExistingDoc := OpWord1.OpenDocument(
    'C:\My Documents\SomeValidFile.Doc');
  OpWord1.Visible := True;
end;
```

### Saving and closing a Word document

You can save a document with the TOpWordDocument's Save or SaveAs methods. To close a document, simply free the Document object representing the document. When the Document object is destroyed, it is also removed from the Documents collection automatically.

The following code illustrates how to save and close the active document at run time:

```
procedure TForm1.btnSaveAndCloseClick(Sender: TObject);
begin
  WorkingDoc := OpWord1.Documents[0];
  OpWord1.ActiveDocument.Save;
  OpWord1.ActiveDocument.Free;
end;
```

### Finding and replacing text in a Word document

You can find and replace text in a Word document with the TOpWordDocument's Find, FindNext, and Replace methods. Find starts the search at the beginning of the document, and FindNext starts from the match found. Replace provides the option of replacing a single match or all matches found in the document.

The following code illustrates how to replace text in a document at run time:

```
procedure TForm1.btnReplaceClick(Sender: TObject);
const
  SampleText = 'Word makes writing easy';
var
  Doc : TOpWordDocument;
begin
  OpWord1.Connected := True;
  Doc := OpWord1.NewDocument;
  Doc.Insert(SampleText, True);
  OpWord1.Visible := True;
  Doc.Replace('Word', 'OfficePartner',
              True, roReplaceAll);
  Doc.Replace('writing', 'automating Word',
              True, roReplaceAll);
end;
```

### Accessing a Word automation object

While OfficePartner provides a component-based framework for encapsulating many common Word automation tasks, it is inevitable that you will require automation functionality that is not currently encapsulated within OfficePartner. The type libraries for the Word Automation objects contain literally thousands of methods and properties. Every OfficePartner class that represents a Word Automation interface provides a simple property to get that interface.

For the TOpWord component, the Server property provides access to the main Automation CoClass interface. Through this property, a developer has access to any methods or properties of a particular Word Application's main interface (commonly called Application or \_Application). The hierarchy of classes underneath TOpWord provides a VCL wrapper to the interfaces contained within this main interface. These interfaces can be easily accessed by a property named AsXXX.

For example, the following code illustrates how to access a Document object to set the documents Track Changes option:

```
procedure TForm1.btnTrackRevisionsClick(Sender: TObject);
var
  Doc : TOpWordDocument;
  IDoc : Document;
begin
  OpWord1.Connected := True;
  Doc := OpWord1.NewDocument;
  IDoc := Doc.AsDocument;
  IDoc.TrackRevisions := True;
  OpWord1.Visible := True;
end;
```

### Adding a table to a Word document

There are many ways to add, populate, and format a table of data in a document. One method is to simply create a Table object via the Document's Tables object, and manipulate the table's cells directly. Other ways are to use a TOpEventModel that fires events that request data for a cell when required, or use a TOpDataSetModel to populate with data from a database.

The following code illustrates how to use a Table object to create and populate a 3x3 table with a header row:

```
procedure TForm1.btnAddTableClick(Sender: TObject);
var
  Doc : TOpWordDocument;
  IDoc : Document;
  ITable : Table;
  i, j : Integer;
begin
  OpWord1.Connected := True;
  Doc := OpWord1.NewDocument;
  Doc.Insert('Populating a Table directly through the Table
```

```
interface', True);
IDoc := Doc.AsDocument;

{ Create 4x3 table after last sentence. 1st row is header }
ITable := IDoc.Tables.AddOld(IDoc.Characters.Last, 4, 3);
ITable.AllowAutoFit := True;

{ Populate header row }
ITable.Cell(1, 1).Range.Text := 'Col 1';
ITable.Cell(1, 1).Range.Bold := 1;
ITable.Cell(1, 2).Range.Text := 'Col 2';
ITable.Cell(1, 2).Range.Bold := 1;
ITable.Cell(1, 3).Range.Text := 'Col 3';
ITable.Cell(1, 3).Range.Bold := 1;

{ Populate table data }
for j := 1 to 3 do
  for i := 1 to 3 do
    ITable.Cell(j+1, i).Range.Text := IntToStr(i + j);

ITable.Columns.AutoFit;
OpWord1.Visible := True;
end;
```





# TOpWord Component

The TOpWord component represents an instance of the Microsoft Word Application. This component is used to control the Word Application and contains numerous properties that control the application-wide aspects of Word. New and existing Documents can be opened using the NewDocument and OpenDocument methods. The ActiveDocument property provides access to the document that is currently active. The Documents collection property provides access to all the open documents.

## Hierarchy

TComponent (VCL)

TOpBaseComponent(OpShared)

① TOpOfficeComponent (OpShared) 443

TOpWordbase (OpWord2k)

TOpWord (OpWord)

5

## Properties

ActiveDocument	Documents	ServerTop
BrowseExtraFileTypes	EnableCancelKey	ServerWidth
Caption	① MachineName	StartupPath
① ClientState	① OfficeVersion	UserAddress
① Connected	① OfficeVersionStr	UserInitials
DefaultSaveFormat	① PropDirection	UserName
DefaultTableSeparator	PrintPreview	Version
DisplayAlerts	ScreenUpdating	Visible
DisplayRecentFiles	Server	WindowState
DisplayScreenTips	ServerHeight	
DisplayScrollBars	ServerLeft	

## Methods

AddConnectListener	① GetAppInfo	OpenDocument
① CoCreate	① GetFileInfo	① RemoveConnectListener
① Create	① HandleEvent	
① CreateItem	NewDocument	

## Events

BeforeDocumentClose	OnDocumentChange	OnWindowActivate
BeforeDocumentPrint	OnDocumentOpen	OnWindowDeactivate
BeforeDocumentSave	OnNewDocument	OnWindowSelectionCha...
BeforeWindowDoubleClick	OnQuit	
BeforeWindowRightClick	OnStartup	

## Reference Section

5

### ActiveDocument

run-time, read-only property

property ActiveDocument : TOpWordDocument

↳ Identifies the active document.

Read ActiveDocument to obtain the TOpWordDocument representing the open document that has focus. If no documents are open a nil pointer is returned.

See also: TOpWordDocument

### BeforeDocumentClose

event

property BeforeDocumentClose : TOpOnDocumentBeforeClose

```
TOpOnDocumentBeforeClose = procedure(Sender : TObject;
  const Doc : Document; var Cancel : WordBool) of object;
```

↳ Defines an event handler that is called just before a document is closed.

This event can be used to perform last minute processing of the document before it is closed. Closing the document can be cancelled by setting Cancel to True.

Doc identifies the Word Document object.

Available only with Word 2000.

## BeforeDocumentPrint

event

```
property BeforeDocumentPrint : TOpOnDocumentBeforePrint  
TOpOnDocumentBeforePrint = procedure(  
  Sender : TObject; const Doc : Document;  
  var Cancel : WordBool) of object;
```

↳ Defines an event handler that is called just before a document is printed.

This event can be used to perform last minute processing of the document before printing. The print operation can be cancelled by setting Cancel to True.

Doc identifies the Word Document object.

5

Available only with Word 2000.

## BeforeDocumentSave

event

```
property BeforeDocumentSave : TOpOnDocumentBeforeSave  
TOpOnDocumentBeforeSave = procedure(Sender : TObject;  
  const Doc : Document; var SaveAsUI : WordBool;  
  var Cancel : WordBool) of object;
```

↳ Defines an event handler that is called just before a document is saved.

This event can be used to perform last minute processing of the document before it is saved. The save operation can be cancelled by setting Cancel to True. To invoke the Word Save As dialog, set SaveAsUI to True.

Doc identifies the Word Document object.

Available only with Word 2000.

## BeforeWindowDoubleClick

event

```
property BeforeWindowDoubleClick : TOpOnWindowBeforeDoubleClick  
TOpOnWindowBeforeDoubleClick = procedure(Sender : TObject;  
  const Sel : Selection; var Cancel : WordBool) of object;
```

↳ Defines an event handler that is called just before a double-click on the specified document window is processed.

This event can be used to override the default Word double-click handling by setting Cancel to True.

Sel identifies the Word Selection range object.

Available only with Word 2000.

## BeforeWindowRightClick

event

```
property BeforeWindowRightClick : TOpOnWindowBeforeRightClick  
TOpOnWindowBeforeRightClick = procedure(  
  Sender : TObject; const Sel : Selection;  
  var Cancel : WordBool) of object;
```

- ↳ Defines an event handler that is called just before a right-click on a document window is processed.

This event can be used to override the default Word context menu operation by setting Cancel to True.

5

Sel identifies the Word Selection range object.

Available only with Word 2000.

## BrowseExtraFileTypes

property

```
property BrowseExtraFileTypes : WideString
```

Default: Empty string

- ↳ Specifies other types of files that can be browsed with Word.

For example, set BrowseExtraFileTypes to “text/html” to allow hyperlinked HTML files to be opened in Word (instead of the default Internet browser).

## Caption

property

```
property Caption : string
```

Default: Empty string

- ↳ Specifies the text string displayed in the caption of the application window.

To change the caption of the application window to the default text, set Caption to an empty string.

```
property DefaultSaveFormat : WideString
```

Default: Empty string

↳ Specifies the default format that will appear in the Save As dialog box.

The string used with this property is the file converter class name. The default value for this property (an empty string) will cause Word Document to be the default format in the Save As dialog box.

The following class names will result in these internal Word formats being the default formats in the Save As dialog box:

**Table 1:**

<b>Class name</b>	<b>Document type</b>
" "	Word Document
"Dot"	Document Template
"Text"	Text Only
"CRTText"	Text Only with Line Breaks
"8Text"	MS-DOS Text
"8CrText"	MS-DOS Text with Line Breaks
"Rtf"	Rich Text Format
"Unicode"	Unicode Text

See also: TOpWordDocument.Save, TOpWordDocument.SaveAs

**DefaultTableSeparator****property**

```
property DefaultTableSeparator : WideString
```

Default: Empty string

↳ Specifies the single character used to separate text into cells when text is converted to a table.

**DisplayAlerts****property**

```
property DisplayAlerts : TOpWdAlertLevel
```

```
TOpWdAlertLevel = (wdalNone, wdalMessageBox, wdalAll);
```

Default: wdalNone

↳ Specifies the way certain alerts and messages are handled while a macro is running.

Set DisplayAlerts to wdalNone when no alerts or message boxes are to be displayed. Setting DisplayAlerts to wdalMessageBox indicates that only message boxes are displayed; errors are trapped and returned to the macro. If DisplayAlerts is set to wdalAll, then all message boxes and alerts are displayed; errors are returned to the macro.

**DisplayRecentFiles****property**

```
property DisplayRecentFiles : WordBool
```

Default: False

↳ Specifies whether to display recently used files in the File Menu.

**DisplayScreenTips****property**

```
property DisplayScreenTips : WordBool
```

Default: False

↳ Specifies whether screen tips are displayed.

If DisplayScreenTips is True, comments, footnotes, endnotes, and hyperlinks are displayed as tips. Text marked as having an associated comment is highlighted.

## **DisplayScrollBars**

**property**

**property DisplayScrollBars : WordBool**

**Default:** False

↳ Specifies whether to display scroll bars or not.

Setting **DisplayScrollBars** to True causes both horizontal and vertical scroll bars to be displayed in all Word application windows. Setting **DisplayScrollBars** to False turns off all scroll bars in all Word application windows.

## **Documents**

**property**

**property Documents : TOpWordDocuments**

↳ Contains a collection of all the documents that are currently open.

**Documents** is a **TOpNestedCollection** descendent which allows **TOpWordDocument(s)** to be stored and restored with a form. Another nice feature is the ability to work with documents whether **Connected** is set to True or False. If the **TOpWord** component is connected to a running instance of Word, document changes to are immediately reflected in Word. If the component is not connected, changes may still be made to the **Documents** property which will be applied when the component connects to Word.

For example, if the VCL program adds two Document(s) to the **Documents** property of **TOpWord** while the **TOpWord** component is not connected to Word, Word will have the two documents open and displayed when **TOpWord** connects to Word with no further coding required.

See also: **NewDocument**, **OpenDocument**, **TOpWordDocuments**

## **EnableCancelKey**

**property**

**property EnableCancelKey : TOpWdEnableCancelKey**

**TOpWdEnableCancelKey = (wdeckDisabled, wdeckInterrupt);**

**Default:** **wdeckDisabled**

↳ Specifies the way that Word handles Ctrl+Break user interruptions.

Setting **EnableCancelKey** to **wdeckDisabled** prevents Ctrl+Break from interrupting a macro. Setting **EnableCancelKey** to **wdeckInterrupt** allows a macro to be interrupted by Ctrl+Break.

⚠ Caution: If you use **wdeckDisabled**, there's no way to interrupt a runaway loop or other non-terminating code.

**NewDocument****method**

```
function NewDocument : TOpWordDocument;
```

- ↳ Creates a new blank document.

Use NewDocument to create a new document. The function creates a TOpWordDocument instance that represents the document and adds it to the Documents collection. The TOpWordDocument instance is returned from the function call.

See also: Documents, OpenDocument, TOpWordDocument

**OnDocumentChange****event**

```
property OnDocumentChange : TNotifyEvent
```

- ↳ Defines an event handler that is called when the active document is changed.

This event is fired when a new document is created, when an existing document is opened, or when another document is made the active document.

See also: ActiveDocument

**OnDocumentOpen****event**

```
property OnDocumentOpen : TOpOnDocumentOpen
```

```
TOpOnDocumentOpen = procedure(
  Sender : TObject; const Doc : Document) of object;
```

- ↳ Defines an event handler that is called when a document is opened.

Doc is a pointer to the Word Document object.

**OnNewDocument****event**

```
property OnNewDocument : TOpOnNewDocument
```

```
TOpOnNewDocument = procedure(
  Sender : TObject; const Doc : Document) of object;
```

- ↳ Defines an event handler that is called when a new document is created.

Doc is a pointer to the new Word Document object.

See also: NewDocument

## OnQuit

event

```
property OnQuit : TNotifyEvent
```

Defines an event handler that is called when the user quits Word.

This event allows the application settings to be adjusted before the application closes. For example, you may wish to reset various options to default values for when Word is started again.

See also: OnStartup

## OnStartup

event

```
property OnStartup : TNotifyEvent
```

Defines an event handler that is called when the Word application is started.

This event can be used to set various application values prior to displaying the Word application window.

See also: OnQuit

## OnWindowActivate

event

```
property OnWindowActivate : TOpOnWindowActivate
```

```
TOpOnWindowActivate = procedure(
  Sender : TObject; const Doc : Document;
  const Wn : WordWindow) of object;
```

Defines an event handler that is called when any document window is activated.

Doc identifies the Word Document object for the document displayed in the activated window. Wn identifies the WordWindow object being activated.

See also: OnWindowDeactivate

## OnWindowDeactivate

event

```
property OnWindowDeactivate : TOpOnWindowDeactivate
```

```
TOpOnWindowDeactivate = procedure(
  Sender : TObject; const Doc : Document;
  const Wn : WordWindow) of object;
```

Defines an event handler that is called when any document window is deactivated.

Doc identifies the Document object for the document displayed in the deactivated window. Wn identifies the WordWindow object being deactivated.

See also: OnWindowActivate

## **OnWindowSelectionChange**

**event**

```
property OnWindowSelectionChange : TOpOnWindowSelectionChange
```

```
TOpOnWindowSelectionChange = procedure(
  Sender : TObject; const Sel : Selection) of object;
```

- ↳ Defines an event handler that is called when the selection changes in the active document window.

Sel contains the new Selection object.

## **OpenDocument**

**method**

```
function OpenDocument(
  const FileName : TFileName) : TOpWordDocument;
```

- ↳ Opens the document specified by file name.

Use OpenDocument to open an existing document on disk. The function creates a TOpWordDocument instance that represents the document and adds it to the Documents collection. The TOpWordDocument instance is returned from the function call.

See also: Documents, NewDocument

## **PrintPreview**

**property**

```
property PrintPreview : WordBool
```

Default: False

- ↳ Switches Word to or from Print Preview mode.

When PrintPreview is set to True, each page of a document is displayed as it will be printed.

## **ScreenUpdating**

**property**

```
property ScreenUpdating : WordBool
```

Default: False

- ↳ Controls updating of the Word application screen.

Setting ScreenUpdating to False, allows you to suspend document screen repainting while changes are made to the document. Toolbars remain visible and Word still allows the procedure to display or retrieve information using status bar prompts, input boxes, dialog boxes, and message boxes.

## **Server**

**run-time, read-only property**

```
property Server : _Application  
_Application = interface(IDispatch);
```

↳ Returns the Automation interface for the Word Server.

This property allows the developer direct access to the Word COM interface that the component is using. This allows the developer a way to access the myriad of features in Word which OfficePartner does not directly expose.

It is not feasible to document all methods and properties of this interface. You should consult the source code and any additional information (such as MSDN) prior to invoking methods of this interface.

5

## **ServerHeight**

**property**

```
property ServerHeight : Integer
```

Default: 480

↳ The Height (in pixels) of the Word application window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerLeft, ServerTop, ServerWidth

## **ServerLeft**

**property**

```
property ServerLeft : Integer
```

Default: 0

↳ The Left position (in pixels) of the Word application window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerTop, ServerWidth

## **ServerTop**

**property**

```
property ServerTop : Integer
```

Default: 0

↳ The Top position (in pixels) of the Word application window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerWidth

---

**ServerWidth** property

property ServerWidth : Integer

Default: 640

☞ The Width (in pixels) of the Word application window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerTop

---

**StartupPath** property

property StartupPath : WideString

Default: Empty string

☞ Specifies the complete path to the Startup folder for the Word application.

The Startup folder is where Templates and add-ins are located. These are automatically loaded when the Word application starts.

---

**UserAddress** property

property UserAddress : WideString

Default: Empty string

☞ Specifies the user's mailing address.

UserAddress is used as a return address on envelopes.

See also: UserName, UserInitials

---

**UserInitials** property

property UserInitials : WideString

Default: Empty string

☞ Specifies the user's initials.

UserInitials is used by the Word application to construct comment marks.

See also: UserAddress, UserName

---

**UserName** property

```
property UserName : WideString
```

Default: Empty string

↳ Specifies the user's name.

UserName is used by the Word application for envelopes and to identify the document's author.

See also: UserAddress, UserInitials

---

**WindowState** property

```
property WindowState : TOpWdWindowState
```

```
TOpWdWindowState = (
    wdwsNormal, wdwsMaximized, wdwsMinimized);
```

↳ Specifies the state of the active document window.

The state of an inactive document window cannot be set. Use the Activate method of the TOpWordDocument to activate an inactive document prior to setting the window state.

See also: TOpNestedCollectionItem.Activate, TOpOfficeComponent.Connected

# TOpWordDocuments Class

The TOpWordDocuments collection represents all of the open documents of a TOpWord instance. This class encapsulates and provides access to all of the open documents. The Add method is used to create a new document by creating a new TOpWordDocument object and adding it to the collection. The Items property provides access to individual TOpWordDocument objects in the collection.

## Hierarchy

TCollection (VCL)

① TOpNestedCollection (OpShared) 434

TOpWordDocuments (OpWord)

## Properties

① ItemName

Items

① ParentItem

① RootCollection

① RootComponent

## Methods

Add

① Create

① FindItem

① ForEachItem

## Reference Section

### Add

### method

function Add : TOpWordDocument;

↳ Adds a new document to the collection of open documents.

Calling Add opens a new document and adds it to Items. To close the document, call the Free method of the TOpWordDocument.

See also: Items, TOpWordDocument

Items	property
-------	----------

```
property Items[index : integer] : TOpWordDocument
```

↳ Items provides access to all the documents the parent document has open.

Use the Items property to access individual documents. Items is the default array property for TOpWordDocuments. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
Excel1.Workbooks[0].SaveAs('Default.Txt');  
Excel1.Workbooks.Items[0].SaveAs('Default.txt');
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add, TOpWordDocument

# TOpWordDocument Class

The TOpWordDocument class is a persistent class that represents a Word Document object. This is the fundamental working class used for Word automation. The TOpWordDocument is used to control an individual document and provides access to the contents of the document and all the word processing functionality available in the Word application. The document can be saved by using the Save and SaveAs methods. Closing the document is accomplished by simply destroying the TOpWordDocument object. Text is inserted in the document by the Insert and InsertStrings methods, and text is searched and replaced using the Find, FindNext, and Replace methods. The Bookmarks, Hyperlinks, Tables, Ranges, and Shapes properties are collections containing objects embedded in the document. The AsDocument property provides access to the Word Document interface where the full functionality of Word is available.

## Hierarchy

TCollectionItem (VCL)

- ① TOpNestedCollectionItem (OpShared)438
  - TOpWordDocument (OpWord)

## Properties

AsDocument	MailMerge	SaveFormsData
AutoHyphenation	Name	SaveSubsetFonts
Bookmarks	① ParentCollection	Shapes
CodeName	① ParentItem	ShowGrammaticalErrors
ConsecutiveHyphensLimit	PrintFormsData	ShowRevisions
DefaultTabStop	PrintPostScriptOverText	ShowSpellingErrors
DocFile	PropDirection	ShowSummary
DocumentKind	PrintRevisions	SpellingChecked
EmbedTrueTypeFonts	RangeEnd	① SubCollection
GrammarChecked	RangeLimit	① SubCollectionCount
① Intf	RangeStart	SummaryLength
HasRoutingSlip	ReadOnlyRecommended	SummaryViewMode
HyperLinks	① RootCollection	Tables
HyphenateCaps	① RootComponent	TrackRevisions
HyphenationZone	Saved	UpdateStylesOnOpen

UserControl	● VerbCount
● Verb	ViewType

## Methods

● Activate	Find	Replace
● Connect	FindNext	Save
● Create	Insert	SaveAs
Destroy	InsertStrings	SendMailWithOutLook
ExecuteMailMerge	PopulateMailMerge	
● ExecuteVerb	Print	

## Reference Section

---

<b>Activate</b>	<b>method</b>
-----------------	---------------

---

```
procedure Activate; override;
```

↳ Activates the document if Word is visible and connected.

If Word is not visible or is not connected, then calling this method will have no effect. If Word is connected and visible, then the Word window will receive focus, and the document will be brought to the front. The following code demonstrates how to activate the second document in the collection:

```
OpWord1.Documents[1].Activate;
```

---

<b>AsDocument</b>	<b>run-time, read-only property</b>
-------------------	-------------------------------------

---

```
property AsDocument : _Document
  _Document = interface(IDispatch)
```

↳ Contains the automation interface for the Document.

Much of the functionality of the document has been exposed through the interface of the TOpDocument class. It may sometime be necessary to expand on what OfficePartner has encapsulated and this is the interface you would use to do just that. This interface provides access to virtually all of the document functionality that Microsoft has elected to expose. It is not practical to document the entire functionality of this interface and it is suggested that you consult the source code and any additional information published by Microsoft (MSDN for example).

The following code shows how AsDocument is used to repaginate the document for example:

```
OpWord1.ActiveDocument.AsDocument.Repaginate;
```

### **AutoHyphenation**

**property**

```
property AutoHyphenation : WordBool
```

Default: False

- ↳ Enables or disables automatic hyphenation.

5

When AutoHyphenation is True, Word automatically hyphenates text as the user types and automatically inserts hyphens where they are needed in the document. If the document line breaks are changed, then Word rehyphenates the document.

### **Bookmarks**

**property**

```
property Bookmarks : TOpDocumentBookmarks
```

- ↳ The collection that represents all the bookmarks in the document.

This property provides access to all the bookmarks currently set. To add a bookmark, call the Add method of TOpDocumentBookmarks.

See also: TOpDocumentBookmarks

### **CodeName**

**read-only property**

```
property CodeName : WideString
```

- ↳ Returns the document's code name.

CodeName is the name for the module that houses event macros for a document. The default name for the module is "ThisDocument".

## **ConsecutiveHyphensLimit**

**property**

**property ConsecutiveHyphensLimit : Integer**

**Default: 0**

↳ Specifies the maximum number of consecutive lines that can end with hyphens

Setting **ConsecutiveHyphensLimit** to 0 allows any number of consecutive lines to end with hyphens.

## **DefaultTabStop**

**property**

**property DefaultTabStop : Single**

**Default: 0**

↳ Specifies the interval (in points) between the default tab stops.

## **Destroy**

**destructor**

**destructor Destroy;**

↳ Closes the open document, removes itself from its parent collection and frees the object.

Freeing the **TOpWordDocument** instance automatically closes the document and removes it from the parent **TOpWordDocuments** collection.

See also: **TOpWordDocuments**

## **DocFile**

**property**

**property DocFile : TFileName**

↳ Specifies the file name of the document.

Changing **DocFile** will cause the specified document to be opened. The current open document is not saved.

**DocumentKind****property**

```
property DocumentKind : TOpWdDocumentKind
TOpWdDocumentKind = (
    wddkNotSpecified, wddkLetter, wddkEmail);
```

Default: wddkNotSpecified

- ↳ Specifies the document type used for automatic formatting.

**EmbedTrueTypeFonts****property**

```
property EmbedTrueTypeFonts : WordBool
```

Default: False

- ↳ Specifies whether or not to embed TrueType fonts when the document is saved.

This allows others to view the document with the same fonts that were used to create it.

**ExecuteMailMerge****method**

```
procedure ExecuteMailMerge;
```

- ↳ Generates the final document with all the merged fields.

ExecuteMailMerge should be called after a successful execution of the PopulateMailMerge procedure.

See also: MailMerge, PopulateMailMerge

**Find****method**

```
function Find(FindText : string; Forward : Boolean) : Boolean;
```

- ↳ Performs a simple search for the specified text.

If an occurrence of the text is found, that occurrence is selected in the document window. Forward specifies the direction of search; True causes the search to start at the top of the document and False causes the search to start from the bottom. The method returns True if an occurrence is found.

Find will always find the first occurrence. To continue searching, use the FindNext method.

For access to the full range of search criteria, use the Find property of a specific Range in the document.

See also:FindNext, Replace

## **FindNext**

## **method**

```
function FindNext : Boolean;
```

- ↳ Continues the simple search begun by Find.

The search is resumed starting immediately after the position in the document where the last occurrence of the text specified by the initial Find method call was found. The method returns True if an occurrence is found.

For access to the full range of search criteria, use the Find property of a specific Range in the document.

See also: FindNext, Replace

## **GrammarChecked**

## **property**

```
property GrammarChecked : WordBool
```

Default: False

- ↳ Indicates whether or not a grammar check has been run.

A possible grammatical error is identified in Word by a wavy, green line under the phrase in question. When GrammarChecked returns False, it indicates that some or all of the document has not been checked. Setting GrammarChecked to False causes the document to be rechecked.

Grammatical error marks can be displayed or hidden by setting the ShowGrammaticalErrors property appropriately.

See also: ShowGrammaticalErrors, SpellingChecked

## **HasRoutingSlip**

## **property**

```
property HasRoutingSlip : WordBool
```

Default: False

- ↳ Specifies whether or not the document has a routing slip attached to it.

Setting HasRoutingSlip to True creates a routing slip; setting it to False deletes the routing slip.

---

**HyperLinks** property

```
property HyperLinks : TOpDocumentHyperLinks
```

- ↳ The collection that represents all the hyperlinks in the document.

This property provides access to all the hyperlinks in the doocument. To add a hyperlink, call the Add method of TOpDocumentHyperlinks.

See also: TOpDocumentHyperlinks

---

**HyphenateCaps** property

```
property HyphenateCaps : WordBool
```

Default: False

- ↳ Specifies whether or not words in all capital letters can be hyphenated.

---

**HyphenationZone** property

```
property HyphenationZone : Integer
```

Default: 1

- ↳ Specifies the width of the hyphenation zone, in points.

The hyphenation zone is the maximum amount of space that Word leaves between the end of the last word in a line and the right margin.

---

**Insert** method

```
procedure Insert(Text : WideString; AtEnd : Boolean);
```

- ↳ Inserts the specified line of text into the document.

Text is the text to insert. AtEnd specifies whether or not to add the block of text to the end of the document or insert the block of text based on the RangeLimit, RangeStart and RangeEnd properties.

To Insert a block of text, use the InsertStrings method.

See also: InsertStrings, RangeEnd, RangeLimit, RangeStart

## InsertStrings

method

```
procedure InsertStrings(Lines : TStrings; AtEnd : Boolean);
```

↳ Inserts a block of text into the document.

Lines is the list of the lines of a text block. AtEnd specifies whether or not to add the block of text to the end of the document or insert the block of text based on the RangeLimit, RangeStart and RangeEnd properties.

To Insert a single line of text, use the InsertString method.

See also: InsertString, RangeEnd, RangeLimit, RangeStart

## MailMerge

property

```
property MailMerge : TOpMailMerge
```

↳ Specifies the associated TOpMailMerge object.

The MailMerge object represents a Document MailMerge object and identifies the mail merge data source. A TOpMailMerge object is automatically created and assigned to this property when an instance of TOpWordDocument is created.

To specify the Office Model used for the data source, set the OfficeModel property of the TOpMailMerge object. For example, the following code specifies a TOpEventModel component for the active document's mail merge OfficeModel:

```
OpWord1.ActiveDocument.MailMerge.OfficeModel := OpEventModel1;
```

See also: TOpDataSetModel, TOpEventModel, TOpMailMerge

## Name

read-only property

```
property Name : string
```

↳ Returns the document's file name.

Use Name to determine the filename of an open document.

See also: DocFile

## PopulateMailMerge

method

```
procedure PopulateMailMerge;
```

- ↳ Obtains data from the current data source record for the document's fields.

The data source is identified by the `OfficeModel` property of the `MailMerge` object which must be assigned prior to calling `PopulateMailMerge`.

See also: `ExecuteMailMerge`, `MailMerge`

## Print

method

5

```
procedure Print;
```

- ↳ Prints the entire document.

The document is printed according to the print options specified by the `PrintFormsData`, `PrintPostScriptOverText`, and `PrintRevisions` properties. To print with different options, use the `PrintOut` method of the Word Document object.

See also: `PrintFormsData`, `PrintPostScriptOverText`, `PrintRevisions`

## PrintFormsData

property

```
property PrintFormsData : WordBool
```

Default: False

- ↳ Specifies whether or not Word prints onto a preprinted form.

When the document is printed, if `PrintFormsData` is true then Word prints only the data from the form fields but not the form itself.

See also: `Print`, `PrintPostScriptOverText`, `PrintRevisions`

## PrintPostScriptOverText

property

```
property PrintPostScriptOverText : WordBool
```

Default: False

- ↳ Specifies whether or not PostScript commands in a document are to be printed on top of text.

If the document contains no PRINT fields, this property has no effect.

See also: `Print`, `PrintFormsData`, `PrintRevisions`

## PrintRevisions

property

property PrintRevisions : WordBool

Default: False

↳ Specifies whether or not revision marks are printed with the document.

With change-tracking enabled (TrackRevisions = True), the document can contain revision marks which indicate where insertion, deletion, and other editing changes have been made. To include these revision marks in the printed document, set PrintRevisions to True before calling Print.

See also: Print, PrintFormsData, PrintPostScriptOverText, TrackRevisions

5

## PropDirection

property

property PropDirection : TOpPropDirection

TOpPropDirection = (pdToServer, pdFromServer)

↳ Specifies whether the document is initialized by the component, or loaded from an existing file.

If PropDirection is set to pdToServer then the document is initialized by the properties of this object. If PropDirection is set to pdFromServer, then the properties and methods of this object are initialized according to the properties of the document defined by DocFile.

See also: Activate

## RangeEnd

property

property RangeEnd : Integer

Default: 0

↳ Specifies the end of a range used to insert text.

The InsertString and InsertStrings methods add text at the end of a range of objects determined by RangeStart and RangeEnd. RangeEnd specifies the ending paragraph, sentence, word, or character of the range as determined by RangeLimit.

See also: InsertString, InsertStrings, RangeLimit, RangeStart

**RangeLimit****property**

```
property RangeLimit : TOpWdRangeLimit  
  TOpWdRangeLimit = (  
    wdrlParagraphs, wdrlSentences, wdrlWords, wdrlCharacters);
```

Default: wdrlCharacters

↳ Specifies the kind of range object used for inserting text.

The InsertString and InsertStrings methods add text at the end of a range of objects determined by RangeStart and RangeEnd. RangeLimit specifies whether you are referring to a range of paragraphs, sentences, words, or characters.

See also: InsertString, InsertStrings, RangeEnd, RangeStart

**RangeStart****property**

```
property RangeStart : Integer
```

Default: 0

↳ Specifies the start of a range used to insert text.

The InsertString and InsertStrings methods add text at the end of a range of objects determined by RangeStart and RangeEnd. RangeStart specifies the starting paragraph, sentence, word, or character of the range as determined by RangeLimit.

See also: InsertString, InsertStrings, RangeEnd, RangeLimit

<b>ReadOnlyRecommended</b>	<b>property</b>
<code>property ReadOnlyRecommended : WordBool</code>	
↳ Specifies whether or not to display a message box whenever a user opens the document, suggesting that it be opened as read-only.	
<b>Replace</b>	<b>method</b>
<pre>function Replace(FindText, ReplaceText : string;   Forward : Boolean; ReplaceOption : TOpWdReplaceOption) :   Boolean;  TOpWdReplaceOption = (   wdroReplaceNone, wdroReplaceOne, wdroReplaceAll);</pre>	
↳ Replaces text specified by FindText with text specified by ReplaceText.	

Replace performs a simple search for the first occurrence of the text specified by FindText and replaces it with ReplaceText according to the ReplaceOption. The possible values for ReplaceOption are:

**Table 2:**

<b>Setting</b>	<b>Behavior</b>
wdroReplaceNone	Causes the method to behave identically to the Find method. The first occurrence of FindText is merely selected.
wdroReplaceOne	Causes the first occurrence of FindText to be replaced by ReplaceText. A subsequent call to Replace will then locate the next occurrence and so on.
wdroReplaceAll	Causes all occurrences of FindText to be replaced with ReplaceText. Subsequent searches will not find any occurrences of FindText.

For access to the full range of search and replace criteria, use the Find property of a specific Range in the document.

See also: Find, FindNext

**Save****method**

```
procedure Save;
```

↳ Saves the document.

If the document has not been saved before, the Save As dialog box prompts the user for a file name. To determine whether or not the document has changed since the last time it was saved, use the Saved property.

See also: SaveAs, Saved

**SaveAs****method**

```
procedure SaveAs(const FileName : string);
```

↳ Saves the document to a file specified by FileName.

FileName specifies the path for the file. If a document with the specified FileName already exists, the document is overwritten without the user being prompted first.

See also: Save, Saved

**Saved****property**

```
property Saved : WordBool
```

Default: False

↳ Indicates whether or not the document has changed since it was last saved.

Setting Saved to False before closing the document will cause Word to display a prompt to save changes when the document is closed.

See also: Save, SaveAs

**SaveFormsData****property**

```
property SaveFormsData : WordBool
```

Default: False

↳ Indicates whether or not to save only the data entered in a form.

When the document is saved, if SaveFormsData is true, then only the data in the form is saved. In such cases, the data is saved as a tab-delimited record for use in a database.

See also: PrintFormsData

## **SaveSubsetFonts**

**property**

**property SaveSubsetFonts : WordBool**

Default: False

- ↳ Specifies whether or not Word saves a subset of the embedded TrueType fonts with the document.

If fewer than 32 characters of a TrueType font are used in a document, Word embeds the subset (only the characters used) in the document. If more than 32 characters are used, Word embeds the entire font.

See also: Save, SaveAs

5

## **SendMailWithOutLook**

**method**

```
procedure SendMailWithOutLook(WordSubject, SendTo, SendCC,  
    SendBC : string; OutLookComp : TOpOutlook;  
    AsAttachment : Boolean);
```

- ↳ Emails the document via Microsoft Outlook.

Use SendMailWithOutlook to transmit the document as e-mail using the TOpOutlookClient component identified by OutlookComp. WordSubject, SendTo, SendCC, SendBCC specify the text of the corresponding e-mail message header fields.

AsAttachment specifies whether the document is sent as MIME attachment or as the plain text body of the message.

See also: TOpOutlook

## **Shapes**

**property**

**property Shapes : TOpDocumentShapes**

- ↳ The collection that represents all the shapes in the document.

Shapes provides access to Shape objects in the document excluding the headers and footers. These objects can represent drawings, shapes, pictures, OLE objects, ActiveX controls, and text objects.

To access a particular shape in the collection, use the TOpDocumentShapes.Items property. To Add a shape to the collection, use the TOpDocumentShapes.Add method.

See also: TOpDocumentShapes, TOpDocumentShape

**ShowGrammaticalErrors****property**

```
property ShowGrammaticalErrors : WordBool
```

Default: False

↳ Specifies whether or not grammatical errors are marked.

When ShowGrammaticalErrors is True and grammar checking is enabled, then possible grammatical errors are identified by a wavy green line under the word or phrase in question. Setting GrammarChecked to False causes the document to be rechecked.

See also: GrammarChecked, ShowSpellingErrors

**ShowRevisions****property**

```
property ShowRevisions : WordBool
```

Default: False

↳ Specifies whether or not tracked changes are shown on the screen.

Changes are identified by revision marks which show where deletions, insertions, or other editing changes have been made in the document. When change-tracking is enabled (TrackRevisions = True), then setting ShowRevisions to True will cause any revision marks to be displayed on the screen.

See also: PrintRevisions, TrackRevisions

**ShowSpellingErrors****property**

```
property ShowSpellingErrors : WordBool
```

Default: False

↳ Specifies whether or not spelling errors are marked.

When ShowSpellingErrors is True and spell checking is enabled, then possible spelling errors are identified by a wavy red line on the screen. Setting SpellingChecked to False causes the document to be rechecked.

Spelling errors are marked with a wavy red line.

See also: ShowGrammaticalErrors, SpellingChecked

**ShowSummary****property****property ShowSummary : WordBool****Default:** False

↳ Specifies whether or not an automatic summary is displayed.

An automatic summary identifies the key points so the document can be quickly scanned. Setting ShowSummary to True causes an automatic summary to be displayed on the screen according the SummaryLength and SummaryViewMode property settings.

**See also:** SummaryLength, SummaryViewMode

**SpellingChecked****property****property SpellingChecked : WordBool****Default:** False

↳ Indicates whether or not the document has been spell checked.

A possible spelling error is identified in Word by a wavy, red line under the word in question. When SpellingChecked returns False, it indicates that some or all of the document has not been checked. Setting SpellingChecked to False causes the document to be rechecked.

Spelling error marks can be displayed or hidden by setting the ShowSpellingErrors property appropriately.

**See also:** GrammarChecked, ShowSpellingErrors

**SummaryLength****property****property SummaryLength : Integer****Default:** 0

↳ Specifies the length of the automatic summary as a percentage of the document length.

An automatic summary identifies the key points so the document can be quickly scanned. The larger the value contained in SummaryLength, the more detail is included in the automatic summary.

**See also:** ShowSummary, SummaryViewMode

**SummaryViewMode****property**

```
property SummaryViewMode : TOpWdSummaryMode
TOpWdSummaryMode = (wdsmHighlight, wdsmHideAllButSummary,
wdsmInsert, wdsmCreateNew);
```

Default: sdsdHighlight

↳ Specifies how an automatic summary is displayed.

An automatic summary identifies the key points so the document can be quickly scanned. When automatic summary is enabled (ShowSummary = True) it is displayed according to SummaryViewMode settings.

The following describes the behavior caused by the SummaryViewMode settings:

**Table 3:**

<b>Value</b>	<b>Behavior</b>
wdsmHighlight	Highlights the key points of the document and displays the AutoSummarize toolbar
wdsmInsert	Inserts a summary at the beginning of the document
wdsmCreateNew	Creates a new document and inserts the specified summary
wdsmHideAllButSummary	Hides everything except the specified summary and displays the AutoSummarize toolbar

See also: ShowSummary, SummaryLength

**Tables****property**

```
property Tables : TOpDocumentTables
```

↳ The collection that represents the tables in the document.

This property provides access to tables contained in the document. To add a table, call the Add method of TOpDocumentTables, and to access a particular table in the collection, use the Items property of TOpDocumentTables.

See also: TOpDocumentTables

## **TrackRevisions**

**property**

`property TrackRevisions : WordBool`

Default: False

↳ Specifies whether or not changes to the document are tracked.

Changes are identified by revision marks which show where deletions, insertions, or other editing changes have been made in the document. Setting TrackRevisions to True enables change-tracking.

When change-tracking is enabled, any revisions marks can be displayed or hidden by setting the ShowRevisions property appropriately.

To include revision marks when the document is printed, set PrintRevisions to True.

See also: PrintRevisions, ShowRevisions

## **UpdateStylesOnOpen**

**property**

`property UpdateStylesOnOpen : WordBool`

Default: False

↳ Specifies whether or not styles are updated when the document is opened.

When UpdateStyleOnOpen is True, all styles are copied from the attached template into the document when it is opened. Any existing styles in the document that have the same name will be overwritten.

## **UserControl**

**read-only property**

`property UserControl : WordBool`

↳ Identifies whether the document was opened by the user.

When the document is opened programmatically, UserControl is False.

## **ViewType** property

---

```
property ViewType : TOpWdViewType  
  
TwdViewType = (  
    wdvtNormalView, wdvtOutlineView, wdvtPrintView, wdvtPrintPreview,  
    wdvtMasterView, wdvtWebView);
```

- ↳ Specifies the view when text is retrieved from the document's active window.

Changing ViewType doesn't change the display on the screen. Instead, it determines what characters in the document will be included when the range in the active window is retrieved.

# TOpDocumentTables Class

The TOpDocumentTables collection represents tables in document. This class encapsulates and provides access to the tables defined in a document. The Add method is used to create a new TOpDocumentTable object and add it to the collection. The Items property provides access to individual TOpDocumentTable objects in the collection.

## Hierarchy

TCollection (VCL)

- ① TTopNestedCollection (OpOfficeShared)434
- TOpDocumentTables (OpWordClient)

## Properties

① ItemName	① ParentItem	① RootComponent
Items	① RootCollection	

## Methods

Add	① FindItem
① Create	① ForEachItem

## Reference Section

---

<b>Add</b>	<b>method</b>
------------	---------------

---

function Add : TOpDocumentTable;

↳ Adds a new table to the parent Document.

Calling Add creates a new table in the the parent document and adds it to Items To delete a table from the document, call the Free method of TOpDocumentTable.

See also: TOpDocumentTable

Items	property
property Items[index : integer] : TOpDocumentTable	

⌚ Array property containing individual table (TOpDocumentTable) items.

Use the Items property to access individual tables. Items is the default array property for TOpDocumentTable. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpWord1.ActiveDocument.Tables.Items[2].Free;  
OpWord1.ActiveDocument.Tables[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add, TOpDocumentTable

## TOpDocumentTable Class

A TOpDocumentTable item represents a grid that is made up of rows and columns of cells that can be filled with text and graphics. The underlying Table object interface provides numerous methods that can be used to manipulate a table in a Word document such as Format, Sort, Convert to text, and access cell data.

The TOpDocumentTable is designed to be used with a TOpDataModel or TOpEventManager to obtain data for the document table. The PopulateTable method is used to populate new table, and the RepopulateTable method updates the data in an existing table.

### Hierarchy

TCollectionItem (VCL)

① TOpNestedCollectionItem (OpOfficeShared)438

TOpDocumentTable (OpWordClient)

### Properties

- |                    |                      |             |
|--------------------|----------------------|-------------|
| ① Intf             | ① RootCollection     | ① Verb      |
| OfficeModel        | ① RootComponent      | ① VerbCount |
| ① ParentCollection | ① SubCollection      |             |
| ① ParentItem       | ① SubCollectionCount |             |

### Methods

- |           |               |                    |
|-----------|---------------|--------------------|
| Activate  | ① Create      | PopulateDocTable   |
| ① Connect | ① ExecuteVerb | RepopulateDocTable |

## Reference Section

### Activate method

---

```
procedure Activate;
```

↳ Activates a table.

If the Word server is not connected or is not visible, calling this method will have no effect. Otherwise, the document to which this table belongs will receive focus and the table is selected.

5

### OfficeModel property

---

```
property OfficeModel : TOpUnknownComponent
```

↳ Specifies the data provider model for the table.

This property can be assigned to any component that descends from TOpUnknownComponent and implements the IOpModel interface. Specify a TOpDataSetModel to populate the table with values obtained from a data base, or assign a TOpEvent model to use VCL events to provide the table data through event handlers you define.

See also: PopulateDocTable, TOpDataSetModel, TOpEventManager

### PopulateDocTable method

---

```
procedure PopulateDocTable;
```

↳ Fills a Table with data from an Office Model.

When PopulateDocTable is called, data for the table is obtained via the Office Model. The model is iterated and the table will be filled with all rows and columns. Although the PopulateDocTable method can be called at design time (from the Nested Collection Editor), populated values will not be persisted. PopulateDocTable should be called at run time to fill the table from the associated model. PopulateDocTable will supply column headers if they are supported by the Model. If the Model has a DetailModel (for TOpDatasetModels) then the Table will be generated in outline form, showing the master-detail relationship.

Generally, you will need to call PopulateDocTable only once for each table. To update the table values in a previously populated table, call the RepopulateDocTable method.

See also: OfficeModel, RepopulateDocTable, TOpDataSetModel, TOpEventManager

## **RepopulateDocTable**

**method**

```
procedure RepopulateDocTable;
```

↳ Refills a Table with data from an Office Model.

Use RepopulateDocTable to refresh or update the contents of an existing table. This method is essentially identical to the PopulateDocTable except that a new Table object is not created.

See also: PopulateDocTable, TOpDataSetModel, TOpEventManager

# TOpDocumentBookmarks Class

The TOpDocumentBookmarks collection represents all of the Bookmarks of a TOpWordDocument instance. This class encapsulates and provides access to all of the bookmarks set in a document. The Add method is used to create a new TOpDocumentBookmark object and add it to the collection. The Items property provides access to individual TOpDocumentBookmark objects in the collection.

## Hierarchy

TCollection (VCL)

- ① TOpNestedCollection (OpOfficeShared) 434
  - TOpDocumentBookmarks (OpWordClient)

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ① ItemName | ① ParentItem     | ① RootComponent |
| Items      | ① RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ① FindItem    |
| ① Create | ① ForEachItem |

## Reference Section

### Add

### method

function Add : TOpDocumentBookmark;

↳ Adds a new bookmark to the parent document.

Calling Add creates a new bookmark in the the parent document and adds it to Items. To delete a bookmark from the document, call the Free method of TOpDocumentBookmark.

See also: TOpDocumentBookmark

<b>Items</b>	<b>property</b>
--------------	-----------------

```
property Items[index : integer] : TOpDocumentBookmark
```

⌚ Array property containing individual bookmark (TOpDocumentBookmark) items.

Use the Items property to access individual bookmarks. Items is the default array property for TOpDocumentBookmark. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpWord1.ActiveDocument.Bookmarks.Items[2].Free;  
OpWord1.ActiveDocument.Bookmarks[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add, TOpDocumentBookmark

# TOpDocumentBookmark Class

A TOpDocumentBookmark item represents a bookmark in a document. A bookmark is an item or location in a document that can be identified by name for future reference.

## Hierarchy

TCollectionItem (VCL)

① TOpNestedCollectionItem (OpOfficeShared)438

TOpDocumentBookmark (OpWordClient)

## Properties

BookmarkName

① RootCollection

① Verb

① Intf

① RootComponent

① VerbCount

① ParentCollection

① SubCollection

① ParentItem

① SubCollectionCount

## Methods

Activate

① Create

① Connect

① ExecuteVerb

## Reference Section

### Activate

method

procedure Activate;

↳ Activates the bookmark.

If the Word server is not connected or is not visible, calling this method will have no effect. Otherwise, the document to which this bookmark belongs will receive focus and the bookmark will be selected.

**BookmarkName****property**

```
property BookmarkName : WideString
```

Specifies the name of the bookmark in the Document.

Bookmark names must begin with a letter, can contain numbers, but cannot include spaces.

---

# TOpDocumentHyperLinks Class

The TOpDocumentHyperLinks collection represents all of the hyperlinks of a TOpWordDocument instance. This class encapsulates and provides access to all of the hyperlinks defined in a document. The Add method is used to create a new TOpDocumentHyperLink object and add it to the collection. The Items property provides access to individual TOpDocumentHyperLink objects in the collection.

## Hierarchy

5

TCollection (VCL)

- ❶ TOpNestedCollection (OpOfficeShared) 434
  - TOpDocumentHyperLinks (OpWordClient)

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ❶ ItemName | ❶ ParentItem     | ❶ RootComponent |
| Items      | ❷ RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ❶ FindItem    |
| ❶ Create | ❶ ForEachItem |

## Reference Section

### Add method

---

```
function Add : TOpDocumentHyperLink;
```

Adds a new hyperlink to the parent document.

Calling Add creates a new hyperlink in the the parent document and adds it to Items. To delete a hyperlink from the document, call the Free method of the TOpDocumentHyperLink.

See also: TOpDocumentHyperLink

Items	property
-------	----------

```
property Items[index : integer] : TOpDocumentHyperLink
```

↳ Array property containing individual Shape (TOpDocumentHyperLink) items.

Use the Items property to access individual hyperlinks. Items is the default array property for TOpDocumentHyperLink. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpWord1.ActiveDocument.HyperLinks.Items[2].Free;  
OpWord1.ActiveDocument.HyperLinks[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add, TOpDocumentHyperLink

# TOpDocumentHyperLink Class

A TOpDocumentHyperLink item represents the colored range of text that specifies a jump to a location in a file, HTML page, newsgroup, FTP site, etc. The underlying HyperLink object interface provides numerous methods that can be used to manipulate a HyperLink.

## Hierarchy

TCollectionItem (VCL)

- ① TOpNestedCollectionItem (OpOfficeShared)438
- TOpDocumentHyperLink (OpWordClient)

## Properties

Address	① ParentItem	① SubCollectionCount
① Intf	① RootCollection	① Verb
NewWindow	① RootComponent	① VerbCount
① ParentCollection	① SubCollection	

## Methods

Activate	① Create	FollowHyperLink
① Connect	① ExecuteVerb	

## Reference Section

Activate	method
----------	--------

procedure Activate;

↳ Activates the HyperLink.

If the Word server is not connected or is not visible, calling this method will have no effect. Otherwise, the document to which this hyperlink belongs will receive focus and the hyperlink will be selected.

---

**Address** property

```
property Address : WideString
```

- ↳ Specifies the address (for example, a file name or URL) of the hyperlink.

---

**FollowHyperLink**method

```
procedure FollowHyperLink;
```

- ↳ Executes the link specified by Address.

FollowHyperLink resolves the hyperlink, downloads the target document, and displays the document in the appropriate application.

See also: Address

---

**NewWindow**property

```
property NewWindow : Boolean
```

- ↳ Specifies whether or not the link starts in the active window or a new browser window.

Available only with Word 2000.

---

# TOpDocumentShapes Class

The TOpDocumentShapes collection represents all of the shapes (e.g. drawings, pictures, etc.) of a TOpWordDocument instance. This class encapsulates and provides access to all of the shapes defined in a document. The Add method is used to create a new TOpDocumentShape object and add it to the collection. The Items property provides access to individual TOpDocumentShape objects in the collection.

## Hierarchy

TCollection (VCL)

- ① TOpNestedCollection (OpOfficeShared) 434
  - TOpDocumentShapes (OpWordClient)

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ① ItemName | ① ParentItem     | ① RootComponent |
| Items      | ① RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ① FindItem    |
| ① Create | ① ForEachItem |

## Reference Section

---

<b>Add</b>	<b>method</b>
------------	---------------

---

function Add : TOpDocumentShape;

↳ Adds a new Shape to the parent Document.

Calling Add creates a new shape in the the parent document and adds it to Items. To delete a shape from the document, call the Free method of the TOpDocumentShape.

See also: TOpDocumentShape

---

<b>Items</b>	<b>property</b>
--------------	-----------------

```
property Items[index : integer] : TOpDocumentShape
```

↳ Array property containing individual Shape (TOpDocumentShape) items.

Use the Items property to access individual shapes. Items is the default array property for TOpDocumentShape. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpWord1.ActiveDocument.Shapes.Items[2].Free;  
OpWord1.ActiveDocument.Shapes[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: Add, TOpDocumentShape

---

# TOpDocumentShape Class

A TOpDocumentShape item represents a shape object such as an AutoShape, freeform, OLE object, ActiveX control, or picture. Shape objects are anchored to a range of text but are free-floating so they can be positioned anywhere on the page.

## Hierarchy

TCollectionItem (VCL)

5

① TOpNestedCollectionItem (OpOfficeShared)438  
    TOpDocumentShape (OpWordClient)

## Properties

Height	① RootCollection	Top
① Intf	① RootComponent	① Verb
Left	① SubCollection	① VerbCount
① ParentCollection	① SubCollectionCount	Width
① ParentItem	ShapeName	

## Methods

Activate	① Create
① Connect	① ExecuteVerb

## Reference Section

### Activate method

---

procedure Activate;

↳ Activates the Shape.

If the Word server is not connected or is not visible, calling this method will have no effect. Otherwise, the document to which this shape belongs will receive focus and the shape will be selected.

## AddTextBox

method

```
procedure AddTextBox;
```

↳ Adds TextBox shape to the Document.

The TextBox shape is positioned at the location in the document window specified by the Left and Top properties. Its dimensions are specified by the Height and Width properties.

See also: Height, Left, Top, Width

### Height

property

```
property Height : Single
```

↳ Specifies the shape's vertical height in pixels.

The position of a shape is measured from the upper-left corner of the shape's bounding box to the shape's anchor.

See also: Left, Top, Width

### Left

property

```
property Left : Single
```

↳ Specifies the shape's horizontal position in pixels.

The position of a shape is measured from the upper-left corner of the shape's bounding box to the shape's anchor.

See also: Height, Top, Width

### ShapeName

property

```
property ShapeName : WideString
```

↳ The Name of the Shape in a Document.

Shape names must begin with a letter, can contain numbers, but cannot include spaces.

## Top

## property

property Top : Single

↳ Specifies the shape's vertical position in pixels.

The position of a shape is measured from the upper-left corner of the shape's bounding box to the shape's anchor.

See also: Height, Left, Width

## Width

## property

5

property Width : Single

↳ Specifies the shape's horizontal width in pixels.

The position of a shape is measured from the upper-left corner of the shape's bounding box to the shape's anchor.

See also: Height, Left, Top

---

# TOpMailMerge Class

The TOpMailMerge class provides access to a MailMerge object and identifies the mailmerge data source.

## Hierarchy

TPersistent (VCL)

TOpMerge (OpWordClient)

## Properties

AsMailMerge

OfficeModel

5

## Methods

SetDoc

## Reference Section

### AsMailMerge

**run-time, read-only property**

```
property AsMailMerge : MailMerge  
  MailMerge = interface;
```

↳ Contains the automation interface for the document MailMerge object.

Use AsMailMerge to access the extended functionality of the underlying MailMerge object.

The following example shows how to send the results of a mail merge to a new document:

```
with OpWord1.OpenDocument('c:\mailmerge.doc') do begin  
  MailMerge.OfficeModel := OpDatasetModel1;  
  PopulateMailMerge;  
  MailMerge.AsMailMerge.Destination :=  
    wdSendToNewDocument;  
  ExecuteMailMerge;  
end;
```

See also: TOpWordDocument.MailMerge

## OfficeModel

## property

```
property OfficeModel : TOpUnknownComponent
```

- ↳ Specifies the data provider model for mail merging.

This property can be assigned to a component that descends from TOpUnknownComponent and implements the IOpModel interface. Specify a TOpDataSetModel to mail merge with values obtained from a data base, or assign a TOpEvent model uses VCL events to provide the data through event handlers you define.

See also: TOpDataSetModel, TOpEventManager

5

## SetDoc

## method

```
procedure SetDoc(Doc : _Document);
```

- ↳ Allows a Document interface to be passed to the MailMerge interface in Word.

Doc contains an interface to the Word Document object to which the mail merge applies.

See also: AsMailMerge, OfficeModel

# Chapter 6: Working with PowerPoint

This chapter provides information on ways in which OfficePartner can give you control over Microsoft PowerPoint. The TOpPowerPoint component, and its various supporting classes, encapsulate and provide access to the functionality of the PowerPoint automation objects they represent.

## The Application object

The PowerPoint Application object is the top-level object in the PowerPoint object model. The Application object can determine or specify application-level properties or execute application-level methods. The Application object is also the entry point into the rest of the PowerPoint object model.

Like other Office application object models, the PowerPoint Application object exposes several properties which can work with a currently active PowerPoint object. The Application object exposes PowerPoint presentations, slides, shapes, etc., each of which exposes their own functionality. For example, the Slides property of a Presentation object returns the Slides collection that contains all the slides for the corresponding presentation.

In OfficePartner, the PowerPoint Application object is represented and controlled by the TOpPowerPoint component. Congruent to the PowerPoint Application Object, TOpPowerPoint is the primary component in the OfficePartner PowerPoint VCL class hierarchy.

## The Presentation object

In the PowerPoint object model, the Presentation object appears just below the Application object. The Presentation object represents an open presentation (\*.ppt) file. The Presentation object allows you to work with a single PowerPoint presentation. The properties and methods of the Presentation object allow you to control presentation-wide aspects and behavior. The Presentation object is represented and controlled by the TOpPresentation class. A collection of these is maintained in the TOpPresentations class. All Presentation objects are contained in the Application object's Presentations collection. The Presentations collection has a Count property you can use to determine how many presentations are open.

## The Slide object

A PowerPoint presentation consists of a series of slides which are displayed in sequence. The Slide object represents a particular slide in the presentation. The properties and methods of the Slide object allow you to control the format, content and behavior of an individual slide. The Slide object is represented and controlled by the TOpSlide class. A collection of these is maintained in the TOpSlides class. All Slide objects are contained in the Application object's

Slides collection. All Slide objects are contained in the Presentation object's Slides collection. The Slides collection has a Count property you can use to determine how many Slides are in the presentation.

### The Slide Transition object

In a PowerPoint slideshow, moving from slide to the next can be effected with a wide variety of animation effects. The new slide can appear as sliding into place from various angles, or it can materialize gradually as dots in the slide are filled in. There are dozens of animation effects. The TOpSlideTransition class encapsulates these animation effects.

### The Shape object

A PowerPoint slide consists of a collection of OLE objects such as text boxes, tables, charts, graph, pictures, etc., each of which is represented by a Shape object. Using properties and methods of the Shape object, you can control the shape's appearance, the text or OLE object it can contain, and the way it behaves during a slide show. The Shape object is represented and controlled by the TOpShape class. A collection of these is maintained in the TOpShapes class. All Shape objects are contained in the Slide object's Shapes collection. The Shapes collection has a Count property you can use to determine how many Shapes are on the slide.

---

# Microsoft PowerPoint Tutorial

The following sections will guide you through various steps in the process of opening a PowerPoint presentation, running a slide show, and manipulating various slide objects.

## Starting PowerPoint

Setting the `TOpPowerPoint.Connected` property to True starts the PowerPoint automation server. This can be done at run time or at design time from the Delphi or C++Builder IDE object inspector. Setting the `Visible` property to True will activate the window as demonstrated in the following example:

```
procedure TForm1.btnStartPowerPoint(Sender: TObject);
begin
  OpPowerPoint1.Connected := True;
  OpPowerPoint1.Visible := True;
end;
```

## Opening a new PowerPoint presentation

Opening a new PowerPoint presentation is simply a matter of creating a `Presentation` object. This can be done in two ways: at run time by calling the `TOpPowerPoint.Presentations` collection's `Add` method, or at design-time using the `Presentations` property editor from the Delphi or C++Builder IDE object inspector.

The following code illustrates how to open a new presentation at run time and add a slide:

```
procedure TForm1.bntNewPresentationClick(Sender:
  TObject);
var
  Pres : TOpPresentation;
  Slide : TOpSlide;
begin
  OpPowerPoint1.Connected := True;
  Pres := OpPowerPoint1.Presentations.Add;
  Slide := Pres.Slides.Add;
  OpPowerPoint1.Visible := True;
end;
```

## Opening an existing PowerPoint presentation

To open an existing PowerPoint presentation, first create a new `Presentation` object as outlined in the previous section. Then, specify the presentation file (\*.ppt) by setting the `PresentationFile` property in `TOpPresentation`.

The following code illustrates how to open and run an existing presentation:

```
procedure TForm1.btnRunSlideShowClick(Sender: TObject);
var
  Pres : TOpPresentation;
begin
  OpPowerPoint1.Connected := True;
  Pres := OpPowerPoint1.Presentations.Add;
  Pres.PresentationFile :=
    'd:\Office Partner\demos\PowerPoint\demo.ppt';
  OpPowerPoint1.Visible := True;
  Pres.RunSlideShow;
end;
```

## Saving and closing a PowerPoint presentation

You can save a presentation with TOpPresentation's Save or SaveAs methods. To close a presentation, simply free the Presentation object representing the presentation. When the Presentation object is destroyed, it is removed from the Presentations collection automatically.

The following code illustrates how to save and close an existing presentation at run time:

```
procedure TForm1.btnSaveAndCloseClick(Sender: TObject);
var
  Pres : TOpPresentation;
begin
  Pres := OpPowerPoint1.Presentations[0];
  Pres.Save;
  Pres.Free;
end;
```

## Adjusting slide transition effects

There is a plethora of animated slide transition effects available in PowerPoint. The TOpSlideTransition class allows you to set the transition effect for a particular presentation slide.

For example, the following code illustrates how to set entry into each slide to a random animation effect:

```
var
  i : Integer;
  Pres : TOpPresentation;
  Transition : TOpSlideTransition;
begin
  Pres := OpPowerPoint1.Presentations[0];
  for i := 0 to Pred(Pres.Slides.Count) do begin
    Transition := Pres.Slides[i].TransitionEffect;
    Transition.EntryEffect := eeEffectRandom;
  end;
end;
```

### Accessing PowerPoint automation interfaces

OfficePartner provides a component-based framework for encapsulating a presentation, however most of the functionality you are likely to use will require direct access to a particular PowerPoint automation object. The Presentation and Slide interfaces are easily accessed by AsPresentation and AsSlide properties.

For example, the following code illustrates how to access the presentation PageSetup object to set the presentation to start with slide number 8:

```
var
  Pres : TOpPresentation;
  IPres : _Presentation;
begin
  Pres := OpPowerPoint1.Presentations[0];
  IPres := Pres.AsPresentation;
  IPres.SlideShowSettings.StartingSlide := 8;
end;
```





# TOpPowerPoint Component

The TOpPowerPoint component represents an instance of the Microsoft PowerPoint Application. This component is used to control the PowerPoint Application and contains properties that control the application-wide aspects of PowerPoint. New and existing presentations can be opened using the NewPresentation and OpenPresentation methods. The ActivePresentation property provides access to the presentation that is currently active. The Presentations collection property provides access to all the open presentations.

6

## Hierarchy

TComponent (VCL)

TOpBaseComponent (OpShared)

① TOpOfficeComponent (OpShared) 443

TOpPowerPointBase (OpPpt2k)

TOpPowerPoint (OpPower)

## Properties

ActivePresentation	① OfficeVersionStr	ServerTop
Caption	Presentations	ServerWidth
① ClientState	① PropDirection	Version
① Connected	Server	Visible
① MachineName	ServerHeight	
① OfficeVersion	ServerLeft	

## Methods

① AddConnectListener	① GetFileInfo	Quit
① Create	① HandleEvent	① RemoveConnectListener
① CreateItem	NewPresentation	
① GetAppInfo	OpenPresentation	

## Events

OnOpConnect

OnOpDisconnect

OnGetInstance

## Reference Section

---

<b>ActivePresentation</b>	<b>run-time, read-only property</b>
---------------------------	-------------------------------------

---

```
property ActivePresentation : TOpPresentation
```

- ↳ Identifies the active presentation.

Read ActivePresentation to obtain the TOpPresentation representing the open presentation that has focus. If no presentation is open a nil pointer is returned.

See also: NewPresentation, OpenPresentation

---

<b>Caption</b>	<b>property</b>
----------------	-----------------

---

6

```
property Caption: string
```

- ↳ Specifies the text string displayed in the title bar of the application window.

To change the title bar caption of the application window to the default text, set Caption to an empty string.

---

<b>NewPresentation</b>	<b>method</b>
------------------------	---------------

---

```
function NewPresentation : TOpPresentation;
```

- ↳ Opens a new presentation.

Use NewPresentation to create a new presentation. The function creates a TOpPresentation instance that represents the presentation and adds it to the Presentations collection. The TOpPresentation instance is returned from the function call.

See also: Presentations, OpenPresentation

---

<b>OpenPresentation</b>	<b>method</b>
-------------------------	---------------

---

```
function OpenPresentation(
  const FileName : TFileName) : TOpPresentation;
```

- ↳ Opens the an existing presentation.

Use OpenPresentation to open an existing presentation. FileName is the name of the file on disk to open. The function creates a TOpPresentation instance that represents the presentation and adds it to the Presentations collection. The TOpPresentation instance is returned from the function call.

See also: Presentations, NewPresentation

## **Presentations**

**property**

```
property Presentations: TOpPresentations
```

- ↳ Contains a collection of all the PowerPoint presentations that are currently open.

Presentations is a TOpNestedCollection descendent which allows TOpPresentation(s) to be stored and restored with a form. Another nice feature is the ability to work with documents whether Connected is set to True or False. If the TOpPowerPoint component is connected to a running instance of PowerPoint, document changes to Presentations are immediately reflected in PowerPoint. If the component is not connected, changes may still be made to the Presentations property which will be applied when the component connects to PowerPoint.

See also: NewPresentation, OpenPresentation

## **Quit**

**method**

```
procedure Quit;
```

- ↳ Shuts down the PowerPoint application.

At run time, you have to call this procedure if you need to shut down the PowerPoint server because of the singleton nature of the PowerPoint server.

## **Server**

**read-only property**

```
property Server: _Application  
_Application = interface(IDispatch);
```

- ↳ Returns the Automation interface for the PowerPoint Server.

This property allows the developer direct access to the PowerPoint COM interface that the component is using. This allows the developer a way to access the myriad of features in Office which OfficePartner does not directly expose.

It is not feasible to document all methods and properties of this interface. You should consult the source code and any additional information (such as MSDN) prior to invoking methods of this interface.

## **ServerHeight**

**property**

```
property ServerHeight: Integer
```

- ↳ The Height (in pixels) of the PowerPoint MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerLeft, ServerTop, ServerWidth

---

**ServerLeft****property**

```
property ServerLeft: Integer
```

↳ The Left position (in pixels) of the PowerPoint MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerTop, ServerWidth

---

**ServerTop****property**

```
property ServerTop: Integer
```

↳ The Top position (in pixels) of the PowerPoint MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerWidth

---

**ServerWidth****property**

```
property ServerWidth: Integer
```

↳ The Width (in pixels) of the PowerPoint MDI window.

This value is converted from points to pixels, so small rounding errors may occur.

See also: ServerHeight, ServerLeft, ServerTop

---

**Visible****property**

```
property Visible: Boolean
```

↳ Specifies whether or not the PowerPoint application window is visible.

If Connected is True, this property specifies the visibility of the PowerPoint application. The PowerPoint server is different than any other Office server in that it has to be visible to send it automation calls. Setting Connected to True always sets Visible to True.

# TOpPresentations Class

The TOpPresentations collection represents all of the open presentations of a TOpPowerPoint instance. This class encapsulates and provides access to all of the open presentations. The Add method is used to open a new presentation by creating a new TOpPresentation object and adding it to the collection. The Items property provides access to individual TOpPresentation objects in the collection.

## Hierarchy

TCollection (VCL)

- ❶ TOpNestedCollection (OpShared) 434
  - TOpPresentations (OpPower)

6

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ❶ ItemName | ❶ ParentItem     | ❶ RootComponent |
| Items      | ❶ RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ❶ FindItem    |
| ❶ Create | ❶ ForEachItem |

## Reference Section

Add	method
-----	--------

function Add: TOpPresentation;

↳ Adds a new presentation to the collection of open presentations.

Calling Add opens a new presentation and adds it to Items. To close the presentation, call the Free method of the TOpPresentation object.

See also: TOpPresentation

---

Items	property
-------	----------

```
property Items[index: integer]: TOpPresentation
```

- ☞ Items provides access to all the documents the parent document has open.

Use the Items property to access an individual presentation. Items is the default array property for TOpPresentations. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpPowerPoint1.Presentations.Items[2].Free;  
OpPowerPoint1.Presentations[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: TOpPresentation

# TOpPresentation Class

The TOpPresentation class is a persistent class that represents a PowerPoint Presentation object. The TOpPresentation is used control an individual presentation and provides access to the slides of the presentation. The presentation can be saved by using the Save and SaveAs methods. Call the RunSlideShow method to start the slide show of a presentation. Closing the presentation is accomplished by simply destroying the TOpPresentation object.

Individual slides in the presentation are contained by the Slides collection property. The AsPresentation property provides access to the PowerPoint Presentation interface where the full functionality of PowerPoint is available.

Call the RunSlideShow method to start the slide show of the presentation.

## Hierarchy

TCollectionItem (VCL)

- ❶ TOpNestedCollectionItems (OpShared)
- TOpPresentation (OpPower)

## Properties

AsPresentation	PresentationFile	❶ SubCollection
❶ Intf	❶ RootCollection	❶ SubCollectionCount
ManualAdvance	❶ RootComponent	❶ Verb
❶ ParentCollection	Saved	❶ VerbCount
❶ ParentItem	Slides	

## Methods

❶ Activate	❶ ExecuteVerb	SaveAs
❶ Connect	RunSlideShow	
❶ Create	Save	

## Reference Section

### AsPresentation

### property

```
property AsPresentation: _Presentation
_Presentation = interface(IDispatch)
```

- ↳ Contains the automation interface for the presentation.

Much of the functionality of the Presentation object has been exposed through the interface of the TOPPresentation class. It may sometimes be necessary to expand on what OfficePartner has encapsulated and this is the interface you would use to do just that. This interface provides access to virtually all of the presentation functionality that Microsoft has elected to expose. It is not practical to document the entire functionality of this interface and it is suggested that you consult the source code and any additional information published by Microsoft (MSDN for example).

The following code shows how AsPresentation is used to print the presentation for example:

```
OpPowerPoint1.ActivePresentation.AsPresentation.Printout;
```

### ManualAdvance

### property

```
property ManualAdvance : Boolean
```

Default: False;

- ↳ Specifies whether or not the slide show advances in manual mode.

If ManualAdvance is False, the slide show advances according to the timing specified in the transition settings of each individual slide. Otherwise, each slide is advanced by the operator.

See also: TOpSlideTransition

### PresentationFile

### property

```
property PresentationFile: TFileName
```

- ↳ Specifies the presentation file name.

Changing PresentationFile will cause the specified presentation to be opened. The current open presentation is not saved.

## **RunSlideShow**

**method**

```
procedure RunSlideShow;
```

- ↳ Runs the SlideShow according to current slide show settings.

The following example illustrates how to start a PowerPoint slideshow:

```
OpPowerPoint1.Connected := True;  
with OpPowerPoint1.OpenPresentation(  
  'c:\OfficePartner\Examples\Demo.ppt') do  
  RunSlideShow;  
OpPowerPoint1.Quit;
```

## **Save**

**method**

```
procedure Save;
```

- ↳ Saves the presentation.

If the presentation has not been saved before, the Save As dialog box prompts the user for a file name. To determine whether or not the presentation has changed since the last time it was saved, use the Saved property.

See also: SaveAs, Saved

## **SaveAs**

**method**

```
procedure SaveAs(const FileName: string);
```

- ↳ Saves the presentation to a file specified by FileName.

FileName specifies the path and file name for the file. If a presentation with the specified FileName already exists, the presentation is overwritten without the user being prompted first.

See also: Save, Saved

## **Saved**

**property**

```
property Saved: Boolean
```

- ↳ Indicates whether or not the presentation has changed since it was last saved.

Setting Saved to False before closing the presentation will cause PowerPoint to display a prompt to save changes when the presentation is closed.

See also: Save, SaveAs

`property Slides: TOpSlides`

- « The collection that represents all the slides in the presentation.

This property provides access to all the Slide objects in the presentation. Use the Add method of TOpSlidesto create a new slide and add it to the collection.

To access a particular slide in the collection, use the Items property of TOpSlides .

See also: TOpSlides

---

# TOpSlides Class

The TOpSlides collection represents all of the slides of a TOpPresentation instance. This class encapsulates and provides access to all of the slides defined in a presentation. The Add method is used to create a new TOpSlide object and add it to the collection. The Items property provides access to individual TOpSlide objects in the collection.

## Hierarchy

TCollection (VCL)

- ① TOpNestedCollection (OpShared)434
  - TOpSlides (OpPower)

6

## Properties

- |            |                  |                 |
|------------|------------------|-----------------|
| ① ItemName | ① ParentItem     | ① RootComponent |
| Items      | ① RootCollection |                 |

## Methods

- |          |               |
|----------|---------------|
| Add      | ① FindItem    |
| ① Create | ① ForEachItem |

## Reference Section

---

### Add method

function Add: TOpSlide;

↳ Adds a new slide to the parent presentation.

Calling Add creates a new slide in the the parent presentation and adds it to Items. To delete a slide from the presentation, call the Free method of the TOpSlide instance.

See also: TOpSlide

---

### Items property

property Items[index: integer]: TOpSlide

↳ Array property containing individual slide (TOpSlide) items.

---

Use the Items property to access an individual presentation. Items is the default array property for TOpPresentations. As such, it is not necessary to specifically reference the Items property, although you may if you wish. The following two lines of code, for example, produce identical results:

```
OpPowerPoint1.ActivePresentation.Slides.Items[2].Free;  
OpPowerPoint1.ActivePresentation.Slides[2].Free;
```

Note that C++Builder does not support default array properties. For this reason, C++Builder users must reference the Items property.

See also: TOpSlide

# TOpSlide class

A TOpSlide item represents the primary unit for storing objects in a PowerPoint presentation. A slide contains one or more objects, such as title text, a bulleted list, drawing, picture, or chart. Each object on a slide can have associated formatting, such as font settings, color, and animation effects.

## Hierarchy

TCollectionItem (VCL)

- ① TOpNestedCollectionItem (OpShared)438
- TOpSlide (OpPower)

## Properties

AsSlide	① ParentCollection	① SubCollection
DisplayMasterShapes	① ParentItem	① SubCollectionCount
FollowMasterBackground	① RootCollection	TransitionEffect
① Intf	① RootComponent	① Verb
Layout	SlideName	① VerbCount

## Methods

Activate	① Create	① ExecuteVerb
① Connect	① Destroy	

## Reference Section

### Activate

### method

procedure Activate; override;

↳ Activates the slide.

If the PowerPoint server is not connected or is not visible, calling this method will have no effect. Otherwise, the presentation to which this slide belongs will receive focus and the slide will be selected.

```
property AsSlide : _Slide  
_Slide = interface(IDispatch)
```

- ↳ Contains the automation interface for the slide.

Much of the functionality of the Slide object has been exposed through the interface of the TOPSlide class. It may sometime be necessary to expand on what OfficePartner has encapsulated and this is the interface you would use to do just that. This interface provides access to virtually all of the presentation functionality that Microsoft has elected to expose. It is not practical to document the entire functionality of this interface and it is suggested that you consult the source code and any additional information published by Microsoft (MSDN for example).

**6**

The following code shows how AsSlide is used to obtain the slide ID for third slide in the active presentation for example:

```
var  
  Slide : TOPSlide;  
  ID : Integer;  
begin  
  Slide := OpPowerPoint1.ActivePresentation.Slides[2];  
  ID := Slide.AsSlide.SlideID;  
  ...
```

**DisplayMasterShapes****property**

```
property DisplayMasterShapes : Boolean
```

Default: True

- ↳ Specifies whether the background objects from the slide master are displayed.

These background objects can include text, drawings, OLE objects, and clip art you add to the slide master.

**FollowMasterBackground****property**

```
property FollowMasterBackground : Boolean
```

Default: True

- ↳ Specifies whether the slide follows the slide master background.

Set FollowMasterBackground to False if you want the slide to have a custom background.

## Layout property

---

```
property Layout: TOpPpSlideLayout

TOpPpSlideLayout = (ppslTitle, ppslText,
    ppslTwoColumnText, ppslTable, ppslTextAndChart,
    ppslChartAndText, ppslOrgchart, ppslChart,
    ppslTextAndClipart, ppslClipartAndText,
    ppslTitleOnly, ppslBlank, ppslTextAndObject,
    ppslObjectAndText, ppslLargeObject, ppslObject,
    ppslTextAndMediaClip, ppslMediaClipAndText,
    ppslObjectOverText, ppslTextOverObject,
    ppslTextAndTwoObjects, ppslTwoObjectsAndText,
    ppslTwoObjectsOverText, ppslFourObjects,
    ppslVerticalText, ppslClipArtAndVerticalText,
    ppslVerticalTitleAndText,
    ppslVerticalTitleAndTextOverChart);
```

Default: ppslTitle

↳ Specifies the Layout of the slide.

There are 24 predesigned slide layouts to choose from. For example, there's a layout that has placeholders for a title, text, and a chart; and there's another with placeholders for a title and clip art. The title and text placeholders follow the formatting of the slide master for your presentation.

When you apply a new layout, all text and objects remain on the slide, but you might need to rearrange them to fit the new layout.

## SlideName property

---

```
property SlideName: WideString
```

↳ Specifies the name assigned to the slide.

When a slide is added into a presentation, PowerPoint automatically assigns it a name in the form Sliden, where n is an integer that represents the order in which the slide was created in the presentation.

For example, the first slide added to a presentation is automatically named “Slide1”.

## TransitionEffect property

---

```
property TransitionEffect: TOpSlideTransition
```

↳ Returns the TOpSlideTransition instance associated with the slide.

---

Use `TransitionEffect` to control the animation effect when the slide becomes active in a slide show.

See also: `TOpSlideTransition`

# TOpSlideTransition Class

TOpSlideTransition is a persistent class that controls of the animation effects for a slide transition. When a slide becomes active in the context of a slide show, it can appear on the screen with a wide variety of different animations and animation speeds. The TOpSlideTransition class also controls how a slide transition is initiated.

## Hierarchy

TPersistent (VCL)

TOpSlideTransition (OpPower)

## Properties

AdvanceOnClick	AdvanceTime	Speed
AdvanceOnTime	EntryEffect	

## Methods

Create

## Reference Section

### AdvanceOnClick property

property AdvanceOnClick: Boolean

Default: False

↳ Specifies if the slide show can advance to the next slide upon a mouse click.

Set AdvanceOnClick to True to allow the slide show to advance to the next slide when the operator clicks the mouse. If you set both the AdvanceOnClick and the AdvanceOnTime properties to True, the slide will advance either when it's clicked or when the specified amount of time has elapsed.

See also: AdvanceOnTime

### AdvanceOnTime property

property AdvanceOnTime: Boolean

Default: False

Specifies if the slide show advances automatically to the next slide.

Set AdvanceOnTime to allow the slideshow to automatically advance to the next slide after a period of time specified by the AdvanceTime property. If you set both the AdvanceOnClick and the AdvanceOnTime properties to True, the slide will advance either when it's clicked or when the specified amount of time has elapsed whichever comes first.

**Note:** The slide transition will not start automatically if the ManualAdvance property of the TOpPresentation presentation is True.

See also: AdvanceOnTime, AdvanceTime, TOpPresentation.ManualAdvance

### AdvanceTime

### property

---

property AdvanceTime: Single

Default: 10

Specifies the amount of time, in seconds, after which the slide transition will occur.

If AdvanceOnClick is True, then set AdvanceTime to the number of seconds the slide show will wait before automatically advancing to the next slide. If AdvanceOnClick is False, then the value contained by AdvanceTime is not used.

**Note:** The slide transition will not start automatically if the ManualAdvance property of the TOpPresentation presentation is True.

See also: AdvanceOnTime, TOpPresentation.ManualAdvance

### Create

### constructor

---

constructor Create(SlideComp: TOpSlide);

Creates an instance of the TOpSlideTransition class.

SlideComp is the parent TOpSlide object to which this TOpSlideTransition object belongs.

## **EntryEffect**

**property**

```
property EntryEffect: TOpPpEntryEffect

TOpPpEntryEffect = (ppeeNone, ppeeCut,
    ppeeCutThroughBlack, ppeeRandom,
    ppeeBlindsHorizontal, ppeeBlindsVertical,
    ppeeCheckerboardAcross, ppeeCheckerboardDown,
    ppeeCoverLeft, ppeeCoverUp, ppeeCoverRight,
    ppeeCoverDown, ppeeCoverLeftUp, ppeeCoverRightUp,
    ppeeCoverLeftDown, ppeeCoverRightDown, ppeeDissolve,
    ppeeFade, ppeeUncoverLeft, ppeeUncoverUp,
    ppeeUncoverRight, ppeeUncoverDown,
    ppeeUncoverLeftUp, ppeeUncoverRightUp,
    ppeeUncoverLeftDown, ppeeUncoverRightDown,
    ppeeRandomBarsHorizontal, ppeeRandomBarsVertical,
    ppeeStripsUpLeft, ppeeStripsUpRight,
    ppeeStripsDownLeft, ppeeStripsDownRight,
    ppeeStripsLeftUp, ppeeStripsRightUp,
    ppeeStripsLeftDown, ppeeStripsRightDown,
    ppeeWipeLeft, ppeeWipeUp, ppeeWipeRight,
    ppeeWipeDown, ppeeBoxOut, ppeeBoxIn,
    ppeeSplitHorizontalOut, ppeeSplitHorizontalIn,
    ppeeSplitVerticalOut, ppeeSplitVerticalIn,
    ppeeAppear);
```

Default: ppeeNone

↳ Specifies the animation effect for the slide when activated in a slide show.

Set EntryEffect to achieve the desired animation effect. You'll probably want to experiment a bit to get the right look.

See also: Speed

## **Speed**

**property**

```
property Speed: TOpPpTransitionSpeed

TOpPpTransitionSpeed = (pptsMixed, pptsslow,
    pptsmedium, pptsfast);
```

Default: pptsmixed

↳ Specifies the speed of transition between slides.

Set Speed to adjust how quickly the animation specified by EntryEffect will progress. This is another one to experiment with.

See also: EntryEffect

# Chapter 7: Working with the Office Assistant

The TOpAssistant class encapsulates Microsoft Office's Help Assistant. OfficePartner allows you to customize the behavior of the Assistant as users of your application interact with one of the four supported Office applications (Excel, Word, Outlook, and PowerPoint). The interactive help system for each of these applications is essentially the same. Users interacting with any of the applications have at all times the ability to customize the help behavior global to all Office applications. When a user right-clicks the Assistant and selects Properties from the context menu, several boolean properties are displayed under the Options tab. OfficePartner encapsulates each of these properties and exposes them as simple Delphi properties. These properties are broken down into two categories. The first category determines how the Assistant responds to various events; the second category defines what sorts of tips the Assistant offers to users.

In addition to these properties, The TOpAssistant provides properties that further expand your control of the Assistant. In particular, you can control whether the Assistant is visible using the Visible property. You can also control the size of the Assistant using the Reduced property. Reduced is a boolean property that determines whether the Assistant appears in its default size (False) or reduced size (True).

## Using the balloon

The TOpAssistant provides a balloon property that can be used to present customized modal messages to your users. Merely setting the Visible property of the TOpAssistant to True will not display the agent. Instead, you need to set the Text property of the Assistant's balloon, and then invoke the balloon's Show method. The following code demonstrates how to do this:

```
Assistant.Balloon.Text := Edit1.Text;  
Assistant.Balloon.Show;
```

## Changing the appearance of the Assistant

You are by no means limited to the default Assistant when controlling an Office application via OfficePartner. Microsoft provides several Assistants, each of which is stored in its own agent character file (\*.acs) file. For the ambitious, new character files can be created in addition to the basic ones provided. The TOpAssistant class provides a FileName property that can be used to specify an Assistant other than the default. Simply set this property to the path and filename of a \*.acs file to display a different (or new) character.

## Linking the Assistant to an Office application

The Assistant must be ‘linked’ to one of the Office applications (Excel, Word, Outlook, or PowerPoint). The TOpAssistant class provides yet another property for this. The OfficeComponent property is of type TOpOfficeComponent, from which all of OfficePartner’s application components (TOpExcel, TOpWord, TOpOutlook, and TOpPowerPoint) are derived. Access to the Assistant requires two components be dropped on a form: The TOpAssistant class, and one of the four Office application components described above. Once both of these components are placed on a form, the Office application component can be selected in the OfficeComponent property of the TOpAssistant.





# TOpAssistant Class

Create an instance of the TTopAssistant class to return the Assistant object. There isn't a collection for the Assistant object; only one Assistant object can be active at a time. Use the Visible property to display the Assistant.

The default Assistant is Rocky. To select a different Assistant programmatically, use the FileName property.

## Hierarchy

TComponent (VCL)

TOpAssistant (OpMSO)

## Properties

7

AssistantLeft	FileName	Reduced
AssistantTop	GuessHelp	SearchWhenProgramming
AssistWithAlerts	HighPriorityTips	Sounds
AssistWithHelp	KeyboardShortcutTips	TipOfDay
AssistWithWizards	MouseTips	Visible
Balloon	MoveWhenInTheWay	
FeatureTips	OfficeComponent	

## Methods

Create	GetFileInfo
Destroy	GetOfficeConnected

## Reference Section

### AssistantLeft

### property

property AssistantLeft : SYSINT

↳ Determines the left location of the Office Assistant in pixels.

Use this property to set the left location (in screen pixels) of the Assistant.

See also: AssistantTop, Visible

## **AssistantTop**

**property**

---

**property AssistantTop : SYSINT**

- ↳ Determines the top location of the Office Assistant in pixels.

Use this property to set the top location (in screen pixels) of the Assistant.

See also: AssistantLeft, Visible

## **AssistWithAlerts**

**property**

---

**property AssistWithAlerts : WordBool**

Default: False

- ↳ Determines whether the Office Assistant balloon delivers application alerts when the Office Assistant is visible.

The AssistWithAlerts property corresponds to the Display alerts option under Use the Office Assistant on the Options tab in the Office Assistant dialog box.

If this property is set to False, the application displays alerts in dialog boxes.

See also: AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

7

## **AssistWithHelp**

**property**

---

**property AssistWithHelp : WordBool**

Default: False

- ↳ Determines how the Office Assistant appears when the user presses the F1 key.

If AssistWithHelp is set to False, the default Office help topic will be displayed when the user pressed the F1 key, otherwise, the Office Assistant appears when the user presses the F1 key.

The AssistWithHelp property corresponds to the “Respond to F1 key” option under Use the Office Assistant on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## AssistWithWizards

## property

```
property AssistWithWizards : WordBool
```

Default: False

- ↳ Determines whether or not the Office Assistant provides online Help with wizards.

The AssistWithWizards property corresponds to the Help with Wizards option under Use the Office Assistant on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## Balloon

## property

```
property Balloon : TOpBalloon
```

7

- ↳ Returns the Balloon object encapsulated within this instance of the Office Assistant.

Use the Text property of the balloon object to customize messages displayed to users.

See also: TOpBalloon reference

## Create

## method

```
constructor Create(AOwner : TComponent); override;
```

- ↳ Creates an instance of a TOpAssistant object.

An associated instance of TOpBalloon is also instantiated and available via the Balloon property.

See also: Destroy, Balloon

## Destroy

## method

```
destructor Destroy; override;
```

- ↳ Disconnects the Assistant from the Office application and frees the instance of the TOpAssistant object.

The associated TOpBalloon contained in the Balloon property is also freed.

See also: Create

```
property FeatureTips : WordBool
```

Default: False

- ↳ Determines whether or not the Office Assistant provides information about using application features more effectively.

Feature tips provide additional information regarding features you may be unaware of, or provide additional tips and tricks to help you expedite a process.

The FeatureTips property corresponds to the Using features more effectively check box on the Options tab in the Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

```
property FileName : TFileName
```

- ↳ Returns or sets the path and file name for the active Office Assistant.

In Windows, the file name extension for Assistant files is .acs. The files are installed in the \Program Files\Microsoft Office\Office folder in Windows 95, Windows 98, and Windows NT.

The following file names correspond to seven of the Assistants provided in Microsoft Office 2000. Additional Assistant choices may be available, depending on the language settings on an individual computer.

**Table 1:**

<b>Character</b>	<b>File name</b>
Clippit	Clippit.acs
Links	OffCat.acs
Rocky	Rocky.acs
Office Logo	Logo.acs
The Dot	Dot.acs
Mother Nature	MNature.acs
F1	F1.acs

7

See also: [GetFileInfo](#)

---

**GetFileInfo** method

---

```
procedure GetFileInfo(var Filter, DefExt: string);
```

↳ Gathers filter and default file extension from the OpOfficeConst.pas file.

Default values for these parameters are defined as follows:

```
SAssistantFilter = 'Assistant 97 *.act|*.act|Assistant 2000  
*.acs|*acs>All Files *.*|*.*';  
SAssistantDef = '*.acs';
```

See also: [FileName](#)

---

**GetOfficeConnected** read-only method

---

```
function GetOfficeConnected : Boolean;
```

↳ Indicates whether or not the Office component contained in the OfficeComponent property has been created and is connected.

If the OfficeComponent property = nil or OfficeComponent Connected property = False, then GetOfficeConnected will be False.

See also: [Visible](#)

## **GuessHelp**

**property**

**property GuessHelp : WordBool**

**Default:** False

- ↳ Determines whether or not the Office Assistant balloon presents a list of Help topics based on keywords the user selects before clicking the Assistant window or pressing F1.

The GuessHelp property corresponds to the Guess help topics option under Use the Office Assistant on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## **HighPriorityTips**

**property**

**property HighPriorityTips : WordBool**

**Default:** False

- ↳ Determines whether or not the Office Assistant displays high-priority tips.

The HighPriorityTips property corresponds to the Only show high priority tips option under Show tips about on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## **KeyboardShortcutTips**

**property**

**property KeyboardShortcutTips : WordBool**

**Default:** False

- ↳ Determines whether or not the Office Assistant displays Help about keyboard shortcuts.

The KeyboardShortcutTips property corresponds to the Keyboard shortcuts option in the Show tips about section on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## MouseTips

property

property MouseTips : WordBool

Default: False

- ↳ Determines whether or not the Office Assistant provides suggestions for using the mouse effectively.

The MouseTips property corresponds to the Using the mouse more effectively option under Show tips about on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## MoveWhenInTheWay

property

7

property MoveWhenInTheWay : WordBool

Default: False

- ↳ Determines whether or not the Office Assistant window automatically moves when it's in the way of the user's work area.

For example, the Assistant will move if it's in the way of dragging or dropping or in the way of keystroke entries.

The MoveWhenInTheWay property corresponds to the Move when in the way option in the Use the Office Assistant section on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, Reduced, SearchWhenProgramming, Sounds, TipOfDay

## OfficeComponent

property

property OfficeComponent: TOfficeComponent

- ↳ Contains the instance of the TOffice component with which the Office Assistant is associated.

Use the properties, methods, and events of the OfficeComponent to customize the behavior of the assistant.

See also: GetOfficeConnected

**Reduced****property****property Reduced : WordBool****Default:** False

↳ Determines whether or not the Office Assistant window appears in its smaller size.

This property is not used in Office 2000.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, SearchWhenProgramming, Sounds, TipOfDay

**SearchWhenProgramming****property****property SearchWhenProgramming : WordBool****Default:** False

↳ Determines whether or not the Office Assistant displays application and programming Help while the user is working in Visual Basic.

The default value is False. SearchWhenProgramming property corresponds to the option called Search for both product and programming help when programming. That option is in the Use the Office Assistant section on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, Sounds, TipOfDay

**Sounds****property****property Sounds : WordBool****Default:** False

↳ Determines whether or not the Office Assistant produces the sounds that correspond to animations.

The Sounds property corresponds to the Make sounds option under Use the Office Assistant on the Options tab in the Office Assistant dialog box. If a sound card is not installed, this property has no effect.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, TipOfDay

---

**TipOfDay****property**

```
property TipOfDay : WordBool
```

Default: False

- ↳ Determines whether or not the Office Assistant displays a special tip each time the Office application is opened.

The default value is False. TipOfDay property corresponds to the Show the Tip of the Day at startup option under Show tips about on the Options tab in the Office Assistant dialog box.

See also: AssistWithAlerts, AssistWithHelp, AssistWithWizards, FeatureTips, GuessHelp, HighPriorityTips, KeyboardShortcutTips, MouseTips, MoveWhenInTheWay, Reduced, SearchWhenProgramming, Sounds

---

**Visible****property**

```
property Visible : Boolean
```

Default: False

- ↳ Determines whether or not the Office Assistant is visible.

By default, the assistant is not visible. You must set this property to True in order to display the assistant;

See also: AssistantLeft, AssistantTop

---

# TOpBalloon Class

Objects of TTopBalloon work in conjunction with the application assistant to provide helpful feedback to the user as they work with Microsoft Office applications. TTopBalloon is a container for the text messages that the application assistant displays to users. The text is displayed in a ‘balloon’ adjacent to the assistant. New balloon messages are displayed by setting the text property of the TTopBalloon object, calling SetAppAssistant with a TTopAppAssistant as a parameter, and finally calling the Show method of the TTopBalloon class.

## Hierarchy

TPersistent (VCL)

TOpBalloon (OpMSO)

## Properties

Text

## Methods

Create	SetAppAssistant
GetOwner	Show

7

## Reference Section

Create	method
--------	--------

constructor Create(AOwner : TPersistent);

↳ Creates an instance of a TTopBalloon object.

See also: Show

**GetOwner****method**

```
function GetOwner : TPersistent; override;
```

- ↳ Returns the owner control of the TOpBalloon object.

This function is introduced in TPersistent (VCL) and overridden in descendant classes so that the descendant classes can appear in the object inspector.

See also: Show, SetAppAssistant

**SetAppAssistant****method**

```
procedure SetAppAssistant(COMAssistant : Assistant);
```

- ↳ Creates a new balloon for the AppAssistant.

The current property value of Text is assigned to the balloon caption.

See also: GetOwner

**Show****method**

```
procedure Show;
```

- ↳ Modally displays the balloon.

When this procedure is invoked, the property value Text appears in the balloon adjacent to the Office Assistant.

See also: Create, GetOwner, SetAppAssistant

**Text****property**

```
property Text : string
```

- ↳ Defines the text that will be displayed in the balloon.

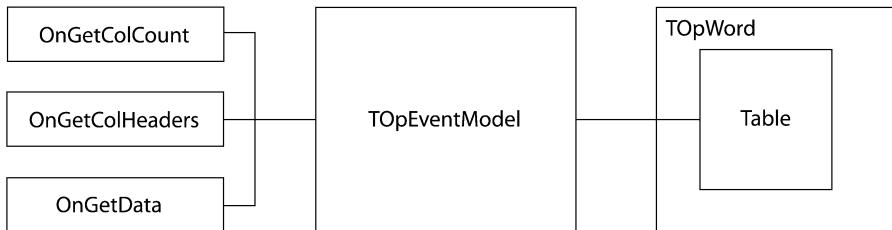
This property can be set at anytime, but the balloon will not be visible until TOpBalloon's Show method is called.

See also: Show, SetAppAssistant

# Chapter 8: The OfficePartner Event Model

The TOpEventModel component serves as a data provider for a Word Table or Excel range whereby data access is relegated to VCL event handlers. When this component is assigned to the OfficeModel property of a TDocumentTable object, a call to the Populate or Repopulate methods of the corresponding table will fire the OnGetColCount, OnGetColHeaders, and OnGetData events.

Within the event handlers you can obtain the data for a Word table in whatever fashion you care to implement. For example, you could obtain data values from a flat file, string list, excel worksheet, etc.



The following paragraphs illustrate how to create and populate a simple Word table with data from a string grid using the TOpWord and TOpEventModel components.

First, we'll drop a TOpEventModel on the form and assign event handlers to each of the three events supported by the TOpEventModel (namely, OnGetColCount, OnGetColHeaders, and OnGetData).

Within the event handler for the OnGetColCount event, the following code allows us to provide the desired column count:

```
procedure TForm1.OpEventModel1GetColCount(
  Sender : TObject; var ColCount : Integer);
begin
  ColCount := StringGrid1.ColCount;
end;
```

Within the event handler for the OnGetColHeaders event, we'll assign ColHeaders to a variant array. This must be done to avoid a run-time error when accessing the headers:

```
procedure TForm1.OpEventModel1GetColHeaders(
  Sender : TObject; var ColHeaders : Variant);
begin
  ColHeaders := Headers;
end;
```

Within the event handler for the OnGetData event, we'll transfer values from the string grid to the var parameters passed in:

```
procedure TForm1.OpEventModel1GetData(Sender : TObject;
  Index, Row : Integer; Mode: TRetrievalMode;
  var Size : Integer; var Data: Variant);
var
  OleStr : OleVariant;
begin
  OleStr := StringGrid1.Cells[Index, Row];
  Data := OleStr;
  Size := Length(OleStr);
end;
```

With the event handlers firmly in place, let's turn our attention to the code that launches Word, opens a new document, initializes a variant array for the column headers, and finally, creates and populates the table with data from the string grid. The following code accomplishes all of these things:

```
{ Public variables }
var
  Doc : TWordDocument;
  Table : TDocumentTable;
  Headers : Variant;
  ...
{ Launch word and open a new document }
OpWord1.Connected := True;
Doct := OpWord1.Documents.Add;
{ Initialize 1x3 variant array for column headers }
Headers := VarArrayCreate([0, 2], varVariant);
Headers[0] := 'Col 1';
Headers[1] := 'Col 2';
Headers[2] := 'Col 3';
{ Create table, hook up to TOpEventModel, and populate }
Table := Doc.Tables.Add;
Table.OfficeModel := OpEventModel1;
OpEventModel1.RowCount := StringGrid1.RowCount;
Table.PopulateDocTable;
OpWord1.Visible := True;
```

Now all that remains is a mechanism by which we can update the word table to reflect the latest data contained in the string grid. One line of code is all that is required to do this:

```
procedure TForm1.btnUpdateClick(Sender: TObject);
begin
  Table.RePopulateDocTable;
end;
```





# TOpEventModel Component

TOpEventModel component provides a framework of VCL events that are used for obtaining data with which to populate Excel ranges and Word tables. By defining VCL event handlers appropriately, data can be read, for example, from a VCL TStringGrid or TStringList.

## Hierarchy

TComponent (VCL)

TOpUnknownComponent (OpOfficeModels)

TOpEventModel (OpOfficeModels)

## Properties

RowCount

SupportedModes

VariableLengthRows

## Events

OnGetColCount

OnGetColHeaders

OnGetData

## Reference Section

### OnGetColCount

event

```
property OnGetColCount : TOpModelGetColCountEvent  
  TOpModelGetColCountEvent = procedure(  
    Sender : TObject; var ColCount : Integer);
```

Defines an event handler that is called when the column count is needed.

The Event Model will call this handler whenever it needs the column count of your implementation. Sender is the TOpEventModel that generated the event. Set the ColCount parameter to the appropriate column count value (e.g. ColCount := 5).

See also: RowCount, OnGetColHeaders, OnGetColCount

## OnGetColHeaders

event

```
property OnGetColHeaders : TOpModelGetColHeadersEvent  
TOpModelGetColHeadersEvent = procedure(  
  Sender : TObject; out ColHeaders : Variant);
```

Defines an event handler that is called when the column headers are needed.

Sender is the TOpEventModel that generated the event. Assign the column headers to the ColHeaders parameter within the event handler. as the following code illustrates:

```
ColHeaders :=  
  VarArrayOf(['Header1', 'Header2', 'Header3']);
```

The array should be 0 based and contain as many values as you have specified in ColCount.

See also: OnGetColCount, RowCount

## OnGetData

event

```
property OnGetData : TModelGetDataEvent  
TOpModelGetDataEvent = procedure(  
  Sender : TObject; Index : Integer; Row : Integer;  
  Mode : TOpRetrievalMode; out Size : Integer; out Data : Variant);  
TOpRetrievalMode = (rmCell, rmPacket, rmEntire);
```

Defines an event handler that is fired when the client requests data from the model.

Sender is the TOpEventModel that generated the event.

Index and Row correspond to column number and row number respectively. Both of these parameters are 0-based.

Mode specifies how the data is returned. A well written client should not ask for data in a retrieval mode that is not supported, however it may be wise to verify that the Mode parameter is supported. For rmPacket mode, the handler should return a Variant array representing an entire row of data. For rmCell mode, the handler should return a single value based on the Row and Index parameters that are passed. OfficePartner clients currently only support a packet size (Size parameter) of 1, which means a single row.

The size (in bytes) of the data returned should be assigned to the Size parameter, and the actual data should be assigned to the Data parameter within the event handler.

See also: SupportedModes, RowCount, OnGetColCount, OnGetColHeaders

**RowCount****property**

```
property RowCount : Integer
```

Default: 0

↳ Specifies the number of rows.

Set this property to specify how many rows exist in your Model. This will provide navigation functionality and proper implementation of EOF and BOF properties

See also: OnGetData, OnGetColCount, OnGetColHeaders

**SupportedModes****property**

```
property SupportedModes : TRetrievalModes
TOpRetrievalModes = set of TOpRetrievalMode;
TOpRetrievalMode = (rmCell,rmPacket,rmEntire);
```

Default: empty set

↳ Specifies which data retrieval modes your Model supports.

Currently Excel ranges can use rmPacket and rmCell and Word tables can use rmCell. Supplying a model that supports rmPacket will greatly increase performance when populating Excel ranges. If you specify that your Model supports a given mode, it is your responsibility to implement the data retrieval functions accordingly.

See also: OnGetData

**VariableLengthRows****property**

```
property VariableLengthRows : Boolean
```

Default: False

↳ Specifies whether the model can return a different column count for each row.

Currently OfficePartner does not use this setting as it assumes VariableLengthRows is True.

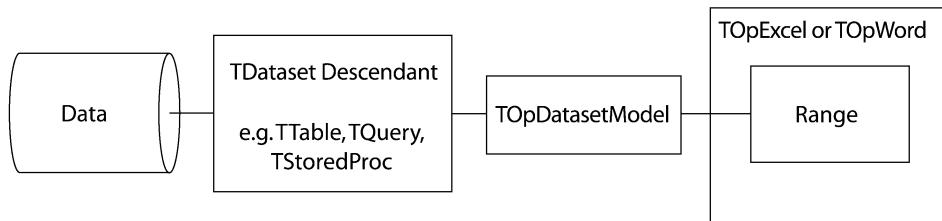
See also: OnGetColCount





## TOpDatasetModel Component

The TOpDataSetModel component serves as the link between one of the VCL's DataSet descendants (such as TTable or TQuery) and a Word or Excel range. This component allows for easy transmission of data from a database into a range in Excel or a table in Word. Use of this component allows you to call the Populate and PopulateCurrent methods of the corresponding range. Populate fills the associated range with all records and rows of the database via the TOpDataSetModel and PopulateCurrent fills the associated range with all columns in the current record of the database via the TOpDataSetModel.



To make this work, the `OfficeModel` property of the range must be set to a `TOpDataSetModel` component. This can be done via the nested collection editor or programmatically at run time. Then the `Dataset` property of the `TOpDataSetModel` must be set to any `TDataSet` descendant. Once both of these properties are established, simply call the `Populate` or `PopulateCurrent` method of the `Range` object.

The TOpDataSetModel component serves as the link between one of the VCL's DataSet descendants (such as TTable or TQuery) and a Word or Excel range. This component allows for easy transmission of data from a database into a range in Excel or a table in Word. Use of this component allows you to call the `Populate` and `PopulateCurrent` methods of the corresponding range. `Populate` fills the associated range with all records and rows of the database via the TOpDataSetModel and `PopulateCurrent` fills the associated range with all columns in the current record of the database via the TOpDataSetModel.

To use the TOpDataSetModel component, drop a TTable on the form, set its name to TblCustomers, and connect it to an existing table. Now, drop a TOpDataSetModel on the form and set its properties as follows:

**Table 1:**

<b>Property</b>	<b>Value</b>	<b>Description</b>
DataSet	TblCustomers	Connects the model to a TDataSet descendant (TTable or TQuery).
DetailModel	{leave blank}	Can be set to another TOpDataSetModel to establish a master-detail relationship (leave blank for now).
Name	MdlCustomers	Component name...
Tag	0	n/a in this example
WantFullMemos	False	Determines whether or not memo fields should be read in entirety from disk, or only the text "(Memo)" should be displayed in the associated range.

Now we'll set those same properties at run time. The following code demonstrates how to do this and also shows you how to invoke the `Populate` method of the range to populate the Excel range with all of the data contained in the dataset:

```

var
  Rng : TOpExcelRange;
begin
  {add a range to the first worksheet}
  Rng := OpExcel.Workbooks[0].Worksheets[0].Ranges.Add;

  {Give the range a name (simplifies access)}
  Rng.Name := 'CustomerRange';

  {We'll only set the anchor cell since we are
   populating the range through a dataset
   (unknown columns & rows)}
  Rng.Address := 'A1';

  {associate the DataSetModel with the range}
  Rng.OfficeModel := dmExcelDemo.mdlCustomers;

  {associate the DataSetModel with the DataSet}
  dmExcelDemo.mdlCustomers.Dataset :=
    dmExcelDemo.tblCustomers;

  {Fill the first worksheet (starting with the
   anchor cell) with every data column and row
   in the DataSet}
  OpExcel.RangeByName['CustomerRange'].Populate;
end;

```

In the previous example, the `populate` method filled the Excel worksheet (starting at the anchor cell A1) with all of the fields and records associated with `tblCustomers`. If we had called `PopulateCurrent` instead of `Populate`, only the values in the current record of `tblCustomers` would have been copied into the worksheet.

## Hierarchy

TComponent (VCL)

TOpUnknownComponent (OpOfficeModels)

TOpDataSetModel (OpDbOffice)

## Properties

DataSet

DetailModel

WantFullMemos

---

## Reference Section

Dataset	property
---------	----------

```
property Dataset : TDataSet
```

- ↳ Determines the dataset to which the TOpDataSetModel will connect.

Valid property values can be any accessible TDataSet descendant such as TTable, TQuery, or TStoredProc. The TOpDataset model then serves as the conduit between this property and a Word or Excel range.

See also: DetailModel

DetailModel	property
-------------	----------

```
property DetailModel : TOpDatasetModel
```

- ↳ Allows detail model rows to be interspersed with master model rows.

This property can be assigned to another TOpDatasetModel. When using with the TOpExcelRange OfficeModel property, this will result in a worksheet shown in Excel outline form where detail records are shown underneath each master record. The VCL dataset linked to the DetailModel property should be linked to the dataset from the master model in a master-detail fashion.

See also: DataSet

WantFullMemos	property
---------------	----------

```
property WantFullMemos : Boolean
```

Default: False

- ↳ Determines whether the text “Memo” appears in the output of the associated range or the full text of the memo appears.

If the WantFullMemos property is set to True, Excel will show the entire memo text in the populated Range. If WantFullMemos is set to False, then the string “Memo” will be shown in the populated cell.

See also: DataSet, DetailModel





## TOpContactsDataSet Component

The TOpContactsDataSet component is a VCL dataset that provides access to the Outlook contact manager. Basic dataset functionality, such as edits, appends, and deletes are supported by this component. More advanced functionality, such as inserts, sorting, and filters are not supported in this release.

This component essentially provides a familiar, data-oriented approach to accessing an Outlook contact list. The key to using this component successfully is in understanding Outlook contact lists.

This component provides three different methods for accessing an Outlook contact list. The ListSource property of the TdvContactsDataSet component allows you to specify one of the three following values:

- cltContacts
- cltCustom
- cltFolders

8

The following sections will describe the effect that each of these property settings has on traversing the various contact lists.

### ListSource = cltContacts

This setting is the value you'll most often use. This setting provides access to Outlook's default Contacts folder. This folder is guaranteed to reside on your user's computer (assuming, of course, that they have Outlook installed) and is guaranteed to have a default item type of "Contact". No additional properties of the TOpContactsDataset need to be set to access this list. Using this property setting, you can simply treat the TOpContactsDataset

as a (limited) VCL TDataSet descendant. The following code accesses the default contacts and populates a list box with first and last names (LB is a standard listbox, and CDS is a TOpContactsDataset):

```
procedure TForm1.PopulateBtnClick(Sender: TObject);
begin
  LB.Clear;      {clear the listbox}
  CDS.Active := True; {open the contacts dataset}
  CDS.First;   {navigate to the first record}
  while not CDS.eof do begin
    {add the first to fields to the listbox}
    LB.Items.Add(CDS.Fields[0].AsString + ' ' +
                  CDS.Fields[1].AsString);
    Cds.Next;
  end;
  CDS.Active := False; {close the contacts dataset}
end;
```

#### ListSource = cltCustom

This setting works in conjunction with the OnGetListSource event of the TOpContactsDataSet. The OnGetListSource event is generated when the ListSource property is cltCustom in order to obtain the contacts list with which the component will communicate. It is the responsibility of the developer to ensure the Items passed back contains ContactItem elements.

#### ListSource = cltFolders

This setting works in conjunction with the ListName property of the TOpContactsDataset component. The ListName property is used when ListSource is cltFolders in order to determine the list with which the component will communicate. If the first character of ListName is '/' or '\', then the component assumes that you are specifying a full folder path name for the contacts folder in question (e.g., "\Public Folders\My Contacts"). Otherwise, the component will traverse all of the folders within the MAPINameSpace, looking for the

first folder with a DefaultItemType of olContactItem whose name matches ListName. The following code accesses the list of contacts contained under \Personal Folders\New Contacts on the user's machine:

```
procedure TForm1.Button1Click(Sender : TObject);
begin
  cds.ListSource := cltFolders;
  cds.ListName := '\Personal Folders\New Contacts';
  cds.Active := True;
  cds.First;
  while not cds.eof do begin
    ListBox1.Items.Add(cds.Fields[0].AsString);
    cds.next;
  end;
  cds.Active := False;
end;
```

Note: It is the developer's responsibility to ensure that the named folder exists on the user's machine.

8

## Hierarchy

TComponent (VCL)

TDataSet (VCL)

TOpContactsDataSet (OpDbOutlook)

## Properties

Active	ListName	Outlook
ContactInfo	ListSource	

## Methods

GetFieldData

## Reference Section

### Active

### property

```
property Active : Boolean
```

Default: False

↳ Connects the TOpContactsDataSet to the Outlook Contact Manager.

Setting this property to True establishes a connection to the Outlook Contact Manager and setting this property to false breaks the existing connection to the Outlook Contact Manager.

The Outlook property must be set to a valid TOpOutlook component prior to setting this property.

See also: Outlook

### ContactInfo

### property

```
property ContactInfo : TContactInfoKinds  
TContactInfoKinds = set of TContactInfoKind;  
TContactInfoKind = (  
  citDefault, citBusiness, citPersonal, citNetwork, citMisc);
```

Default: citDefault

↳ Enables the user to determine what types of contact information are available in the dataset.

By limiting the items in this set, you can limit the number of automation calls sent to the server in order to read and post data.

See also: GetFieldData

### GetFieldData

### method

```
function GetFieldData(  
  Field : TField; Buffer : Pointer): Boolean;
```

↳ Fills the Buffer with the current value of the field specified by Field.

Use this method to retrieve the field value of the current record. GetFieldData returns True on success, False otherwise.

See also: Active, ContactInfo, ListSource, ListName

## **ListName** property

---

```
property ListName : string
```

- ↳ Determines the list with which the component will communicate.

The ListName property is used only when ListSource is cltFolders.

If the first character of ListName is '/' or '\', then the component assumes that you are specifying a full folder path name for the contacts folder in question (e.g., "\Public Folders\My Contacts"). Otherwise, the component will traverse all of the folders within the MAPINamespace, looking for the first folder with a DefaultItemType of olContactItem whose name matches ListName.

See also: ListSource

## **ListSource** property

---

```
property ListSource : TContactListSource
```

```
TContactListSource = (cltContacts, cltFolders, cltCustom);
```

8

Default: cltContacts

- ↳ Determines the type of address list to which the component will connect.

If the value of this property is cltFolders, then the ListName property is used to determine which list to access. If the value is cltCustom, then the OnGetListSource event is fired to obtain the list, and this event must be handled by the developer. See the documentation for the TContactListSource type for complete information on the possible values for this property.

See also: ListName

## **Outlook** property

---

```
property Outlook : TOpOutlook
```

- ↳ Contains the TOpOutlook server.

The Outlook property should be set to the TOpOutlook component through which you wish to manipulate the contact manager.

See also: Active

---

# Chapter 9: Low Level Classes

OfficePartner components rely on a set of base classes that provides the common core functionality of the components.

The TOpNestedCollection class manages a collection of TOpNestedCollectionItem.

The TOpNestedCollectionItem class represents an abstract automation object that is maintained within a collection.

The TOpOfficeComponent class is the base ancestor of all the OfficePartner components.

---

# TOpNestedCollection Class

TOpNestedCollection is a direct descendant of TCollection. TOpNestedCollection also adds functionality to assist in navigation throughout collection hierarchies.

Descendants of TOpNestedCollection objects contain descendants of TOpNestedCollectionItems. TOpNestedCollection introduces functionality that allows easy navigation throughout collection hierarchy.

Nested collection classes provide simple navigation throughout the entire hierarchy. Nested collections and their items are ultimately owned by a top-level OfficePartner component, and delegate most COM interface creation and retrieval to the top-level component. This is done so that the top-level component may correctly track the state of the automation server. Nested collections allow us to provide clean, consistent, and powerful design-time support for creating, maintaining and restoring the entities within an Office automation server.

## Hierarchy

TCollection (VCL)

TOpNestedCollection (OpOfficeShared)

9

## Properties

ItemName	RootCollection
ParentItem	RootComponent

## Methods

Create	FindItem	ForEachItem
--------	----------	-------------

## Reference Section

### Create

### method

```
constructor Create(RootComponent : TComponent;
  Owner : TPersistent; ItemClass : TCollectionItemClass); virtual;

TCollectionItemClass = class of TCollectionItem;

TCollectionItem = class(TPersistent);
```

- ↳ Creates a new persistent collection of items of type ItemClass.

The RootComponent parameter is usually a descendant of TOpOfficeComponent that owns the entire nested collection hierarchy. The Owner parameter is a descendant of TPersistent that owns the collection. The ItemClass parameter determines the class of the items created.

This constructor is generally called automatically via methods of descendant collections.

See also: ParentItem, RootCollection

### FindItem

### method

```
function FindItem(var Key; Proc : TNestedFindItemProc;
  FindList : TList) : Boolean;

TNestedFindItemProc = function (
  var Key; Item: TOpNestedCollectionItem): Boolean;
```

- ↳ Iterates the entire collection and calls the callback method passed in Proc.

Key is an untyped parameter used to find a match with a nested collection item.

The callback method is free to implement any algorithm to match the untyped Key parameter with a Nested Collection Item. The callback should return True if it finds a match and FindItem will automatically add the Item to the FindList parameter (which must be allocated by the caller). Once FindItem returns, FindList will be populated with all of the matched items.

See also: ForEachItem

## ForEachItem

method

```
procedure ForEachItem(Proc : TNestedForEachProc); virtual;  
TNestedForEachProc = procedure (  
    Item: TOpNestedCollectionItem);
```

↳ Iterates through the entire Nested Collection hierarchy.

The method passed in in the Proc parameter will be called for each item in the collection.

See also: FindItem

## ItemName

read-only property

```
property ItemName : string
```

↳ Readable class name for designer support.

Descendant classes provide their own internal implementation of this property.

See also: ParentItem

## ParentItem

read-only property

```
property ParentItem : TOpNestedCollectionItem
```

↳ Returns the TOpNestedCollectionItem that owns this nested collection.

If the owner is not a TOpNestedCollectionItem, this property will be nil.

See also: ItemName, TOpNestedCollectionItem reference

## RootCollection

read-only property

```
property RootCollection : TOpNestedCollection
```

↳ Returns the TOpNestedCollection at the top of the nested collection hierarchy.

If the current instance of TOpNestedCollection is at the top of the collection hierarchy, then this property will equal Self.

See also: RootComponent, ParentItem

**RootComponent****property**

`property RootComponent : TComponent`

>Returns the TComponent that owns the entire nested collection.

Root component is usually a descendant of TOpOfficeComponent.

See also: RootCollection, ParentItem

---

# TOpNestedCollectionItem Class

TOpNestedCollectionItem is a descendant of TCollectionItem. This class implements functionality which allows it to own other collections and be queried for the collections it owns. The TOpNestedCollectionItem class also implements functionality for easy navigation through a nested collection hierarchy. Currently, TOpNestedCollectionItem is also linked to our Office automation functionality.

TOpNestedCollection and TOpNestedCollectionItem provide the basis for OfficePartner components to correctly track and define the relationships between the interfaces of a specific Automation Server. TOpNestedCollectionItems can themselves contain multiple collections that in turn may contain other TOpNestedCollectionItems. This design gives us a persistable, hierarchical representation of the objects contained in the automation server.

## Hierarchy

TCollectionItem (VCL)

TOpNestedCollectionItem (OpOfficeShared)

## Properties

9

Intf	RootCollection	SubCollectionCount
ParentCollection	RootComponent	Verb
ParentItem	SubCollection	VerbCount

## Methods

Activate	Connect
ExecuteVerb	Create

## Reference Section

### Activate method

procedure Activate; virtual;

↳ Virtual placeholder for descendant classes.

Descendant collection item classes will override this method to provide custom activation.

See also: Connect

## Connect

method

```
procedure Connect;
```

- ↳ Stores an interface to the collection item in the Intf property.

The Connect method is called at construction time, and when the parent Office component is connected. Care should be taken when overriding Connect as it may be called before the class has fully constructed. Connect is used internally for fixing up automation server properties.

See also: Intf, Activate, Create

## Create

method

```
constructor Create(Collection : TCollection);
```

- ↳ Creates a collection item.

The newly created collection item is added to the collection specified by the Collection parameter.

See also: Activate, Connect

## ExecuteVerb

method

```
procedure ExecuteVerb(index : Integer);
```

- ↳ Used for designer support.

Descendant classes define their own implementation of this procedure.

See also: Verb, VerbCount

## Intf

read-only property

```
property Intf: IDispatch
```

- ↳ Returns an interface to the server with which the collection item is associated.

Intf will be unassigned if the RootComponent is not connected and, for some TOpNestedCollectionItems, may be unassigned even if the RootComponent is connected.

See also: Connect

## **ParentCollection**

**read-only property**

`property ParentCollection : TOpNestedCollection`

>Returns the TOpNestedCollection that owns this particular TOpNestedCollectionItem.

If the owner of this nested collection item is not a TOpNestedCollection, then the ParentCollection property will be nil.

See also: ParentItem, RootCollection

## **ParentItem**

**read only property**

`property ParentItem : TOpNestedCollectionItem`

Contains the TOpNestedCollectionItem that is the parent of this TOpNestedCollectionItem.

This property contains nil if the parent of this collection item is not a descendant of TOpNestedCollection.

See also: RootCollection, RootComponent

## **RootCollection**

**read-only property**

`property RootCollection : TOpNestedCollection`

>Returns the root component of the parent collection.

The root component of the parent collection is the TCollection at the top of the nested collection hierarchy.

See also: RootComponent

## **RootComponent**

**property**

`property RootComponent : TComponent`

>Returns the TComponent that owns the entire nested collection hierarchy.

The owner of the entire nested collection hierarchy is usually a descendant of TOpOfficeComponent.

See also: RootCollection, ParentCollection, ParentItem

---

**SubCollection** property

```
property SubCollection[index: Integer] : TCollection
```

- ↳ Allows indexed retrieval of nested collections.

The SubCollection array property can be used in conjunction with the SubCollectionCount property in order to dynamically retrieve nested collections.

See also: SubCollectionCount

---

**SubCollectionCount** read-only property

```
property SubCollectionCount : Integer
```

- ↳ Specifies the number of child collections that this item contains. `

Classes derived from TOpNestedCollectionItem provide their own mechanism for implementing this property.

See also: SubCollection

---

**Verb** read-only property

```
property Verb[index : integer]: string
```

- ↳ Used for designer support.

Descendant classes provide their own implementation of this property.

See also: VerbCount

---

**VerbCount** read-only property

```
property VerbCount : Integer
```

- ↳ Used for designer support.

Descendant classes provide their own implementation of this property.

See also: Verb

# TOpOfficeComponent Class

All of the Microsoft Office Applications define an object hierarchy for the COM interfaces that represent the entities within the application. For example, Excel has Application, Workbook, Worksheet, Range, Chart and Hyperlink entities. Word contains Application, Document, Bookmark, Table, Hyperlink, Shape, and Range entities. To represent these entities as separate components would leave the developer with the daunting task of defining and handling all of the complex relationships between these interfaces.

The top-level OfficePartner components all represent the main Application object (interface) within the automation server. The VCL TCollection and TCollectionItem classes offer one mechanism for holding the children (e.g. Workbooks, Documents, etc.) of the main Application interface. Collections would allow properties to be persisted and thus restored correctly, however this does not solve the problem of modeling how these different objects are related within the Office automation server. OfficePartner solves this problem by implementing specialized classes derived from VCL collections.

TOpOfficeComponent is the base class for the top-level OfficePartner components. Base class functionality is introduced for automation event support, correctly handling property fixups, and creation of automation server (CoClass) instances.

9

## Hierarchy

TComponent (VCL)

TOpOfficeComponent (OpOfficeShared)

## Properties

ClientState	MachineName	OfficeVersionStr
Connected	OfficeVersion	PropDirection

## Methods

AddConnectListener	CreateItem	HandleEvent
CoCreate	GetAppInfo	RemoveConnectListener
Create	GetFileInfo	

---

## Reference Section

---

### AddConnectListener method

```
procedure AddConnectListener(Listener: TOfficeConnectEvent);  
TOfficeConnectEvent = procedure (  
  Instance: TOpOfficeComponent; Connect: Boolean);
```

- ↳ Associates an event handler with a client connection.

The AddConnectListener is used to notify external objects (e.g. TOpAssistant) that the Office Component has connected.

See also: RemoveConnectListener

---

### ClientState property

```
property ClientState : TClientState  
TClientState = set of (csConnecting, csDisconnecting, csInEvent);
```

- ↳ Contains a set representative of the client state.

Members of this set together define the status of a client connection to the server. For example, this property allows the Office Component to tell if it is connecting, disconnecting, or receiving events.

See also: Connected, AddConnectListener, RemoveConnectListener

---

### CoCreate method

```
function CoCreate(const CoClass : TGUID; const Intf : IID;  
  Out Obj): RESULT;
```

- ↳ Creates COM objects.

The CoClass parameter specifies the class ID of the class to instantiate. The Intf parameter specifies the GUID of the interface requested. If creation is successful, the requested interface to the newly created object will be returned in the Out parameter. OfficePartner uses MultiQI to reduce round trips to the server when using DCOM.

See also: Create, CreateItem

---

**Connected** property

```
property Connected : Boolean;
```

Default: False

↳ Determines whether or not an instance of a derived Office application is running (e.g. Excel, Word, PowerPoint, or Outlook).

Connected will be False if no connection exists to an Office application, otherwise, Connected will be True.

See also: ClientState, AddConnectListener, RemoveConnectListener

---

**Create** method

```
constructor Create(AOwner : TComponent);
```

↳ Creates an instance of TOpOfficeComponent.

A list of ConnectionListeners is also created, and PropDirection is set to pdToServer.

See also: CoCreate, CreateItem

---

**CreateItem** method

```
function CreateItem(Item : TOpNestedCollectionItem): IDispatch;
```

↳ Creates and returns an interface to a nested collection item.

Collection children delegate initial automation connections to this method. This is done so that the component can properly synchronize and manage CoClass/Interface retrieval and maintenance. Developers should not call this method directly.

See also: CoCreate, Create

---

**GetAppInfo** method

```
procedure GetAppInfo(Info : TStrings);
```

↳ Returns system/application information from the Automation server in "Key=Value" pairs.

If the component is not connected, this method will temporarily launch the server.

See also: Connected, GetFileInfo

## **GetFileInfo**

## **method**

```
procedure GetFileInfo(var Filter, DefExt : string);
```

- ↳ Returns filename extensions supported by this component.

After calling this procedure, the Filter parameter will contain a string depicting the possible file filters that can be opened by a descendant of this TOpOfficeComponent, and the DefExt parameter will contain the default file extension used by a descendant of this TOpOfficeComponent.

See also: [GetAppInfo](#)

## **HandleEvent**

## **method**

```
procedure HandleEvent(const IID : TIID; DispId : Integer;  
const Params : TVariantArgList);
```

- ↳ Provides a central event dispatching mechanism.

HandleEvent is overridden in each TOpOfficeComponent subclass in order to correctly dispatch automation events to VCL event handlers. Various events are triggered based on the value of DispID.

This procedure is called internally and should not be called directly.

See also: [AddConnectListener](#), [RemoveConnectListener](#)

9

## **MachineName**

## **property**

```
property MachineName : WideString
```

- ↳ Sets or returns the name of a remote computer.

This property is used only for DCOM, An Office component can be launched on another computer provided DCOM is configured correctly.

See also: [PropDirection](#), [Connected](#)

## OfficeVersion

read-only property

```
property OfficeVersion: TOfficeVersion  
TOpOfficeVersion = (ovUnknown, ov97, ov98, ov2000);
```

>Returns the version of MS Office when connected.

OfficeVersion is an enumerated value that indicates the MS Office version to which the component is connected.

**Table 1:**

Value	Description
ovUnknown	Component is not connected to MS Office
ov97	Component is connected to MS Office 97
ov98	Component is connected to MS Office 98
ov2000	Component is connected to MS Office 2000

## OfficeVersionStr

read-only property

```
property OfficeVersionStr: string
```

9

>Returns the version number string of the connected Office product.

The OfficeVersionStr property contains the string (if available) representing the version number of the particular Office product (Excel, Outlook, PowerPoint, or Word) to which the component is connected.

If the component is not connected or the version string is not available, OfficeVersionStr contains an empty string.

## **PropDirection**

## **property**

```
property PropDirection : TPropDirection  
TPropDirection = (pdToServer, pdFromServer);  
Default: pdToServer
```

- ↳ Determines whether information flows from server to client or from client to server.

The PropDirection property controls what happens to the component properties when the component is initially connected. If PropDirection is pdToServer, the streamed design-time properties will be pushed to the automation server immediately after the server is launched.

If PropDirection is set to pdFromServer, the streamed properties will be ignored and the component properties will represent the state of the server when it is launched.

See also: Connected, MachineName

## **RemoveConnectListener**

## **method**

```
procedure RemoveConnectListener(Listener : TOfficeConnectEvent);  
TOfficeConnectEvent = procedure (  
    Instance: TOpOfficeComponent; Connect: Boolean)
```

- ↳ Disassociates an event handler from a client connection.

The RemoveConnectListener is used to unregister an external object (such as TOpAssistant) from receiving connection notifications.

See also: AddConnectListener, ClientState



# Identifier Index

\_Application 331, 382  
\_Document 338  
\_Presentation 387  
\_Slide 393

## A

Account  
    TOpBusinessData 294  
    TOpContactItem 201  
Activate  
    TOpDocumentBookmark 363  
    TOpDocumentHyperLink 367  
    TOpDocumentShape 371  
    TOpDocumentTable 358  
    TOpExcelChart 113  
    TOpExcelRange 94  
    TOpExcelWorkbook 81  
    TOpExcelWorksheet 88  
    TOpNestedCollectionItem 439  
    TOpSlide 393  
    TOpWordDocument 338  
Active 430  
ActiveDocument 322  
ActivePresentation 381  
ActualWork 273  
Add  
    TOpAttachmentList 171  
    TOpDocumentBookmarks 361  
    TOpDocumentHyperLinks 365  
    TOpDocumentShapes 369  
    TOpDocumentTables 356  
    TOpExcelCharts 111  
    TOpExcelHyperlinks 118  
    TOpExcelRanges 92  
    TOpExcelWorkbooks 79  
    TOpExcelWorksheets 86  
    TOpPresentations 385

Add (continued)  
    TOpRecipientList 160  
    TOpSlides 391  
    TOpWordDocuments 335  
AddConnectListener 444  
Address  
    TOpDocumentHyperLink 367  
    TOpExcelHyperlink 120  
    TOpExcelRange 94  
    TOpRecipient 164  
AddressEntry 164  
AddTextBox 371  
AdvanceOnClick 397  
AdvanceOnTime 397  
AdvanceTime 398  
AllDayEvent 179  
AnchorCell  
    TOpExcelHyperlink 120  
    TOpExcelRange 95  
Anniversary 201  
AppointmentItem 179  
AsChart 113  
AsChartObject 113  
AsDocument 338  
AsHyperlink 120  
AsMailMerge 374  
AsPresentation 387  
AsRange 95  
Assign 273  
AssistantLeft 404  
AssistantName  
    TOpBusinessData 294  
    TOpContactItem 201  
AssistantTelephoneNumber  
    TOpBusinessData 294  
    TOpContactItem 201  
AssistantTop 404  
AssistWithAlerts 404

AssistWithHelp 405  
AssistWithWizards 405  
AsSlide 393  
AsWorkbook 81  
AsWorksheet 88  
AttachmentIntf 175  
Attachments  
    TOpAppointmentItem 179  
    TOpJournalItem 232  
    TOpMailItem 245  
    TOpPostItem 260  
    TOpTaskItem 273  
AttachmentsIntf 171  
AutoFitColumns 95  
AutoFitRows 96  
AutoForwarded 245  
AutoHyphenation 338

## B

Balloon 405  
BeforeDocumentClose 322  
BeforeDocumentPrint 322  
BeforeDocumentSave 323  
BeforeSheetDoubleClick 63  
BeforeSheetRightClick 63  
BeforeWindowDoubleClick 323  
BeforeWindowRightClick 324  
BeforeWorkbookClose 64  
BeforeWorkbookPrint 64  
BeforeWorkbookSave 65  
BillingInformation  
    TOpAppointmentItem 180  
    TOpBusinessData 295  
    TOpContactItem 201  
    TOpJournalItem 232  
    TOpPostItem 260  
    TOpTaskItem 273  
Birthday 202  
Body  
    TOpAppointmentItem 180

TOpJournalItem 232  
TOpMailItem 245  
TOpNoteItem 253  
TOpPostItem 260  
TOpTaskItem 274  
BookmarkName 363  
Bookmarks 339  
BorderLineWeight 96  
Borders 96  
BorderStyle 97  
BrowseExtraFileTypes 324  
Business2TelephoneNumber  
    TOpBusinessData 294  
    TOpContactItem 202  
BusinessAddress 202  
BusinessAddressCity  
    TOpBusinessData 294  
    TOpContactItem 202  
BusinessAddressCountry  
    TOpBusinessData 295  
    TOpContactItem 202  
BusinessAddressPostalCode  
    TOpBusinessData 295  
    TOpContactItem 203  
BusinessAddressPostOfficeBox  
    TOpBusinessData 295  
    TOpContactItem 203  
BusinessAddressState  
    TOpBusinessData 295  
    TOpContactItem 203  
BusinessAddressStreet  
    TOpBusinessData 295  
    TOpContactItem 204  
BusinessFaxNumber  
    TOpBusinessData 295  
    TOpContactItem 204  
BusinessHomePage  
    TOpBusinessData 295  
    TOpContactItem 204  
BusinessTelephoneNumber  
    TOpBusinessData 294  
    TOpContactItem 204

- BusyStatus 180
- C**
- CallbackTelephoneNumber  
    TOpContactItem 204  
    TOpMiscData 302
- CancelResponseState 274
- Caption  
    TOpExcel 65  
    TOpPowerPoint 381  
    TOpWord 324
- CardData 274
- CarTelephoneNumber  
    TOpContactItem 204  
    TOpMiscData 302
- Categories  
    TOpAppointmentItem 181  
    TOpContactItem 205  
    TOpJournalItem 232  
    TOpMiscData 302  
    TOpNoteItem 253  
    TOpPostItem 260  
    TOpTaskItem 274
- Charts 88
- ChartType 114
- Children  
    TOpContactItem 205  
    TOpPersonalData 311
- Clear 98
- ClearConversationIndex  
    TOpPersonalData 245  
    TOpPostItem 260
- ClearOnMove 98
- ClearRange 98
- ClearRecurrencePattern  
    TOpAppointmentItem 181  
    TOpTaskItem 274
- ClientState 444
- Close  
    TOpAppointmentItem 181
- TOpContactItem 205  
    TOpJournalItem 233  
    TOpNoteItem 253  
    TOpPostItem 261  
    TOpTaskItem 275
- CoCreate 444
- CodeName 339
- Color  
    TOpExcelRange 98  
    TOpNoteItem 254
- ColumnWidth 99
- Companies  
    TOpAppointmentItem 182  
    TOpBusinessData 296  
    TOpContactItem 205  
    TOpJournalItem 233  
    TOpPostItem 261  
    TOpTaskItem 275
- CompanyAndFullName 206
- CompanyLastFirstNoSpace 206
- CompanyLastFirstSpaceOnly 206
- CompanyMainTelephoneNumber  
    TOpBusinessData 296  
    TOpContactItem 206
- CompanyName  
    TOpBusinessData 296  
    TOpContactItem 207
- Complete 275
- ComputerNetworkName  
    TOpContactItem 207  
    TOpNetworkData 306
- Connect  
    TOpExcelChart 114  
    TOpExcelHyperlink 121  
    TOpExcelRange 99  
    TOpExcelWorkbook 81  
    TOpExcelWorksheet 89  
    TOpNestedCollectionItem 439
- Connected  
    TOpExcel 65  
    TOpOfficeComponent 445  
    TOpOutlook 145

- ConsecutiveHyphensLimit 339  
ContactInfo 430  
ContactItem 207  
ContactNames  
    TOpJournalItem 233  
    TOpTaskItem 276  
Contacts 276  
ConversationIndex  
    TOpAppointmentItem 182  
    TOpJournalItem 233  
    TOpPostItem 261  
    TOpTaskItem 276  
ConversationTopic  
    TOpAppointmentItem 182  
    TOpJournalItem 234  
    TOpPostItem 261  
    TOpTaskItem 276  
Copy  
    TOpAppointmentItem 182  
    TOpContactItem 207  
    TOpJournalItem 234  
    TOpNoteItem 254  
    TOpPostItem 262  
    TOpTaskItem 276  
Count  
    TOpAttachmentList 172  
    TOpRecipientList 160  
Create  
    TOpAppointmentItem 183  
    TOpAssistant 406  
    TOpAttachment 175  
    TOpAttachmentList 172  
    TOpBalloon 413  
    TOpContactItem 208  
    TOpExcel 66  
    TOpExcelWorkbook 82  
    TOpJournalItem 234  
    TOpMailItem 245  
Create (continued)  
    TOpNestedCollection 435  
    TOpNestedCollectionItem 439  
    TOpNoteItem 254  
TOpOfficeComponent 445  
TOpPostItem 262  
TOpRecipient 164  
TOpRecipientList 161  
TOpSlideTransition 398  
TOpTaskItem 277  
CreateAppointmentItem 145  
CreateContactItem 146  
CreateInstance 147  
CreateItem  
    TOpExcel 66  
    TOpOfficeComponent 445  
CreateJournalItem 147  
CreateMailItem 148  
CreateNoteItem 149  
CreateOleObject 23  
CreatePostItem 150  
CreateTaskItem 151  
CreationTime  
    TOpAppointmentItem 183  
    TOpContactItem 208  
    TOpJournalItem 234  
    TOpNoteItem 254  
    TOpPostItem 262  
    TOpTaskItem 277  
CustomerID  
    TOpBusinessData 296  
    TOpContactItem 208

## D

- DataRange 115  
Dataset 425  
DateCompleted 277  
DefaultSaveFormat 325  
DefaultTableSeparator 325  
DefaultTabStop 339  
DeferredDeliveryTime 246  
DelegationState 278  
Delegator 278  
Delete

- TOpAppointmentItem 183
- TOpContactItem 208
- TOpJournalItem 235
- TOpMailItem 246
- TOpNoteItem 255
- TOpPostItem 262
- TOpTaskItem 278
- Department
  - TOpBusinessData 296
  - TOpContactItem 208
- Destroy
  - TOpAppointmentItem 184
  - TOpAssistant 406
  - TOpAttachment 176
  - TOpExcel 66
  - TOpExcelChart 115
  - TOpExcelHyperlink 121
  - TOpExcelWorkbook 82
  - TOpExcelWorksheet 89
  - TOpMailItem 246
  - TOpPostItem 263
  - TOpRecipient 164
  - TOpTaskItem 278
  - TOpWordDocument 339
- DetailModel 425
- Display
  - TOpAppointmentItem 184
  - TOpContactItem 209
  - TOpJournalItem 235
  - TOpMailItem 246
  - TOpNoteItem 255
  - TOpPostItem 263
  - TOpTaskItem 279
- DisplayAlerts 326
- DisplayMasterShapes 394
- DisplayName 176
- DisplayRecentFiles 326
- DisplayScreenTips 326
- DisplayScrollBars 326
- DisplayType 165
- DocFile 340
- DocPosted 235
- DocPrinted 235
- DocRouted 236
- DocSaved 236
- DocumentKind 340
- Documents 327
- DueDate 279
- Duration
  - TOpAppointmentItem 184
  - TOpJournalItem 236

## E

- Email1Address
  - TOpContactItem 209
  - TOpNetworkData 306
- Email1AddressType
  - TOpContactItem 209
  - TOpNetworkData 306
- Email1DisplayName 210
- Email1EntryID 210
- Email2Address
  - TOpContactItem 210
  - TOpNetworkData 306
- Email2AddressType
  - TOpContactItem 210
  - TOpNetworkData 306
- Email2DisplayName 210
- Email2EntryID 211
- Email3Address
  - TOpContactItem 211
  - TOpNetworkData 307
- Email3AddressType
  - TOpContactItem 211
  - TOpNetworkData 307
- Email3DisplayName 211
- Email3EntryID 211
- EmbedTrueTypeFonts 340
- EnableAnimations 66
- EnableAutoComplete 67
- EnableCancelKey
  - TOpExcel 67

- TOpWord 327
- E**
  - End\_
    - TOpAppointmentItem 184
    - TOpJournalItem 236
  - EntryEffect 399
  - EntryID
    - TOpAppointmentItem 185
    - TOpContactItem 212
    - TOpJournalItem 237
    - TOpNoteItem 255
    - TOpPostItem 263
    - TOpRecipient 165
    - TOpTaskItem 279
  - ExAppend 10
  - ExAppts 11
  - ExConn 11
  - ExecuteVerb
    - TOpExcelChart 115
    - TOpExcelHyperlink 121
    - TOpExcelRange 99
    - TOpExcelWorkbook 82
    - TOpNestedCollectionItem 439
  - ExEvent 11
  - ExFields 11
  - ExFind 11
  - ExMerge 11
  - ExO2Xl 11
  - ExpiryTime 263
  - ExPpt 11
  - ExSMail 12
  - ExTbl2 12
- F**
  - FeatureTips 406
  - FileAs 212
  - FileName
    - TOpAssistant 407
    - TOpAttachment 176
  - Filename 83
  - Find 341
- G**
  - FindItem 435
  - FindNext 341
  - FirstName
    - TOpContactItem 212
    - TOpDefaultData 299
  - Follow 121
  - FollowHyperLink 367
  - FollowMasterBackground 394
  - FontAttributes 100
  - FontColor 100
  - FontName 100
  - FontSize 101
  - ForEachItem 436
  - FormDescription 185
  - Forward
    - TOpJournalItem 237
    - TOpMailItem 247
    - TOpPostItem 263
  - ForwardAsVcal 185
  - ForwardAsVcard 212
  - FTPSite
    - TOpContactItem 212
    - TOpNetworkData 307
  - FullName 213
  - FullNameAndCompany 213

TOpAppointmentItem 186  
TOpTaskItem 279  
GovernmentIDNumber  
TOpBusinessData 296  
TOpContactItem 213  
GrammarChecked 341  
GuessHelp 408

## H

HandleEvent  
TOpExcel 68  
TOpOfficeComponent 446  
TOpOutlook 152  
HasRoutingSlip 342  
Height  
TOpDocumentShape 371  
TOpNoteItem 255  
HighPriorityTips 408  
Hobby  
TOpContactItem 213  
TOpPersonalData 311  
Home2PhoneNumber  
TOpContactItem 214  
TOpPersonalData 311  
HomeAddress 214  
HomeAddressCity  
TOpContactItem 214  
TOpPersonalData 311  
HomeAddressCountry  
TOpContactItem 214  
TOpPersonalData 311  
HomeAddressPostalCode  
TOpContactItem 214  
TOpPersonalData 311  
HomeAddressPostOfficeBox  
TOpContactItem 215  
TOpPersonalData 311  
HomeAddressState  
TOpContactItem 215  
TOpPersonalData 312

HomeAddressStreet  
TOpContactItem 215  
TOpPersonalData 312  
HomeFaxNumber  
TOpContactItem 215  
TOpPersonalData 312  
HomePhoneNumber  
TOpContactItem 216  
TOpPersonalData 312  
HorizontalAlignment 101  
HTMLBody  
TOpMailItem 247  
TOpPostItem 264  
Hyperlinks 89  
HyphenateCaps 342

## I

Importance  
TOpAppointmentItem 186  
TOpContactItem 216  
TOpJournalItem 237  
TOpMailItem 247  
TOpPostItem 264  
TOpTaskItem 279  
IndentLevel 102  
Index 166  
Initials 216  
Insert 343  
InsertStrings 343  
Interactive 68  
InternetFreeBusyAddress  
TOpContactItem 216  
TOpNetworkData 307  
Intf 440  
ISDNNumber  
TOpContactItem 216  
TOpNetworkData 307  
IsEmpty 102  
IsOnlineMeeting 187  
IsRecurring

TOpAppointmentItem 187  
 TOpTaskItem 280  
**I**  
 ItemInspector  
   TOpAppointmentItem 187  
   TOpContactItem 217  
   TOpJournalItem 238  
   TOpPostItem 264  
   TOpTaskItem 280  
 ItemName 436  
**I**tems  
   TOpAttachmentList 172  
   TOpDocumentBookmarks 361  
   TOpDocumentHyperLinks 365  
   TOpDocumentShapes 369  
   TOpDocumentTables 356  
   TOpExcelCharts 111  
   TOpExcelHyperlinks 118  
   TOpExcelRanges 92  
   TOpExcelWorkbooks 79  
   TOpExcelWorksheets 86  
   TOpPresentations 385  
   TOpRecipientList 161  
   TOpSlides 391  
   TOpWordDocuments 335

**J**

JobTitle  
   TOpBusinessData 296  
   TOpContactItem 217  
 Journal 217  
 JournalItem 238

**K**

KeyboardShortcutTips 409

**L**

Language

TOpContactItem 217  
 TOpMiscData 302  
 LargeButtons 68  
 LastFirstAndSuffix 217  
 LastFirstNoSpace 218  
 LastFirstNoSpaceCompany 218  
 LastFirstSpaceOnly 218  
 LastFirstSpaceOnlyCompany 219  
 LastModificationTime  
   TOpAppointmentItem 188  
   TOpContactItem 219  
   TOpJournalItem 238  
   TOpNoteItem 255  
   TOpPostItem 264  
   TOpTaskItem 280  
 LastName  
   TOpContactItem 219  
   TOpDefaultData 299  
 LastNameAndFirstName 219  
 Layout 394  
 Left  
   TOpDocumentShape 371  
   TOpExcelChart 116  
   TOpNoteItem 256  
 ListName 431  
 ListSource 431  
 Location 188  
 Login 152  
 Logoff 152  
  
**M**  
 MachineName  
   TOpExcel 69  
   TOpOfficeComponent 446  
   TOpOutlook 153  
 MailingAddress 220  
 MailingAddressCity  
   TOpContactItem 220  
   TOpDefaultData 299  
 MailingAddressCountry

- TOpContactItem 220
- TOpDefaultData 299
- MailingAddressPostalCode
  - TOpContactItem 221
  - TOpDefaultData 299
- MailingAddressPostOfficeBox
  - TOpContactItem 221
  - TOpDefaultData 299
- MailingAddressState
  - TOpContactItem 221
  - TOpDefaultData 299
- MailingAddressStreet
  - TOpContactItem 222
  - TOpDefaultData 300
- MailItem 247
- MailMerge
  - TOpMailMerge 374
  - TOpWordDocument 343
- ManagerName
  - TOpBusinessData 296
  - TOpContactItem 222
- ManualAdvance 387
- MapNamespace 153
- MarkComplete 280
- MeetingStatus 188
- MessageClass
  - TOpAppointmentItem 189
  - TOpJournalItem 238
  - TOpNoteItem 256
  - TOpPostItem 265
  - TOpTaskItem 281
- MiddleName
  - TOpContactItem 222
  - TOpDefaultData 300
- Mileage
  - TOpAppointmentItem 189
  - TOpJournalItem 239
  - TOpPostItem 265
  - TOpTaskItem 281
- MobileTelephoneNumber
  - TOpContactItem 222
  - TOpDefaultData 300
- MouseTips 409
- Move 222
- MoveWhenInTheWay 409
- MsgBCC 248
- MsgCC 248
- MsgTo 248

## N

- Name
  - TOpExcelRange 102
  - TOpExcelWorksheet 90
  - TOpRecipient 166
  - TOpWordDocument 344
- NetMeetingAlias
  - TOpContactItem 222
  - TOpNetworkData 308
- NetMeetingAutoStart 189
- NetMeetingOrganizerAlias 189
- NetMeetingServer
  - TOpAppointmentItem 189
  - TOpContactItem 223
  - TOpNetworkData 308
- NetMeetingType 190
- NewDocument 327
- NewPresentation 381
- NewSession 154
- NewWindow
  - TOpDocumentHyperLink 367
  - TOpExcelHyperlink 122
- NickName
  - TOpContactItem 223
  - TOpPersonalData 312
- NoAging
  - TOpAppointmentItem 190
  - TOpContactItem 223
  - TOpJournalItem 239
  - TOpPostItem 265
  - TOpTaskItem 281
- NoteItem 256

**O**

OfficeComponent 410  
OfficeLocation  
    TOpBusinessData 297  
    TOpContactItem 223  
OfficeModel  
    TOpDocumentTable 358  
    TOpExcelRange 103  
    TOpMailMerge 374  
OfficeVersion 447  
OfficeVersionStr 447  
OnDocumentChange 328  
OnDocumentOpen 328  
OnGetColCount 419  
OnGetColHeaders 419  
OnGetData 420  
OnItemSend 155  
OnNewDocument 328  
OnNewMail 155  
OnNewWorkbook 69  
OnOptionsPagesAdd 155  
OnQuit  
    TOpOutlook 156  
    TOpWord 328  
OnReminder 156  
OnSheetActivate 69  
OnSheetCalculate 69  
OnSheetChange 70  
OnSheetDeactivate 70  
OnSheetSelectionChange 71  
OnStartup  
    TOpOutlook 156  
    TOpWord 329  
OnWindowActivate  
    TOpExcel 71  
    TOpWord 329  
OnWindowDeactivate  
    TOpExcel 72  
    TOpWord 329  
OnWindowResize 72  
OnWindowSelectionChange 330

OnWorkbookActivate 72  
OnWorkbookAddinInstall 73  
OnWorkbookAddinUninstall 73  
OnWorkbookDeactivate 73  
OnWorkbookNewSheet 74  
OnWorkbookOpen 74  
OpenDocument 330  
OpenPresentation 381  
OptionalAttendees 190  
Ordinal 281  
OrganizationalIDNumber  
    TOpBusinessData 297  
    TOpContactItem 223  
Organizer 191  
Orientation 104  
OtherAddress 223  
OtherAddressCity  
    TOpContactItem 224  
    TOpMiscData 302  
OtherAddressCountry  
    TOpContactItem 224  
    TOpMiscData 302  
OtherAddressPostalCode  
    TOpContactItem 224  
    TOpMiscData 302  
OtherAddressPostOfficeBox  
    TOpContactItem 224  
    TOpMiscData 303  
OtherAddressState  
    TOpContactItem 225  
    TOpMiscData 303  
OtherAddressStreet  
    TOpContactItem 225  
    TOpMiscData 303  
OtherFaxNumber  
    TOpContactItem 225  
    TOpMiscData 303  
OtherPhoneNumber  
    TOpContactItem 225  
    TOpMiscData 303  
Outlook 431  
OutlookInternalVersion

- TOpAppointmentItem 191
  - TOpJournalItem 239
  - TOpPostItem 265
  - TOpTaskItem 281
  - OutlookVersion
    - TOpAppointmentItem 191
    - TOpJournalItem 239
    - TOpOutlook 157
    - TOpPostItem 265
    - TOpTaskItem 282
  - ovXxx 447
  - Owner 282
  - Ownership 282
- P**
- PageNumber
    - TOpContactItem 225
    - TOpDefaultData 300
  - ParentCollection 440
  - ParentItem
    - TOpNestedCollection 436
    - TOpNestedCollectionItem 440
  - PathName 176
  - Pattern 104
  - PatternColor 106
  - PercentComplete 282
  - PersonalHomePage
    - TOpContactItem 225
    - TOpNetworkData 308
  - Populate 106
  - PopulateCurrent 106
  - PopulateDocTable 358
  - PopulateMailMerge 344
  - Post 266
  - PostItem 266
  - PresentationFile 387
  - Presentations 382
  - PrimaryPhoneNumber
    - TOpContactItem 226
    - TOpDefaultData 300
- Print**
- TOpExcelWorkbook 83
  - TOpExcelWorksheet 90
  - TOpWord 344
- PrintFormsData** 344
- PrintOut**
- TOpAppointmentItem 191
  - TOpContactItem 226
  - TOpJournalItem 239
  - TOpMailItem 248
  - TOpNoteItem 256
  - TOpPostItem 266
  - TOpTaskItem 283
- PrintPostScriptOverText** 345
- PrintPreview** 330
- PrintRevisions** 345
- Profession**
- TOpBusinessData 297
  - TOpContactItem 226
- PropDirection**
- TOpExcel 74
  - TOpExcelWorkbook 83
  - TOpOfficeComponent 448
  - TOpOutlook 157
  - TOpWordDocument 345
- Q**
- QueryInterface 30
  - Quit
    - TOpOutlook 157
    - TOpPowerPoint 382
- R**
- RadioPhoneNumber
    - TOpContactItem 226
    - TOpNetworkData 308
  - RangeByName 75
  - RangeEnd 346
  - RangeLimit 346

- Ranges 90
  - RangeStart 346
  - ReadOnlyRecommended 347
  - ReceivedTime 266
  - RecipientIntf 166
  - Recipients
    - TOpAppointmentItem 191
    - TOpMailItem 248
    - TOpTaskItem 283
  - RecipientsIntf 161
  - RecipientType 167
  - RecurrenceState 192
  - Reduced 410
  - ReferredBy
    - TOpBusinessData 297
    - TOpContactItem 226
  - ReminderMinutesBeforeStart 192
  - ReminderOverrideDefault
    - TOpAppointmentItem 192
    - TOpTaskItem 283
  - ReminderPlaySound
    - TOpAppointmentItem 193
    - TOpTaskItem 283
  - ReminderSet
    - TOpAppointmentItem 193
    - TOpTaskItem 284
  - ReminderSoundFile
    - TOpAppointmentItem 193
    - TOpTaskItem 284
  - ReminderTime 284
  - Remove
    - TOpAttachmentList 173
    - TOpRecipientList 162
  - RemoveConnectListener 448
  - Replace 347
  - Reply
    - TOpJournalItem 240
    - TOpMailItem 249
    - TOpPostItem 267
  - ReplyAll
    - TOpJournalItem 240
    - TOpMailItem 249
  - ReplyTime 194
  - RepopulateDocTable 359
  - RequiredAttendees 194
  - Resolve 167
  - ResolveAll 162
  - Resolved 168
  - Resources 194
  - Respond
    - TOpAppointmentItem 194
    - TOpTaskItem 285
  - ResponseRequested 195
  - ResponseState 286
  - ResponseStatus 195
  - Role 286
  - RootCollection
    - TOpNestedCollection 436
    - TOpNestedCollectionItem 440
  - RootComponent
    - TOpNestedCollection 437
    - TOpNestedCollectionItem 441
  - RotateDegrees 107
  - RowCount 420
  - RowHeight 107
  - RunSlideShow 388
- S**
- Save
    - TOpAppointmentItem 195
    - TOpContactItem 226
    - TOpExcelWorkbook 84
    - TOpJournalItem 240
    - TOpMailItem 249
    - TOpNoteItem 257
    - TOpPostItem 267
    - TOpPresentation 388
    - TOpTaskItem 286
    - TOpWordDocument 348
  - SaveAs
    - TOpAppointmentItem 196
    - TOpContactItem 227

- TOpExcelWorkbook 84
- TOpJournalItem 240
- TOpNoteItem 257
- TOpPostItem 268
- TOpPresentation 388
- TOpTaskItem 287
- TOpWordDocument 348
- Saved
  - TOpAppointmentItem 196
  - TOpContactItem 227
  - TOpJournalItem 241
  - TOpMailItem 249
  - TOpNoteItem 258
  - TOpPostItem 268
  - TOpPresentation 388
  - TOpTaskItem 287
  - TOpWordDocument 348
- SaveFormsData 348
- SaveSubsetFonts 349
- SchedulePlusPriority 288
- ScreenUpdating 330
- SearchWhenProgramming 410
- Select 107
- SelectedMailingAddress 228
- Send
  - TOpAppointmentItem 197
  - TOpMailItem 249
  - TOpTaskItem 288
- SenderName
  - TOpMailItem 250
  - TOpPostItem 269
- SendMailMessage 158
- SendMailWithOutLook 349
- Sensitivity
  - TOpAppointmentItem 197
  - TOpJournalItem 241
  - TOpMailItem 250
  - TOpPostItem 269
  - TOpTaskItem 288
- Sent 250
- SentOn 269
- Server
  - TOpAppointmentItem 198
  - TOpContactItem 228
  - TOpJournalItem 242
  - TOpMailItem 250
  - TOpNoteItem 258
  - TOpPostItem 269
  - TOpTaskItem 288
- TOpExcel 75
- TOpOutlook 158
- TOpPowerPoint 382
- TOpWord 331
- ServerHeight
  - TOpExcel 75
  - TOpPowerPoint 382
  - TOpWord 331
- ServerLeft
  - TOpExcel 75
  - TOpPowerPoint 383
  - TOpWord 331
- ServerTop
  - TOpExcel 76
  - TOpPowerPoint 383
  - TOpWord 331
- ServerWidth
  - TOpExcel 76
  - TOpPowerPoint 383
  - TOpWord 332
- SetAddressFromSelection
  - TOpExcelChart 116
  - TOpExcelRange 108
- SetAppAssistant 413
- ShapeName 372
- Shapes 349
- Show 413
- ShowGrammaticalErrors 350
- ShowLoginDialog 158
- ShowRevisions 350
- ShowSpellingErrors 350
- ShowSummary 351
- ShrinkToFit 108
- SimpleText 108
- Size
  - TOpAppointmentItem 198
  - TOpContactItem 228
  - TOpJournalItem 242
  - TOpMailItem 250
  - TOpNoteItem 258
  - TOpPostItem 269
  - TOpTaskItem 288

- SkipRecurrence 288
- SlideName 395
- Slides 389
- Sounds 411
- Speed 400
- SpellingChecked 351
- Spouse
  - TOpContactItem 228
  - TOpPersonalData 312
- Start
  - TOpAppointmentItem 198
  - TOpJournalItem 242
- StartDate 289
- StartTimer 242
- StartupPath 332
- Status 289
- StatusOnCompletionRecipients 289
- StatusReport 290
- StatusUpdateRecipients 290
- StopTimer 242
- SubAddress 122
- SubCollection 441
- SubCollectionCount 441
- Subject
  - TOpAppointmentItem 198
  - TOpJournalItem 242
  - TOpMailItem 251
  - TOpNoteItem 258
  - TOpPostItem 269
  - TOpTaskItem 290
- Suffix
  - TOpContactItem 228
  - TOpDefaultData 300
- SummaryLength 351
- SummaryViewMode 352
- SupportedModes 421
  
- T**
- Tables 352
- TaskItem 290
- TBeforeSheetDoubleClick 63
- TBeforeSheetRightClick 63
- TBeforeWorkbookClose 64
- TBeforeWorkbookPrint 64
- TBeforeWorkbookSave 65
- TClientState 444
- TCollectionItem 435
- TCollectionItemClass 435
- TContactInfoKind 430
- TContactInfoKinds 430
- TContactListSource 431
- TeamTask 291
- TelexNumber
  - TOpContactItem 229
  - TOpNetworkData 308
- Text 414
- TipOfDay 411
- Title
  - TOpBusinessData 297
  - TOpContactItem 229
- TNestedFindItemProc 435
- TNestedForEachProc 436
- TOOfficeConnectEvent 444, 448
- TOnNewWorkBook 69
- TOnSheetCalculate 69
- TOnSheetChange 70
- TOnSheetDeactivate 70
- TOnSheetSelectionChange 71
- TOnWindowActivate 71
- TOnWindowDeactivate 72
- TOnWindowResize 72
- TOnWorkbookActivate 72
- TOnWorkbookAddinInstall 73
- TOnWorkbookAddinUninstall 73
- TOnWorkbookDeactivate 73
- TOnWorkbookNewSheet 74
- TOnWorkbookOpen 74
- Top
  - TOpDocumentShape 372
  - TOpExcelChart 116
  - TOpNoteItem 258
- TOpAssistant 403

TOpBalloon 412  
TOpContactsDataSet 426  
TOpDataSetModel 422  
TOpDocumentBookmark 362  
TOpDocumentBookmarks 360  
TOpDocumentHyperLink 366  
TOpDocumentHyperLinks 364  
TOpDocumentShape 370  
TOpDocumentShapes 368  
TOpDocumentTable 357  
TOpDocumentTables 355  
TOpEventModel 418  
TOpExcel 61  
TOpExcelBase 60  
TOpExcelHyperlink 119  
TOpExcelHyperlinks 117  
TOpExcelRange 93  
TOpExcelRanges 91  
TOpExcelWorkbook 80  
TOpExcelWorkbooks 78  
TOpExcelWorksheet 87  
TOpExcelWorksheets 85  
TOpMailMerge 373  
TOpModelGetColCountEvent 419  
TOpModelGetColHeadersEvent 419  
TOpModelGetDataEvent 420  
TOpNestedCollection 434  
TOpNestedCollectionItem 438  
TOpOfficeComponent 443  
TOpOfficeVersion 447  
TOpOnDocumentBeforePrint 322  
TOpOnDocumentBeforeSave 323  
TOpOnDocumentOpen 328  
TOpOnNewDocument 328  
TOpOnWindowActivate 329  
TOpOnWindowBeforeDoubleClick 323  
TOpOnWindowBeforeRightClick 324  
TOpOnWindowDeactivate 329  
TOpOnWindowSelectionChange 330  
TOpPowerPoint 380  
TOpPpEntryEffect 399  
TOpPpSlideLayout 394  
TOpPpTransitionSpeed 400  
TOpPresentation 386  
TOpPresentations 384  
TOpPropDirection 345  
TOpRetrievalMode 421  
TOpSlide 392  
TOpSlides 390  
TOpSlideTransition 396  
TOpUnknownComponent 103  
TOpWdAlertLevel 326  
TOpWdEnableCancelKey 327  
TOpWdRangeLimit 346  
TOpWdReplaceOption 347  
TOpWdSummaryMode 352  
TOpWdWindowState 333  
TOpWord 320  
TOpWordDocument 335, 336  
TOpWordDocuments 334  
TOpXIBorderLineStyles 97  
TOpXIBorders 96  
TOpXIBorderWeights 96  
TOpXICellHorizAlign 101  
TOpXIFontAttributes 100  
TOpXIInteriorPatterns 104  
TOpXIRangeBorders 96  
TOpXIRangeFontAttributes 100  
TotalWork 291  
TPropDirection 448  
TrackingStatus 169  
TrackingStatusTime 169  
TrackRevisions 353  
TransitionEffect 395  
TTYTDDTelephoneNumber  
    TOpContactItem 229  
    TOpNetworkData 309  
TwdViewType 354  
Type\_ 243

## U

UnRead

TOpAppointmentItem 198  
 TOpJournalItem 243  
 TOpMailItem 251  
 TOpPostItem 270  
 TOpTaskItem 291  
 UpdateStylesOnOpen 353  
**User1**  
 TOpContactItem 229  
 TOpMiscData 303  
**User2**  
 TOpContactItem 229  
 TOpMiscData 303  
**User3**  
 TOpContactItem 229  
 TOpMiscData 303  
**User4**  
 TOpContactItem 229  
 TOpMiscData 304  
**UserAddress** 332  
**UserCertificate**  
 TOpContactItem 229  
 TOpMiscData 304  
**UserControl** 353  
**UserInitials** 332  
**UserName**  
 TOpExcel 76  
 TOpWord 333  
**UserProperties** 230

**V**

Value 108  
**VariableLengthRows** 421  
**Verb** 441  
**VerbCount** 442  
**Version** 13  
**VerticalAlignment** 109  
**ViewType** 354  
**Visible**  
 TOpAssistant 411  
 TOpExcel 76

TOpExcelHyperlink 122  
 TOPowerPoint 383

**W**

WantFullMemos 425  
**WebPage**  
 TOpContactItem 230  
 TOpNetworkData 309  
**Width**  
 TOpDocumentShape 372  
 TOpExcelChart 116  
 TOpNoteItem 258  
**WindowState**  
 TOpExcel 77  
 TOpWord 333  
**Workbooks** 77  
**Worksheets** 84  
**WrapText** 109

**X**

xlblsxXX 97  
 xlchaxXxx 101  
 XlckXxx 67  
 XlcoXxx 104  
 XlctXxx 114  
 XLDemo 12  
 XlipXxx 105  
 XlRange 12  
 XlwsXxx 77  
 Xlxxx 49, 51, 52

**Y**

YomiCompanyName  
 TOpContactItem 230  
 TOpMiscData 304  
**YomiFirstName**  
 TOpContactItem 230

TOpMiscData 304  
YomiLastName

TOpContactItem 230  
TOpMiscData 304



---

# Subject Index

## A

### accessing

attachments 170, 172, 174, 179  
bookmark 361  
bookmarks 339, 360, 361  
cell 314  
charts 88, 111  
contacts 426  
contacts database 138  
default folders, Outlook 153  
document 320, 327, 335  
document contents 336  
hyperlink 364, 365  
hyperlinks 118, 342  
main interface 3  
Outlook folders 143  
presentations 380, 385  
ranges 75, 92  
recipients 159, 160, 161, 191, 248  
server interfaces 440  
shapes 349, 369  
slides 386, 389, 391  
table 355  
tables 352, 356, 357  
Word 313  
workbooks 77, 79  
worksheets 84, 86

### activating

Assistant 405  
bookmark 363  
collection items 439  
document 333, 338  
hyperlink 367  
ranges 94, 113  
server 1  
shape 371  
slides 393

### activating (continued)

tables 358  
Word 329  
workbooks 69, 81, 89, 99, 114, 121  
worksheets 88, 89, 94, 99, 113, 114, 121

### active

document 320  
presentations 380

### ActiveX control 27

### adding

attachments 170, 172, 174, 179  
bookmarks 314, 339, 360, 361  
charts 88, 110, 111, 112  
contacts 426  
document 327, 328, 330, 335  
hyperlinks 89, 117, 118, 314, 342, 364, 365  
presentations 377, 385  
ranges 90, 91, 92  
recipients 159, 160, 164, 166  
shapes 349, 369  
slides 389, 391, 395  
table 355  
tables 314, 318, 352  
text 313  
text box 371  
workbook 36, 56, 79, 82  
worksheets 56, 84, 86

### address

accessing lists 134  
determining list type 431  
displaying entries 165  
hyperlink 367  
iterating list 134  
range 108, 116

### advancing

presentations 387, 397  
slide shows 387, 397

### aging

- contact item 223
- post item 263, 265
- task item 281
- anchor cell, specifying 423
- animation
  - controlling effects 396
  - effects 392
  - enabling 66
  - slide effect 399
  - slides 376, 378, 395
  - sounds, Assistant 411
- application
  - information 67, 445
  - PowerPoint interface 382
  - Word interface 317
- Application object
  - Excel 33, 36, 75
  - Office 443
  - Outlook 124
  - PowerPoint 375
- appointment item
  - aging 190
  - all day appointment 179
  - closing 181
  - copying 182
  - creating 145
  - data fields 179–198
  - defining recurrence 186
  - deleting 145, 183
  - deleting recurrence 181, 278
  - displaying 184, 187
  - forwarding 185
  - indexing conversation thread 182
  - inspector 184, 187
  - interface 179
  - message class 189
  - multiple day appointment 179
  - printing 191
  - recurrence 192
  - reminder 192, 193, 196
  - saving 145, 181, 195, 196
- appointment item (continued)
- sending 197
- sensitivity 197
- size 198
- virtual calendar files 185
- assigning
  - column headers 419
  - entry ID 165
  - task item 273
- Assistant
  - activating 405
  - animation sounds 411
  - character files (\*.acs) 401, 407
  - connection notification 408
  - control of 401
  - creating 403, 406
  - customizing message of 405
  - destroying 406
  - determining host application 410
  - displaying 403
  - displaying alerts 404
  - displaying feature tips 406
  - displaying messages 401, 412
  - displaying programming help 410
  - displaying topics 408
  - filtering topics by priority 408
  - keyboard shortcuts 409
  - linking to Office applications 402
  - mouse tips 409
  - moving automatically 409
  - positioning 404
  - selecting 401, 403, 407
  - sizing 410
  - sounds 411
  - tip of the day 411
  - visible 411
  - wizard for 405
- attachments
  - accessing 170, 172, 174, 179
  - adding 170, 172, 174, 179
  - clearing in post item 263
  - collection of 245
  - counting 172

- creating 175
- creating item 183
- creating list 172, 183
- deleting 170, 172, 173, 174, 176, 179
- display name 176
- file name 176
- interface 171, 175
- journal item 232
- linked, path of 176
- mail item 245
- post item 260
- task item 273
- autoformatting 340
- autoforwarding mail items 245
- automatic hyphenation, enabling 338
- automation
  - abstract object 433
  - client 21, 27
  - definition 21
  - event support 443
  - server 21, 22, 27
  - server status 445
  - Word 313
- Automation server 438
  - application information 67
  - COM 1
  - connecting to 65
  - creating 66, 445
  - Excel 75
  - OfficePartner and 1, 443
  - Outlook 124
  - size of window 77
  - visibility 76
- B**
- background objects
  - displaying 394
  - slides 394
  - text 394
- balloon
  - creating 413
  - displaying 413
- displaying messages 412
- setting text 414
- showing messages 412
- base class 433
- binding
  - compile-time 25
  - early 23, 25, 27
  - late 23, 25, 28
  - run-time 25
- bookmark
  - accessing 339, 360, 361
  - activating 363
  - adding 314, 339, 360, 361
  - collectin 360
  - creating 360
  - deleting 361
  - indexing 361
  - naming 363
  - Word 313
- border
  - formatting 96
  - line weight 96
  - properties 96, 97
  - styles 97
- browser, launching 367
- business data
  - contact item 293
  - data fields 294–297
- C**
- calendar items, creating 131
- callback method 435
- calling convention 25
- Cancel key 67
- caption
  - displaying 65
  - setting in PowerPoint 381
- cell
  - accessing 314
  - address 94
  - anchor 94
  - border line weight 96

- border style 96, 97
- calculation 69
- clearing format 98
- color 98
- contents 39
- converting 314
- font color 41, 47, 100
- formatting 39, 314
- formatting alignment 48, 101, 102, 109
- formatting border line color 50
- formatting border line style 50
- formatting borders 49, 50, 51
- formatting characters 44
- formatting currency 43
- formatting dates 41
- formatting font 39, 46, 47, 100, 101
- formatting numbers 39, 45
- formatting pattern 104, 106
- formatting percentages 43
- formatting scientific notation 43
- formatting text 108, 109
- formatting text orientation 49, 104, 107
- formatting time 42
- populating 106
- referencing 94
- selecting 107
- sorting 314
- text shrink to fit 49
- text value 108
- text wrapping 49
- value 108
- changes, tracking 350
- character files (\*.acs) for Assistant 407
- character files, creating new 401
- chart
  - accessing 88, 111
  - adding 88, 110, 111, 112
  - category headers 115
  - collection 34, 110, 111
  - data address 115
  - data range 116
  - embedded 110, 111
- indexing 88, 111
- interface 113
- position 112, 115, 116
- PowerPoint 376
- size 112, 115, 116
- slides 392
- type 114
- chartsheet
  - accessing 35
  - contained 34
  - embedded 34
- OfficePartner support of 35
- class factory 28
- Class ID 23
- class name 436
- clearing cell format 98
- click events, document 323, 324
- client 444
  - adding connected listeners 444
  - connection status 444
  - determining state 444
  - disconnecting 448
  - state 444
- closing
  - appointment item 181
  - document 315, 316, 322, 335, 339
  - journal item 233
  - note item 253
- closing (continued)
  - post item 261
  - presentations 378, 386
  - contact item 205
  - task item 275
  - workbook 37, 64
- CLSID (class ID) 28
- CoClass 23
- collection
  - activating items 439
  - connecting to items 439
  - creating item 445
  - creating items 435, 439
  - determining parent of items 440

- determining parent of root 440
- document 313, 320, 327
- folders 143
- hyperlink 314
- indexing 441
- item owner 436
- items 440
  - iterating 435
  - navigating 438
  - navigating hierarchies 434
  - nested 2
  - of attachments 245
  - of bookmarks 360
  - of borders 96
  - of charts 34, 110, 111
  - of documents 334
  - of hyperlinks 35, 117
  - of ranges 90, 91, 92
  - of recipients 159, 160, 191, 245
  - of shapes 349
  - of tables 352, 355
  - of workbooks 34, 77
  - of worksheets 34, 84, 87
- owner 437
- presentations 375, 382
- querying 438
- removing document 316
- root 436
- scollection (continued)
  - erver associated with items 440
  - slides 375, 386, 389, 390
- collection items
  - maintaining 434
  - restoring 434
- column
  - autofit 53, 95
  - deleting 66
  - determining count 419
  - formatting 95
  - hiding 99
  - indexing 420
  - inserting 66
- referencing 94
- sizing 53, 95
- width 53, 95, 99
- column headers
  - assigning 419
  - determining 419
- COM 1, 28
  - Automation servers 1
  - creating object 444
  - interfaces 2
- committments, conflicting 180
- compile-time
  - binding 25
  - errors 24
- Component Object Model 1
- conflicting committments 180
- connecting
  - dataset 54
  - datasets 57
  - to collection items 439
  - to Contact Manager 430
  - to Excel 65, 81, 88, 89, 94, 99, 113, 114, 121
  - to Outlook 124, 130, 145, 152, 154, 157, 158
  - to PowerPoint 377, 383, 393
  - to server 65, 81, 88, 94, 113, 445
  - to Word 315, 338, 358, 363, 371
  - to workbook 81
- connection points 3, 28
- contact
  - accessing 426
  - accessing database 138
  - adding 426
  - deleting 426
  - editing 426
  - information 430
  - limiting information 430
  - ListName 431
  - obtaining list 427
- contact item
  - aging 223
  - business data 293
  - closing 205

- copying 207
- creating 146, 199, 208
- data fields 201–230
- default data 298
- deleting 146, 208
- displaying 209, 217
- filing 212
- forwarding 212
- inspector 199, 209, 217
- miscellaneous data 301
- moving 222
- network data 305
- personal data 310
- printing 226
- saving 146, 205, 212, 226, 227
- size 228
- user properties 230
- virtual card 212
- Contact Manager, connecting to 430
- contacts, specifying list source 426
- ContactsDataSet 430
- controlling
  - document 336
  - slides 375
- conversation thread
  - clearing 245, 260
  - indexing 182, 233, 276
  - journal item 234
- conversation thread (continued)
  - post item topic 261
  - task item 276
- conversation topic 182
- converting
  - cell 314
  - tables 357
  - text 325
- copying
  - appointment item 182
  - contact item 207
  - journal item 234
  - note item 254
  - post item 262
- task item 276
- creating
  - appointment item 145, 183
  - Assistant 403, 406
  - attachment item 183
  - attachment list 172
  - attachments 175
  - Automation server 443, 445
  - balloon 413
  - bookmarks 360
  - calendar items, Outlook 131
  - collection items 435, 439, 445
  - COM object 444
  - contact item 146, 199, 208
  - document 315, 320, 327, 328, 330, 335
  - hyperlink 364, 365
  - interfaces 434
  - journal item 147, 231, 234
  - mail item 125, 244, 245
  - new character files 401
  - note item 149, 252, 254
  - Outlook items 126
  - Outlook session 154
  - post item 150, 259, 262
  - presentations 377, 381
  - ranges 59
  - recipient list 161, 183
  - routing slip 342
- creating (continued)
  - server 66, 445
  - shapes 369
  - slide transition 398
  - slides 391
  - table 325, 355, 356, 415, 416
  - task item 151, 271, 277
  - workbook 36, 69, 79
  - workbooks 82
  - worksheet 87
- current record 430

## D

data

- address 115
- direction 74, 83, 157, 445, 448
- range 115, 116
- reading 418
- retrieval functions 421
- retrieval modes 421
- returning 420
- saving 348
- source, NameSpace 126
- transferring to Excel 416
- transferring to Word 416
- dataflow 2
- dataset
  - connecting 54
  - connecting to dataset model 422, 425
  - connecting to Excel 57
  - connecting to range 425
  - linking to Excel 422
  - linking to range 103
  - linking to Word 422
  - populating from 57
- dataset model
  - connecting to dataset 422, 425
  - specifying 358
- datatypes
  - automatable 25
  - OLE 25
- DCOM 28, 69, 153, 446
- deactivating Word 329
- default,
  - worksheet 87
- default data
  - contact item 298
  - data fields 299–300
- defining ranges 93
- deleting
  - appointment item 145, 183
  - appointment recurrence 181, 278
  - attachments 170, 172, 173, 174, 176, 179
  - bookmarks 361
  - contact item 146, 208
  - contacts 426
- e-mail 190
- hyperlink 365
- journal item 147, 235
- mail item 246, 249, 250
- note item 149, 255
- post item 150, 263
- recipients 162, 164, 166
- routing slip 342
- shapes 369
- slides 391
- table 356
- task item 151, 278
- task recurrence 274
- text 313
- destroying
  - Assistant 406
  - document 316
- determining parent of collection items 440
- dimension
  - note item 255, 258
- dimensions
  - Word server 331, 332
- direction 445, 448
- disassociating event handler 448
- disconnecting
  - client 448
  - from Outlook 152
- dispatching
  - events 446
  - Outlook events 152
- DISPID 25, 29
- dispinterface 22, 24, 26, 29
- displaying 246
  - address entries 165
  - appointment item 184, 187
  - Assistant 403
  - Assistant alerts 404
  - Assistant feature tips 406
  - Assistant messages 401, 412
  - Assistant tips 411
  - balloon 413
  - balloon messages 412

- buttons 68
- caption 65
- contact item 209, 217
- default folders 126
- docuemtnsummary 351
- document summary 351, 352
- file 326
  - grammatical errors 341
  - Item Inspector 145
  - journal item 235, 238
  - mail item 246
  - note item 255
  - Outlook Explorer 145
  - Outlook login dialog 158
  - post item 263, 264, 280
  - programming help 410
  - screen tips 326
  - scroll bar 326
  - slide shows 3
  - task item 279
- Distributed COM 69, 153, 446
- document
  - accessing 320, 327, 335
  - accessing contents 336
  - activating 333, 338
  - active 320, 322, 328
  - adding 327, 328, 330, 335
- document (continued)
  - adding table 356
  - adding text 313, 346
  - bookmark 360, 361
  - changing 328
  - check spelling 350
  - checking grammar 341
  - click events 323, 324
  - closing 315, 316, 322, 335, 339
  - codename 339
  - collection 313, 320, 327, 334
  - controlling 336
  - creating 315, 320, 327, 328, 330, 335
  - creating table 356
  - deleting table 356
- deleting text 313
- destroying 316
- displaying summary 351, 352
- e-mailing 349
- enabling automatic hyphenation 338
- executing mail merge 340
- fields 344
- file 330
  - file name 340, 344, 348
  - formatting 340
  - formatting text 313
  - freeing 339
  - hyperlink 364, 365
  - hyphenation 339, 342
  - identifying active 322
  - indexing 335
  - inserting text 343
  - interface 336, 338
  - mail merge 343, 373, 374
  - opening 315, 320, 327, 328, 330, 335, 340, 347, 353, 416
  - populating 3
  - printing 322, 344
  - printing options 344, 345
  - printing preview 330
  - range type 346
  - read only 347
- document (continued)
  - removing from collection 316
  - replacing text 347
  - revision tracking 353
  - saving 316, 323, 325, 348
  - screen painting 330
  - selection 330
  - shapes 349, 370
  - summary 351, 352
  - tab stop interval 339
  - table 355
  - tables 318, 352
  - text 315
  - tracking changes 318, 350
  - user 353

- view 354
- window state 333
- Word 313
- drawing slides 392
- dual interfaces 25
- E**
- early binding 23, 24
- editing
  - contacts 426
  - Outlook item properties 128
  - workbooks 68
  - worksheets 68
- effects
  - slide transitions 378
  - slides 376
- e-mail 132
  - archiving 190
  - deleting 190
  - documents 349
  - sending 3, 132, 158
- embedded charts 34
- enabling
  - animation 66
  - Cancel key 67
- entry ID, assigning 165
- error, compile-time 24
- event handler
  - disassociating 448
  - linking to client 444
- event hooks 3
- event model
  - assigning 358
- event support, automation 443
- events
  - dispatching 446
  - triggering 446
- example programs 10
- Excel
  - accessing charts 88, 111
  - accessing hyperlinks 118
  - accessing ranges 75, 92
  - accessing workbooks 77, 79
  - accessing worksheets 84, 86
  - activating workbooks 69
  - adding charts 88, 112
  - adding hyperlinks 118
  - adding hyperlinks collection 89
  - adding ranges 90, 91, 92
  - adding workbook 56, 79, 82
  - adding worksheets 56, 84, 86
  - Application object 33, 36, 75
  - Automation server, visibility of 76
  - Automation server 75
  - Automation server, window size of 77
  - cell address 94
  - cell calculation 69
  - cell color 98
  - cell contents 39
  - cell text shrink to fit 49
  - cell text wrapping 49
  - chart data address 115
  - chart interface 113
  - Chart object 35, 112
  - chart position 115, 116
  - chart size 115, 116
  - chart type 114
  - charts 110, 111
  - chartsheet 34
- Excel (continued)
  - closing workbooks 64
  - column width 53, 99
  - connecting datasets 57
  - connecting to 36, 65, 81, 88, 89, 94, 99, 113, 114, 121
  - creating 66
  - creating ranges 59
  - creating workbook 69, 79, 82
  - creating worksheets 87
  - disconnecting from 89
  - displaying buttons 68
  - editing workbooks 68
  - editing worksheets 68
  - following hyperlink 121

- font color 41, 47, 100  
formatting cell alignment 48, 101, 102, 109  
formatting cell border line color 50  
formatting cell border line style 50  
formatting cell borders 49, 50, 51  
formatting cell pattern 104, 106  
formatting cells 39  
formatting characters 44  
formatting columns 95  
formatting currency 43  
formatting dates 41  
formatting font 39, 46, 47, 100, 101  
formatting percentages 43  
formatting rows 96  
formatting scientific notation 43  
formatting text 108, 109  
formatting text orientation 49, 104, 107  
formatting time 42  
functions 53  
hyperlink address 120  
Hyperlink object 35  
hyperlink, target address of 122  
hyperlinks 117  
indexing chart 88  
indexing charts 111  
indexing hyperlinks 118  
indexing ranges 75, 92
- Excel (continued)
- indexing workbooks 77, 79
  - indexing worksheets 84, 86
  - linking datasets 422
  - naming ranges 59, 75, 102
  - naming worksheet 90
  - number format 39, 45
  - object model 33
  - opening workbooks 83
  - populating range 106
  - populating ranges 57, 59, 95, 99, 106
  - printing workbooks 64
  - printing worksheet 90
  - range address 108, 116
  - range interface 95
- Range object 34, 35  
range value 108  
ranges, defining 93  
ranges, naming 93  
row height 53  
saving workbooks 65, 68, 79, 82, 84  
saving worksheets 68, 86  
selecting cells 107  
selecting range 99, 107  
server dimensions 75, 76  
server position 75, 76  
server size 75, 76  
server, window size of 77  
server, visibility of 76  
servers 75  
text shrink to fit 108  
text value 108  
transferring data to 416  
visibility 81, 88, 89, 94, 99, 113, 114, 121  
visible 36  
workbooks 34, 77  
worksheets 34, 87  
wrap text to fit 109
- Explorer, Outlook 127  
extension 68  
extensions, file 446
- F**
- field, specifying 430  
file 68
  - displaying 326
  - document 330
  - document name 344
  - extensions 83, 446
  - filters 68
  - names 83
  - presentation name 387
  - Word 325file filters 446  
file name, presentations 388  
files
  - Assistant character (\*.acs) 401

- browsing with Word 324
  - form 10
  - package 10
  - source 7
  - filing contact item 212
  - filling worksheets 3
  - filtering Assistant help topics 408
  - filters, file 446
  - finding, text 316, 336, 341
  - folder
    - default item type 431
    - specifying path 427
  - folder collection 143
  - folders 431
  - following hyperlink 121, 122, 366, 367
  - font settings, slides 392
  - fonts
    - embedded 349
    - embedding 340
    - saving 349
    - True Type 340
    - TrueType 349
  - form files 10
  - formatting
    - borders 96
    - cell 314
    - cell alignment 48, 101, 102, 109
    - cell border line color 50
    - cell border line style 50
    - cell borders 49, 50, 51
    - cell characters 44
    - cell currency 43
    - cell dates 41
    - cell font color 41, 47, 100
    - cell fonts 39, 46, 47, 100, 101
    - cell pattern 104, 106
    - cell percentages 43
    - cell scientific notation 43
    - cell text orientation 49, 104, 107
    - cell time 42
    - cells 39
    - clearing 98
  - columns 53
  - document 340
  - rows 53
  - slides 392, 394
  - tables 318, 357
  - text 313
  - forwarding
    - contact item 212
    - journal item 237
    - mail item 247
    - post item 263
  - freeing, document 339
  - functions, defining 53
- G**
- GetIDsOfNames 23, 25
  - getting default folders, Outlook 128, 130
  - getting field data 344
  - glossary 27
  - grammatical errors
    - marking 350
  - graphs, PowerPoint 376
  - GUID 23, 29
- H**
- hardware requirements 5
  - help, on-line 18
  - hierarchy 13
  - hyperlink
    - accessing 35, 118, 342, 364, 365
    - activating 367
    - adding 89, 117, 118, 314, 342, 364, 365
    - address 120, 367
    - collection 89, 117, 314
    - collection of 35
    - creating 364, 365
    - deleting 365
    - destination address 120
    - following 121, 122, 366, 367
    - indexing 118, 365
    - interface 120
    - launching browser 367

- target address 122
- visibility 122
- Word 313
- Hyperlink object 35
- hyphenation
  - behavior 342
- hypnentation
  - width 342
- |
- ID, slides 393
- identifying document 322
- IDispatch 22, 25, 29
- IDL (Interface Declaration Language) 24
- implementing data retrieval functions 421
- importing type library 24
- indexing
  - attachments 179
  - bookmark 361
  - charts 88, 111
  - collection 441
  - column 420
  - conversation thread 182, 233, 261, 276
- indexing (continued)
  - document 335
  - hyperlinks 118, 365
  - nested collection 441
  - Outlook folders 143
  - Outlook items 143
  - presentations 385
  - ranges 75, 92
  - recipients 161, 166
  - row 420
  - shapes 369
  - slides 391
  - tables 356
  - workbooks 77, 79
  - worksheets 84, 86
- information
  - application 445
  - system 445
- initializing variant array 416
- in-process servers 21, 29
- inserting text 336, 343
- inspector
  - journal item 235, 238
  - mail item 246, 255
  - task item 279
- installing OfficePartner 5
- interface
  - appointment item 179
  - attachment list 171
  - attachments 175
  - chart 113
  - defined 30
  - document 336, 338
  - hyperlink 120
  - IDispatch 22, 23
  - journal item 238
  - mail item 245, 247
  - mailmerge 374
  - MAPINNameSpace 131, 153
  - note item 252, 256
  - Outlook application 147, 158
  - Outlook folders 153
- interface (continued)
  - Outlook form 185
  - post item 266
  - range 95
  - recipient 163, 166
  - recipient list 161
  - server 331
  - slides 393
  - task item 290
  - workbook 81
  - worksheet 88
- Interface Declaration Language (IDL) 24
- interfaces 2, 434, 443
  - \_Application/Application 3
  - creating 434
  - delegating 434
  - dual 25
  - IDispatch 25
  - related server 438

- typecasting 3
  - internal version, Outlook 191, 265, 281
  - interruption macro 327
  - invalidating post item 263
  - invoke 23
  - item inspector 187
  - Item Inspector, Outlook 127
  - item type, default 426, 431
  - iterating
    - collections 435
    - nested collections 436
    - recipients 191
  - IUnknown 30
- J**
- journal item
    - attachments 232
    - closing 233
    - conversation thread 234
    - copying 234
    - creating 147, 231, 234
    - data fields 232–243
    - deleting 147, 235
  - journal item (continued)
    - displaying 235, 238
    - forwarding 237
    - indexing conversation thread 233
    - inspector 235, 238
    - interface 238
    - printing 235, 239
    - replying 240
    - saving 147, 233, 240
    - sensitivity 241
    - size 242
- K**
- key, registry 23
  - keyboard shortcuts, Assistant 409
  - knowledgebase 19
- L**
- late binding 23
- launching browser 367
  - layout, slides 394
  - list
    - obtaining contacts 427
    - Outlook contacts 426
  - ListName, contacts 431
  - logging off of Outlook 152, 157
  - logging on to Outlook 152, 154
  - login, displaying dialog 158
- M**
- macro
    - Cancel key in 67
    - error handling 326
    - interruption handling 327
    - Word 326
  - mail item
    - attachment list 245
    - attachments 245
    - autoforwarding 245
    - clearing conversation thread 245
    - creating 125, 244, 245
    - data fields 245–251
  - mail item (continued)
    - deleting 246, 249, 250
    - forwarding 247
    - inspector 246, 255
    - interface 245, 247
    - printing 248
    - recipients 248
    - replying 249
    - saving 249
    - sending 125, 244, 249
    - sensitivity 250
    - size 250
  - mail merge
    - data source 373
    - document 343, 373
    - executing 340
    - interface 374
    - set document 374
  - mail service startup 127

- maintaining collection items 434
  - manipulating slides 377
  - manual, organization of 17
  - MAPINameSpace
    - assigning 130
    - interface 131, 153
    - object 124
  - marking
    - grammatical errors 350
    - spelling errors 350
  - marshalling
    - automatic 21
    - defined 30
    - parameters 21
  - master-detail relationship 425
  - memo fields 423
  - memo, controlling behavior 425
  - message class
    - appointment item 189
    - note item 256
    - post item 265
    - task item 281
  - messaging address 164
  - Microsoft Developer Network 4
  - miscellaneous data
    - contact item 301
    - data fields 302–304
  - model, requesting data from 420
  - modified presentations 388
  - modifying recipients 159, 160
  - mouse tips Assistant 409
  - moving Assistant 409
  - moving contact item 222
  - MSDN 4
- N**
- name
    - document 344
    - remote computer 446
    - shape 372
  - NameSpace
    - data source 126
- O**
- object
    - application 2
    - hierarchy 13
    - MAPINameSpace 124

- object model
  - Excel 33
  - PowerPoint 375
  - Word 313
- obtaining contact list 427
- OCX controls 27
- Office version 447
- OLE
  - controls 27
  - data types 21
- OLE datatypes 25
- on-line help 18
- opening
  - document 315, 320, 327, 328, 330, 335, 340, 347, 353, 416
  - existing presentations 377, 381
  - new presentations 377
  - presentations 377, 378, 380, 381, 385, 387
  - Word files 324
  - workbook 37, 83
- organization of manual 17
- out of process servers 29
- Outlook
  - adding recipients 164
  - application interface 147, 158
  - Application object 124
  - appointment item reminder 192, 193
  - appointment items 124
  - assigning MAPINNameSpace 130
  - attachment file name 176
  - attachment, display name 176
  - Automation server 124
  - conflicting commitments 180
  - connecting to 124, 130, 145, 152, 154, 157, 158
  - Contact items 124
  - counting attachments 172
  - creating calendar items 131
  - creating items 126
  - creating session 154
  - default item folder 426, 431
  - default item type 426
  - defining task item recurrence 279
  - deleting post item 263
  - disconnecting 152
  - dispatching events 152
  - displaying address entries 165
  - displaying default folders 126
  - displaying Explorer 145
  - displaying Item Inspector 145
  - displaying login dialog 158
  - Distribution list items 124
  - editing item properties 128
  - Explorer 127
  - file type 196
  - folder interface 153
  - folder type constants 125
  - folder, default item type 125
  - folders 124
  - form interface 185
  - getting default folders 128, 130
  - internal version 191, 265, 281
- Outlook (continued)
  - Item Inspector 127
  - item types 124
  - Journal items 124
  - linked attachment, path of 176
  - loggin off 152
  - loggin on to 152
  - logging off 157
  - logging on to 154
  - Mail items 124
  - mail service startup 127
  - Note items 124
  - organization of 124
  - Post items 124
  - product information 151, 157
  - profile name 152
  - profile password 152
  - recipient tracking status 169
  - routing e-mail 132
  - saving contact item 227
  - saving note item 257
  - saving post item 268

- saving task item 287
- sending e-mail 132, 158
- server 124, 431
- specifying recipient address 164
- starting session 154
- task item sensitivity 288
- Task items 124
- terminating 157
- user interface 145
- user profile 127
- version 191, 265, 282
- version information 151, 157
- virtual card 212
- Outlook Contact Manager 430
- out-of-process servers 21
- overriding default reminder, task item 283
- owner
  - collection 437
  - collection item 436
- P**
- package files 10
- parameters
  - empty 4
  - marshalling 21
  - optional 4
- personal data
  - contact item 310
  - data fields 311–312
- picture, slides 392
- populating 357
  - document 3
  - from datasets 57
  - range 106, 418, 422, 423
  - ranges 59, 95, 103
  - table 318, 415, 416, 418, 423
  - tables 358, 359
- position
  - note item 256, 258
  - of chart 112
  - of shape 370, 371, 372
  - of text box 371
- Word server 331
- positioning
  - Assistant 404
  - PowerPoint 383
- post item
  - aging 263, 265
  - attachments 260
  - clearing conversation thread 260
  - closing 261
  - conversation thread 261
  - copying 262
  - creating 150, 259, 262
  - data fields 260–270
  - deleting 150, 263
  - displaying 263, 264, 280
  - forwarding 263
  - indexing conversation thread 261
  - interface 266
  - invalidating 263
- post item (continued)
  - message class 265
  - posting 259, 266
  - printing 266
  - replying 267
  - saving 150, 259, 261, 267, 268
  - sensitivity 269
  - size 269
- posting, post item 259, 266
- PowerPoint
  - accessing directly 379
  - active presentation 381
  - application interface 382, 386, 387
  - application object 375
  - charts 376
  - connecting to 377, 383, 393
  - default file names 395
  - dimensioning 382, 383
  - graphs 376
  - object model 375
  - positioning 383
  - presentations 375, 384
  - server 382

- setting caption 381
- shapes 375, 376
- sizing 382, 383
- slides 375
- tables 376
- textboxes 376
- visibility 377, 383
- visibility of 393
- presentation** 380
  - collection 375
  - collections 382
  - speed 400
- presentation file name** 387
- presentation files (\*.ppt)** 375
- presentations**
  - accessing 380, 385
  - active 380, 381
  - adding 377, 385
  - advance time 398
- presentations (continued)**
  - advancing 387, 397
  - closing 378, 386
  - creating 377, 381
  - customizing 379
  - file name 381, 388
  - indexing 385
  - modified 388
  - opening 377, 378, 380, 381, 385, 387
  - opening existing 377, 381
  - opening new 377
  - PowerPoint 375, 384
  - quitting 382
  - running 378, 386, 388
  - saving 378, 386, 388
  - timing 387
- presenting Assistant help topics** 408
- printing**
  - appointment item 191
  - contact item 226
  - document 322, 344
  - journal item 235, 239
  - mail item 248
- note item** 256
- options in document** 344, 345
- post item** 266
- preview** 330
- task item** 283
- workbooks** 64
- worksheet** 90
- process space** 21
- profile**
  - name 152
  - password 152
- ProgID** 23
- programs, example** 10
- properties**
  - persisting** 2
  - reading** 1
  - writing** 1

**Q**

- querying collections** 438
- QueryInterface** 30
- quitting**
  - presentations** 382
  - Word** 328

**R**

- range**
  - address 108, 116
  - addresses 94
  - anchor cell 94, 95
  - borders 96, 97
  - cell color 98
  - column width 99
  - connecting to dataset 425
  - data address 115
  - end 346
  - interface 95
  - linking dataset 106
  - linking to dataset 103
  - naming 102
  - populating 95, 99, 103, 106, 422, 423
  - selecting 99, 107

- setting address 99
- specifying 314
- start 346
- text 314
- type 346
- value 108
- Word 313
- Range object 34, 35
  - collection of cells 35
  - column 35
  - multiple worksheets 35
  - row 35
  - single cell 35
- ranges
  - accessing 75, 92
  - activating 94, 113
  - adding 91, 92
  - collection of 90, 91, 92
  - creating 59
  - defining 93
  - indexing 75, 92
  - naming 59, 75, 93
  - populating 57, 59, 418
  - referencing 93
- reading data 418
- reading from disk 423
- reading properties 1
- receiving event notification 444
- recipient
  - creating list 183
  - deleting 162
  - interface 163, 166
  - racking status 169, 171
  - resolving 162
  - resolving type 167, 168
  - specifying address 164
  - type 167
- recipient list
  - creating 161
  - interface 161
- recipients
  - accessing 159, 160, 161, 191, 248
- adding 159, 160, 164, 166
- collecton of 245
- collection 159, 160
- collections 191
- deleting 159, 160, 164, 166
- indexing 161, 166
- mail item 248
- modifying 159, 160
- naming 166
- task item 283
- recurrence 192
  - of appointment item 192
  - task item 288
- reference section 18
- reference-counting 30
- referencing
  - cells 95
  - columns 94
  - ranges 95
  - rows 94
- referencing cells 94
- referencing ranges 93
- registry key 23
- relationships, master-detail 425
- reminder
  - appointment item 192, 193
- reminder, appointment item 193, 196
- remote computer name 153, 446
- remote server name 69
- removing Trial-Run Edition 6
- replacing text 336, 347
- replacing, text 316
- replying
  - journal item 240
  - mail item 249
  - post item 267
- requirements
  - hardware 5
  - software 5
- resetting task item 274
- resolving recipient 162
- resolving recipient address type 167, 168

- restoring collection items 434
- retrieval functions, implementing 421
- retrieval modes
  - specifying 420
  - supporting 421
- retrieving 434
  - interfaces 434
  - values 430
- returning data 420
- revisions, tracking 353
- rmCell 421
- rmPacket 421
- root collection 436
- routing 132
- routing slip
  - creating 342
  - deleting 342
- row
  - autofit 53, 96
  - deleting 66
  - formatting 96
  - height 53
  - indexing 420
  - inserting 66
  - referencing 94
  - sizing 53, 96
  - width 96
- rows
  - column count 421
  - returning column counts 421
  - specifying count 420
  - variable length 421
- running 388
  - presentations 378, 386
  - slide show 386
  - slide shows 377, 388
- run-time binding 25
- S**
- safecall 25
- saving
  - appointment item 145, 181, 195, 196
- contact item 146, 205, 212, 226, 227
- data 348
- document 316, 323, 325, 348
- fonts 349
- journal item 147, 233, 240
- mail item 249
- note item 149, 253, 257, 258
- post item 150, 259, 261, 267, 268
- presentations 378, 386, 388
- task item 151, 275, 286, 287
- workbooks 65, 68, 79, 82, 84
- worksheets 68, 86
- screen painting, document 330
- screen tips, displaying 326
- scroll bar, displaying 326
- selecting
  - Assistant 401, 403, 407
  - cells 107
  - range 107
  - slides 393
- sending
  - appointment item 197
  - e-mail 3, 158
  - mail item 125, 244, 249
  - task item 288
- sensitivity
  - appointment item 197
  - journal item 241
  - mail item 250
  - post item 269
  - task item 288
- server
  - accessing Word 317
  - activating 1
  - automation 1, 21, 22, 438, 443
  - automation status 445
  - connecting to 65, 81, 88, 94, 113, 445
  - creating 66, 445
  - Excel 75
  - Outlook 124, 431
  - position 75, 76
  - postition 75, 76

- PowerPoint 382
- related interfaces 438
- remote, name of 69
- size 75, 76
- status 445
- visibility 76, 94, 113
- window size 77
- Word dimensions 331, 332
- Word position 331
- servers
  - creating 443
  - design time access 3
  - in-process 21
  - launching 3
  - out-of-process 21
  - shutting down 3
- setting PowerPoint caption 381
- shape
  - accessing 369
  - activating 371
  - adding 349, 369
  - anchoring 370
  - collection 349
  - creating 369
  - deleting 369
  - indexing 369
  - name 372
  - positioning 370
  - position 371, 372
  - Word 313
- shapes
  - PowerPoint 375, 376
- shapes
  - accessing 349
- showing balloon messages 412
- shrink to fit 108
- size
  - mail item 250
  - note item 258
  - of appointment item 198
  - of contact item 228
  - of journal item 242
  - of text box 371
- post item 269
- task item 288
- sizing
  - Assistant 410
  - PowerPoint 382, 383
- slide collection 375
- slide shows
  - advancing 387, 397
  - displaying 3
  - running 377, 386, 388
  - timing 387
- slides
  - accessing 386, 389, 391
  - activating 393
  - adding 389, 391, 395
  - advance time 398
  - animation 376, 378, 395
  - animation effects 392, 399
  - bulleted list 392
  - chart 392
  - collection of 389
  - collections 386, 390
  - color 392
  - content 375
  - controlling 375
  - controlling animation 396
  - controlling delay 398
  - creating 391
  - creating transition 398
  - customizing background 394
  - deleting 391
  - displaying background objects 394
  - drawing 392
  - effects 376
  - font settings 392
  - formatting 375, 392, 394
  - ID 393
  - indexing 391
  - interface 393
  - manipulating 377
  - naming 395
  - picture 392

- PowerPoint 375
    - selecting 393
    - specifying layout 394
    - title text 392
    - transition delays 400
    - transition effects 395
  - software requirements 5
  - sorting
    - cell 314
    - tables 357
  - sounds, Assistant 411
  - source files 7
  - specifying
    - anchor cell 423
    - class name 436
    - contacts list source 426
    - dataset model 358
    - field 430
    - folder path 427, 431
    - path 431
    - range 314
    - retrieval mode 420
    - slide layout 394
    - Word caption 324
  - starting Outlook session 154
  - starting Word 329
  - startup folder, Word 332
  - state, of client 444
  - status, of server 445
  - suggested reading 31
  - system information 445
- T**
- tab stop interval 339
  - table
    - accessing 352, 355, 356, 357
    - activating 358
    - adding 314, 318, 352, 355, 356
    - collection 352
    - converting 357
    - creating 355, 356, 415, 416
    - creating from text 325
  - deleting 356
  - document 318
  - formatting 318, 357
  - indexing 356
- table (continued)
- obtain data 415
  - populating 318, 357, 358, 359, 415, 416, 418, 422, 423
  - PowerPoint 376
  - sorting 357
  - updating 417
  - Word 313, 314
- table collection 355
- task item
  - aging 281
  - assigning 273
  - attachments 273
  - closing 275
  - conversation thread 276
  - copying 276
  - creating 151, 271, 277
  - data fields 273–291
  - deleting 151, 278
  - deleting recurrence 274
  - displaying 279
  - indexing conversion thread 276
  - inspector 279
  - interface 290
  - message class 281
  - overriding default reminder 283
  - printing 283
  - recipients 283
  - recurrence 288
  - resetting 274
  - saving 151, 275, 286, 287
  - sending 288
  - sensitivity 288
  - size 288
  - time spent 273
- task, defining recurrence 279
- technical support
  - knowledgebase 19

- newsgroups 19
  - options 19
  - templates and add-ins, Word 332
  - terminating
    - Outlook 157
    - Word 328
  - text
    - adding 313
    - background objects 394
    - converting to table 325
    - deleting 313
    - document 315
    - finding 316, 336, 341
    - formatting 313
    - inserting 343
    - insertng 336
    - range 314
    - replacing 316, 336, 347
    - retrieving 354
    - shrinkto fit 108
    - wrap 109
  - text box
    - adding 371
    - postion 371
    - sIze 371
  - text value 108
  - textboxe
    - PowerPoint 376
  - time spent, task item 273
  - timing
    - presentations 387
    - slide shows 387
  - title text
    - slides 392
  - TOpSlides collection 390
  - tracking revisions 353
  - tracking status, recipient 169, 171
  - transition delays, slides 400
  - transition effects, slides 378, 395
  - Trial-Run Edition, uninstalling 6
  - triggering events 446
  - type libraries 3, 26
  - type library
    - defined 30
    - importing 24
  - type of chart 114
  - typecasting interfaces 3
- U**
- uninstalling Trial-Run Edition 6
  - updating table. 417
  - user profile
    - Outlook 127
    - specifying 127
    - storing 127
  - userproperties, contact item 230
- V**
- variable length rows 421
  - variant 30
  - variant array 415
    - initializing 416
  - Variants 1
  - variants 22
  - version 447
  - version information, Outlook 157
  - version, Outlook 191, 265, 282
  - virtual card 212
  - visibility
    - PowerPoint 377, 383, 393
    - Word 315
  - visibility of server 76
  - visible Assistant 411
- W**
- window size 77
  - wizard Assistant 405
  - Word
    - accessing 313
    - accessing server interface 317
    - activating 329
    - automation 313
    - bookmarks 313
    - browsing files 324

- closing document 315
- connecting 338
- connecting to 315, 327, 358, 371
- Word (continued)
  - connecting to 363, 446
  - controlling visibility 363, 371
  - controlling visibility 358
  - deactivating 329
  - document 313
  - file 325
  - formatting text 315
  - hyperlink 313
  - linking datasets 422
  - macro behavior 326
  - macro errors 326
  - object model 313
  - opening document 315
  - opening files 324
  - quitting 328
  - range 313
  - server interface 317
  - sever dimensions 331, 332
  - sever position 331
  - shape 313
  - specifying caption in 324
  - starting 329
  - startup folder 332
  - table 313, 314
  - templates and add-ins 332
  - terminating 328
  - tracking changes 318
  - transferring data to 416
  - visibility 315
- workbook
  - accessig 79
  - activating 69, 81, 89, 99, 114, 121
  - adding 36, 56, 79, 82
  - adding worksheets 86
  - closing 37, 64
  - collection 34, 77
  - connecting to 81
  - creating 36, 69, 79, 82
  - default 34
  - destroying 89
  - editing 68
  - hidden 34
  - indexing 79
  - interface 81
  - naming 36
  - opening 37, 83
  - printing 64
  - recipient 38
  - routing 38
  - saving 36, 65, 79, 82, 84
  - saving changes 37
  - visible 34
- Workbook object 34
- worksheet 34
  - accessig 86
  - accessing 84
  - accessing charts 111
  - accessing hyperlink 118
  - activating 88, 89, 94, 99, 113, 114, 121
  - adding 56, 84, 86
  - adding charts 110, 111, 112
  - adding hyperlink 118
  - adding ranges 90, 92
  - collecton of 84
  - collection 87
  - collection of 34
  - connecting datasets 54
  - creating 87
  - default 87
  - destroying 89
  - editing 68
  - filling 3
  - indexing 84, 86
  - indexing charts 111
  - indexing hyperlink 118
  - interface 88
  - naming 90
  - printing 90
  - saving 68, 86

wrap text 109  
writing properties 1

[www.turbopower.com/search](http://www.turbopower.com/search) 19  
[www.turbopower.com/support](http://www.turbopower.com/support) 19  
[www.turbopower.com/tpslive](http://www.turbopower.com/tpslive) 19