

Understanding Outlook e-mail account options

Microsoft Outlook supports a variety of e-mail server types and communicates with these servers in different ways.

A MAPI-based architecture is provided if you use Outlook with Microsoft Exchange Server or with a MAPI-based store, such as a <u>Personal Folders file</u> (.pst). In general, the Outlook object model and developer-related features were designed for this environment. MAPI uses profiles to configure how an e-mail message is transmitted and where it is stored. Exchange server also provides many of the collaborative capabilities in Outlook, such as the Organizational Forms Library and public folders.

If Outlook is configured with an <u>Internet service provider (ISP</u>), typically using the POP protocol, you can create many types of Outlook solutions. However, some of the topics in this documentation apply only when Outlook is used in conjunction with Microsoft Exchange Server, because that is the mode most commonly used to develop solutions based on Outlook.

Note Developing custom forms with Outlook configured with only an HTTP-based server is not supported.



Design Outlook Workgroup Solutions

With Microsoft Outlook, you can create a variety of workgroup solutions and forms. You can create Workgroup Solutions by using custom views in a Microsoft Exchange <u>public folder</u>. You can also create simple forms with no programming involved or create advanced forms by using custom controls, properties, and Microsoft Visual Basic Scripting Edition (VBScript).

There are a few basic approaches to creating Outlook solutions. You can use existing <u>items</u>, such as tasks or appointment items, put the items in a public folder, and assign a view to the folder. This method gives you an instant Workgroup Solution without writing any code. Another method is to modify an existing item, such as a contact entry or e-mail message, by adding additional pages and fields. With this method, you can extend the use of the item by adding the fields and pages you need without writing any code. To create advanced forms, you can use the <u>Control Toolbox</u> and VBScript to access properties, events, methods, and objects within Outlook.

What method do you want to use?

Create an instant workgroup solution using public folders

Extend the use of an Outlook item by adding additional fields and pages

Create custom forms by using VBScript

Create forms by using Office documents

Overview of a Typical Folder-based Solution

When used with Microsoft Exchange Server, Microsoft Outlook provides groupware capabilities that allow more than one person to work on the same data. Even without using these capabilities, however, you can still customize Outlook folders for your personal use or to create a solution that other people in your organization can use.

If you're planning to build a groupware solution, you should consider first working with Outlook to develop a single-user solution. Since a groupware solution is often as simple as placing a custom form in a Microsoft Exchange public folder, learning the basics of building a folder-based personal forms solutions will be fully applicable to developing solutions in public folders.

Whether it's in one of your personal folders or in a Microsoft Exchange public folder, putting together an Outlook solution typically involves the following steps:

- 1. Decide which type of folder and/or form to customize. For more information, see <u>Types of Forms in Outlook</u>.
- 2. Open a new form and customize it to suit your needs.
- 3. <u>Publish the form to the folder</u> so that it's available only for use in that folder.
- 4. <u>Set the custom form as the default form</u> for the folder. This will ensure that when you create new items for use in this folder they'll be based on the custom form you created.
- 5. If you have any existing items in the folder, update these items so they will use the new form when they're opened. For more information, see <u>Change</u> <u>the Form Used by Existing Items in a Folder</u>.

Note If you plan to use a custom form in a folder, you should create the custom

form before creating any items based on the form. This will ensure that all the items in the folder use the same set of fields.

If you need to update the form after having created some items in the folder, follow these steps:

- 1. Open a new, blank item based on your custom form.
- 2. Update the form to suit your needs.
- 3. Republish the form to the folder using the same name.

Using views to create a solution

Although custom forms provide a powerful way to work with data in a folder, you can also customize Microsoft Outlook to a great extent simply by creating a view that suits your needs. In some cases, depending on how you want to work with Outlook, you can use a custom view instead of a custom form to enter and modify data.

Entering data in a view is usually done be selecting one of the Outlook table views, wherein each row contains one item in the folder and each column represents a field associated with an item. You can select the fields that Outlook displays, and you can also enter or change the contents of fields directly in the table view. This allows you to quickly modify items, even those that contain custom fields.

To allow field editing in a view

- 1. Click **View**, point to **Arrange By**, then point to **Current View**, and then click **Customize Current View**.
- 2. Click **Other Settings**.
- 3. Select **Allow in-cell editing**.

Editing items in a table view lets you keep track of custom data, view multiple items at the same time, enter data into an item without having to open the item, and avoid having to create a custom form.



Change the default view in a folder

- 1. If the Folder List is not visible, click the **View** menu, and then click **Folder List**.
- 2. Right-click the folder you want to change the default view for.
- 3. Click **Properties** on the shortcut menu.
- 4. Click the **Administration** tab.
- 5. **Note** If the **Administration** tab does not appear, you don't have owner permission.
- 6. In the **Initial view on folder** box, enter the view that you want to appear each time the folder is opened. The **Normal** view displays the default Microsoft Outlook view for the type of items in the folder.

To change the default view for a <u>public folder</u>, you must have owner permission for the folder, and there must be at least one custom view defined that is visible to everyone for the folder.

Show only the custom views in a folder

- 1. On the **View** menu, select **Current View**, click **Define Views**.
- 2. Select the **Only show views created for this folder** check box.



Set permissions in a folder

If you are the owner of a public folder, you can set permissions in the folder to give users the ability to perform certain functions. By adding the user names, distribution list names, or public folder names to the **Name** box on the **Permissions** page, you can determine who can:

- Create items
- Read items
- Create subfolders
- Edit items
- Delete items

You can also determine who is the folder owner and the folder contact and to whom the folder is visible.

To set permissions

- 1. In the **Folder List**, right-click the public folder for which you want to set permissions, and then click **Properties** on the <u>shortcut menu</u>.
- 2. Click the **Permissions** page.
- 3. To add a user name to the list, click **Add**.
- 4. Choose a name from the <u>Address Book</u>, and then click **OK**.
- 5. Select the user's name in the **Name** box for whom you want to change permissions.
- 6. In the **Roles** box, choose the role for this user. The permissions check boxes will automatically revert to the default values for this role.
- If you want to define a custom role, select the permissions check boxes that you require for this user. The value in the **Roles** box will change to "Custom."
- 8. When you have set the roles for all the users, click **OK** on the **Properties** page.

Note To remove a name, select the name, and then click **Remove**. You can remove any name from the **Name** box except Default.



Create a custom field

In Outlook, you can create different types of custom fields based on the data type that you select.

- If you want to enter and show information about an <u>item</u>, you can create a simple field with one data type. For example, if you want to enter a date, you can create a Date/Time field.
- If you want to show more than one field in a column in a table (or in a row in a card), you can create a combination field with a Combination data type. You can either show all the selected fields joined together in a column or show only the first non-empty field. This is useful if you want to optimize the use of screen space in a view.
- If you want to show the results of calculations on standard fields or fields you create, you can create a formula field.

What do you want to do?

Create a simple field for a folder

Create a combination field for a folder

Create a formula field for a folder



About Outlook forms

A form is an easy way to distribute and collect information electronically. For example, a form can be available for anyone to order supplies or post information in a <u>public folder</u>. Microsoft Outlook provides built-in forms such as the Message form to compose messages and the Contact form to enter information for a contact. If you have sent an e-mail message or created an appointment, you have used a form. Every Outlook <u>item</u> is based on a form.

You use the Outlook Forms design environment to create custom forms. Use a built-in form to leverage functionality from the item associated with it; for example, to include automatic name checking in a custom form, create it from a form based on an e-mail message. To modify a form, you can add and remove fields, controls, options, and tabs. A form can be saved as a file for use as a template or in another program or in a forms library to make the form available to others.

Developers can extend forms with Microsoft ActiveX controls and put World Wide Web pages on their forms by using the Web Browser control. Forms can be programmed by using <u>Microsoft Visual Basic Scripting Edition (VBScript</u>).

You cannot design Outlook forms in Microsoft Visual Basic or Visual Basic for Applications (VBA), although you can use forms created with the Microsoft Exchange Electronic Forms Designer (EFD) in Outlook. In order to use EFD, you must install an optional EFD run-time component that you can download from the Microsoft Office Download Center.

How Outlook forms and items work together

Microsoft Outlook stores its information in the form of individual items in a folder. An Outlook item is similar to a record in a database in that it consists of a group of fields that store information pertaining to the specific item.

Outlook displays the contents of an item in one of two ways: either through a view in an explorer window, or through a form in an inspector window. A form usually provides a more complete display of the information and lets the user interact with the contents of the item in more ways. In a sense, a form is the principal user interface for an item. Outlook provides one or more standard forms for each type of item (mail message, contact, and so on). You can create customized versions of these forms to change the way Outlook displays items. You can display additional pages that are usually hidden, and you can add controls to those pages. Typically, these controls are bound to fields in the item so that the user can view and edit the contents of those fields. In some cases, you can also customize the default pages of a form.

Every item contains a Message Class field; this field contains the name of the form that Outlook provides to view and edit the item. For example, a contact item will have a default message class of IPM.Contact. If you create a custom form called Customer, the Message Class field of items using that form will contain IPM.Contact.Customer. The message class of all Outlook items always begins with IPM. The second part of the message class denotes the type of Outlook form that the form is based on. The third portion of the message class is present only if the form is a customized version of a standard Outlook form.

Types of forms in Outlook

To create a custom form in Microsoft Outlook, you always begin by choosing one of the Outlook default forms. In most cases, the type of form you want to customize is determined by what type of solution you are creating. The following table lists the types of forms and how they would typically be used.

Form	Description	
Contact	Use to keep track of information about a person or organization.	
Distribution List	Use to create a lists of contacts and e-mail addresses that can be used as a single e-mail address.	
Task	Use to track information about a task that needs to be accomplished.	
Mail Message	Use to send information to someone in a specific format, or to provide a means for them to enter data so that they can send it somewhere.	
Post	Use to facilitate a threaded conversation in a Microsoft Exchange public folder, or use for other purposes, such as posting file attachments to a folder.	
Appointment Use to represent a meeting or scheduled event.		
Journal Entry	Use to log information about another item or an event.	

Following are some things to consider when deciding which type of form to use:

- What is the nature of the solutions you are creating? If the form will be sent, then an e-mail message would be appropriate. If the solution will be used to track information in a folder, then a post, contact, task, or appointment form would be more appropriate.
- Which form most closely provides the needed functionality? The default Outlook forms may provide much of the functionality, so you should choose the one that would require the least customization.
- Which standard fields are available for the form? Each Outlook form has its own set of standard fields to keep track of information for those types of items. You should consider using the form whose fields most closely match

you needs.

Which pages of the form are customizable? Some pages on Outlook forms are not customizable, although you can hide them if you want.



Change the default form for a folder

You can change the default form for a particular folder so that when a new item is created, it will use a custom form. This procedure requires Owner permission for a public folder.

- 1. Right-click the folder for which you want to specify the default form, and then click **Properties** on the shortcut menu.
- 2. In the **When posting to this folder, use** box, click the form that you want others to use.

Ti	n
11	Ľ

To copy, install, delete, and update forms in this folder, click **Manage** on the **Forms** tab. You can also specify available forms in a public folder by selecting an option under the **Allow these forms in this folder** box.

Learn how to change the form for existing items in a folder.

Globally change a default form

You can change the default form in Microsoft Outlook by making changes to the Microsoft Windows registry. The registry settings specify which forms are substituted for the default Outlook forms. For example, if you create a custom form called "Default," that custom form has a message class of IPM.Note.Default, instead of the standard Outlook e-mail message class of IPM.Note. You can add certain registry keys to indicate that Outlook should substitute the IPM.Note.Default form for the standard IPM.Note form.

The Forms Administrator utility has not been updated for Microsoft Office Outlook 2003, and it does not create the registry keys in the correct location for Office Outlook 2003. However, you can use the Forms Administrator utility to create a Windows registry file that requires only minor changes to work with Office Outlook 2003. To use a Windows registry file to change the default form in Office Outlook 2003:

- 1. Download the Outlook 2000 Forms Administrator utility.
- 2. Run the Forms Administrator utility, and then change the settings as you would for Outlook 2000 or Outlook 2002.
- 3. To save the registry settings on your computer, click **Save**. This also makes the **Export Saved Settings** button available.
- 4. Click Export Saved Settings to save a Windows registry (.reg) file.
- 5. Open the .reg file in a text editor, such as Notepad.
- 6. The registry key paths reference 9.0, which is the location for Outlook 2000 registry settings. Change all of the references to 9.0 to 11.0, which is where Office Outlook 2003 stores registry settings.
- 7. Save the .reg file.
- 8. To change a default form for Office Outlook 2003, run the .reg file on the computer so that the keys are added to the Windows registry.

Note If you used the Forms Administrator utility to change the default forms in Outlook 2000 or Outlook 2002, and you then upgrade to Office Outlook 2003, Microsoft Office or Outlook Setup migrates the registry keys to the correct location so that Office Outlook 2003 continues to use the substituted forms.

Change the form used by existing items in a folder

In some cases you may need to change the form associated with items that are already in a folder. This is often necessary after importing items, or if you create a custom form after you have already created items based on a standard Microsoft Outlook form.

The Message Class field cannot be directly changed using the Outlook user interface, but you can use VBScript, Visual Basic, or Visual Basic for Applications to change the Message Class field.

The following Automation code can be used as a basis for developing your own solution. This code assumes that the name of the new form is MyForm. It will change all contacts in your default contacts folder so that they will use MyForm.

```
Sub ChangeMessageClass()
Set olApp = New Outlook.Application
Set olNS = olApp.GetNameSpace("MAPI")
Set ContactsFolder = _______
olNS.GetDefaultFolder(olFolderContacts)
Set ContactItems = ContactsFolder.Items
For Each Itm in ContactItems
If Itm.MessageClass <> "IPM.Contact.MyForm" Then
        Itm.Save
        End If
Next
End Sub
```

Note If you want to use a folder other than a default folder, use the **Folders** collection object to refer to any folder that is available in your Folder List.

Create an Outlook form

All custom forms must be based on existing Microsoft Outlook forms. If you do not wish to use the default form functionality provided by Outlook, you can hide all of the default form pages and use only the extra blank pages to create a complete custom form.

To begin designing an Outlook form:

- 1. Open an Outlook item by doing one of the following:
 - To open a mail message, click the **New Mail Message** icon.
 - Click the arrow beside the **New Mail Message** icon, and then click **Mail Message**.
 - On the **File** menu, click **New**, and then click **Mail Message**.
 - On the **Actions** menu, click **New Mail Message**.
 - Double-click anywhere in an empty area of the **Inbox** view pane.
- When you have the new form opened, on the **Tools** menu of the form, click **Forms**, and then click **Design This Form**. You may also click **Design Form**, and then select to design a different form.
- 3. Add the fields, controls, and code that you want to your new form.

Customize pages on a form

In general, you can customize up to five pages on a form. For e-mail messages and posts, you can customize a sixth page, the Message page; and for contacts, you can customize the General page.

With the form in design mode :

- 1. Click the page that you want to customize.
- 2. From the **Field Chooser**, drag a field to the page. Outlook automatically sizes the field for you.

If the field that you want doesn't appear in the **Field Chooser**, click a different field set in the box at the top of the **Field Chooser**.

- 3. To align, size, or remove fields, right-click the field, and then select the options that you want on the shortcut menu.
- 4. <u>Add and remove controls</u>.

Tips

- To create fields, click **New** in the **Field Chooser**. Type a name for the field in the **Name** box, and then select a field type and format. Fields support calculated expressions, <u>validation formulas</u>, and field formatting.
- If you drag a field or control to a hidden page, the page automatically shows on the form at run time.

Extend the use of an Outlook item by adding additional fields and pages

In this approach, you take a standard Outlook item, such as a contact, mail message, task request, or meeting request and add fields and extra pages.

Note All custom forms must be based on existing Outlook forms. If you do not wish to use the default form functionality provided by Outlook, you can hide all of the default form pages and use only the extra blank pages to create a complete custom form.

- 1. Open the item on which you want to base the form. For example, if you want your form to organize and collect information about a person or organization, open a contact. In the contact, select **Forms** on the **Tools** menu and click **Design This Form**. This opens the item in design mode.
- 2. Set attributes for the form. In design mode, you add attributes such as a caption, password, or description on the **Properties** page. The **Actions** page lists default Reply forms. You can add your own custom Reply forms to the **Actions** page as well as change several aspects of how the default Reply forms work.
- 3. Design the form by moving, removing, and adding fields and pages in the form. All the fields in the form and all the fields you add from the **Field Chooser** retain their built-in properties and functionality. For procedures on designing form pages, see **Creating Forms** under **Solutions with Forms, Public Folders, and Views** on the Help **Contents** tab.
- 4. <u>Test and publish</u> your form. You can instantly see how your form looks; just open your new form in run mode by clicking **Run This Form** on the **Form** menu on the form. You can publish your form to your Personal Forms Library or to a folder to test it, and then publish the form to a public folder or to the Organization Forms Library so other users can use it.

Use the Field Chooser

Although you can add a control to a form and then bind the control to a field, it is usually more effective to use the **Field Chooser**. The **Field Chooser** lets you add a control and bind it to a field in a single step. Also, in many cases, the built-in control provided by the **Field Chooser** has special features and capabilities that cannot be duplicated by binding a field to a standard control.

Microsoft Outlook displays the **Field Chooser** whenever you open a form in design mode.

Note Some entries in the **Field Chooser**, such as Categories, display ellipses after them. These entries correspond to Outlook windows. When you drag one of these entries in the **Field Chooser** to a form page, Outlook creates a button that can be clicked to open the window.

How to open a form

You can open custom forms that you create in Microsoft Outlook. How you open a custom form depends on where you saved it.

To open a form	Do this		
In the Folder	Select the folder in which you saved the form. On the Actions		
Forms Library menu, select the form that you want to open.			
In the	On the File menu, click New , and then click Choose Form . If		
Organizational necessary, on the Choose Form dialog box, click			
Forms Library Organizational Forms Library in the Look In box.			
In the Personal Forms Library	necessary on the ('hoose Horm dialog boy click Personal		
Saved as an Outlook template	On the File menu, click New , and then click Choose Form . If necessary, on the Choose Form dialog box, click User Templates in File System in the Look In box.		
Saved as a file	On the Tools menu, click Find . In the Find pane, click Advanced Find . In the Look for box, click Files (Outlook/Exchange) , and then type the name of the file in the Named box.		

How to edit a form

1. Open the form .

How?

- 2. On the **Tools** menu, select **Forms** and click **Design This Form** to switch to the design mode of the page.
- 3. Make your changes.
- 4. Save the form.

How?



Create custom actions for a form

Actions are a very powerful feature of Microsoft Outlook forms that allow you to create a wide variety of solutions without writing programming code. For example, you can use actions to open other forms, even forms of a different type, or you can create a message form with an action that opens a contact form.

With the form in design mode:

- 1. Click the **(Actions)** page.
- 2. Click New.
- 3. Select the options that you want.

Notes

- You can specify whether you want an action to appear as a command on both the **Actions** menu of the form and the **Form Design** toolbar, or just on the item menu.
- You can also use actions to create a new <u>item</u>. For example, you can create a message form with an action that creates a new contact. The new item is created in the open folder, not in the **Contacts** folder.
- You need to use the **Forward** action to have attachments included in a Reply. So even if your Action is a **Reply Action**, use the **Forward** method to preserve attachments.
- There is no direct way to have the original item automatically close when the new item is launched. You need to use Microsoft Visual Basic Scripting Edition (VBScript) to accomplish this.

Create a reply form

When you create a Message form , you often want users to send a reply by using a reply form.

An action automatically opens a reply form, typically when a user clicks **Reply**, **Forward**, or another command on the form. For example, an action can create a **Forward** button, that, when selected by the user, opens a Forward form for the user to fill in and send to reply to your message. You can specify that the form use standard Reply and Forward headers, and you can even add prefix text, such as RE: to the Subject field of your custom reply form.

You can use the default reply forms included with Outlook (these are the forms listed on the **Actions** page), or you can <u>disable</u> any of these forms and create a custom reply form. The custom reply forms that you create are listed on the **Actions** page in design mode in the form.

You can create custom buttons and menu commands and specify what happens when the user chooses them. You can also create a button or menu command that opens a task, a Calendar item, or even a note.

You can send information without opening a reply form by pre-addressing a custom reply form to go to a specific e-mail address or to a public folder.

What do you want to do?

<u>Create a custom reply form</u>

Add a custom button or menu command to start a reply form

Open a task, Calendar item, or note from a custom form

Use standard Reply and Forward headers on a form

<u>Automatically insert text into the Subject field of a reply form</u>

Specify whether a form prompts the user, opens, or is sent

Pre-address a reply form to a public folder or an e-mail address

About the form name and message class

The message class is an internal identifier used by Outlook and Microsoft Exchange to locate and activate a form .

The <u>MessageClass</u> property corresponds to the MAPI property PR_MESSAGE_CLASS. This class property determines which receiving folder the message should be routed to and which form should be activated to view the message.

In the **Publish Form As** dialog box, when you type a name in the **Display name** field, you will notice that the **Form name** field reflects the display name by default. You can leave the form name to be the same as the display name or you can change the form name. The display name will be the caption at the top of your form. The display name will also be used to construct the name under which your form will be published. When you publish your form, the display name will be listed in the **Choose Form** dialog box.

Outlook automatically constructs a message class for the form by preceding the form name with IPM. For instance, if you publish a mail message form that you want to name "MyForm", in the **Display name** field, type: This is my Form. In the **Form name** field, type: MyForm. At the bottom of the dialog box, Outlook will display the message class for your new form as: IPM.Note.MyForm.

When you search in the **Choose Form** dialog box, you will see "This is my Form" displayed in the list. If you select it, the **Display name** field at the bottom of the dialog box will display, "This is my Form" and the **Form name** field will display, "MyForm".

A message class, which is generated automatically by Outlook from the form name, is assigned to the form. When a form with that message class is selected, Outlook loads and displays an instance of that form. Outlook will use the message class, IPM.Note.MyForm, to locate the form with the display name, "This is my Form".



Item types and message classes

The type of an <u>item</u> is defined by its **Type** property. The message class for an item is defined by its **MessageClass** property.

Message class ID	Item type is used to
IPM.Activity	Create journal entries.
IPM.Appointment	Create appointments.
IPM.Contact	Create contacts.
IPM.DistList	Create distribution lists
IPM.Document	Create documents.
IPM.OLE.Class	Create the exception item of a recurrence series.
IPM	If the specified form cannot be found.
IPM.Note	Create e-mail messages.
IPM.Note.IMC.Notification	Create a report from the Internet Mail Connect (the Exchange Server gateway to the Internet).
IPM.Note.Rules.Oof.Template.Microsoft	Show out-of-office templates.
IPM.Post	Post a note in a folder.
IPM.StickyNote	Create a note.
IPM.Recall.Report	Create a message recall report.
IPM.Outlook.Recall	Retrieve sent messages from recipient Inboxes.
IPM.Remote	Represent a Remote Mail message header.
IPM.Note.Rules.ReplyTemplate.Microsof	t Edit rule reply templates.
IPM.Report	Report item status.
IPM.Resend	Resend a failed message.
IPM.Schedule.Meeting.Canceled	Send meeting cancellations.
IPM.Schedule.Meeting.Request	Create meeting requests.
IPM.Schedule.Meeting.Resp.Neg	Create decline meeting responses.

IPM.Schedule.Meeting.Resp.Pos	Create accept meeting responses.
IPM.Schedule.Meeting.Resp.Tent	Create tentative meeting responses.
IPM.Note.Secure	Send encrypted notes to other people.
IPM.Note.Secure.Sign	Send digitally signed notes to other people.
IPM.Task	Create tasks.
IPM.TaskRequest.Accept	Create accept task request responses.
IPM.TaskRequest.Decline	Create decline task request responses.
IPM.TaskRequest	Create task requests.
IPM.TaskRequest.Update	Create updates to requested tasks.

Outlook fields and equivalent properties

Name of field in Microsoft Outlook Field Chooser	Name of equivalent Outlook object model property
% Complete	PercentComplete
Account	Account
Actual Work	ActualWork
Address Selected	N/A
Address Selector	N/A
All Day Event	AllDayEvent
Anniversary	Anniversary
Assigned	DelegationState
Assistant's Name	AssistantName
Assistant's Phone	AssistantTelephoneNumber
Attachment	Attachments
Bcc	BCC
Billing Information	BillingInformation
Birthday	Birthday
Business Address	BusinessAddress
Business Address City	BusinessAddressCity
Business Address Country	BusinessAddressCountry
Business Address PO Box	BusinessAddressPostOfficeBox
Business Address Postal Code	BusinessAddressPostalCode
Business Address State	BusinessAddressState
Business Address Street	BusinessAddressStreet
Business Fax	BusinessFaxNumber
Business Home Page	BusinessHomePage
Business Phone	BusinessTelephoneNumber
Business Phone 2	Business2TelephoneNumber

Callback Car Phone Categories Cc Changed By Children City Color Company Company **Company Main Phone** Complete Computer Network Name Contact Contacts Content Conversation Country Created Customer ID Date Completed Defer until Department **Distribution List Name** Do Not AutoArchive **Download State** Due By Due Date Duration E-mail E-mail 2 E-mail 3 E-mail Selected

CallbackTelephoneNumber CarTelephoneNumber Categories CC N/A Children **HomeAddressCity** Color **Companies CompanyName CompanyMainTelephoneNumber** Complete ComputerNetworkName FormDescription .ContactName Links Body **ConversationTopic** HomeAddressCountry CreationTime **CustomerID** DateCompleted DeferredDeliveryTime Department **DLName** NoAging N/A FlagDueBy **DueDate** Duration **Email1Address** Email2Address Email3Address N/A

E-mail Selector End Entry Type **Expires** File As First Name Flag Status Follow-up Flag From **FTP** Site Full Name Gender Government ID Number Have Replies Sent To Hobbies Home Address Home Address City Home Address Country Home Address PO Box Home Address Postal Code Home Address State Home Address Street Home Fax Home Phone Home Phone 2 Icon Importance In Folder Initials Internet Free Busy Address **ISDN** Job Title Journal

N/A End Type **ExpiryTime** FileAs **FirstName** FlagStatus FlagRequest **SentOnBehalfOfName FTPSite FullName** Gender GovernmentIDNumber ReplyRecipientNames Hobby **HomeAddress** HomeAddressCity HomeAddressCountry **HomeAddressPostOfficeBox** HomeAddressPostalCode HomeAddressState HomeAddressStreet HomeFaxNumber HomeTelephoneNumber Home2TelephoneNumber **FormDescription**.Icon Importance Parent Initials InternetFreeBusyAddress **ISDNNumber** JobTitle Journal

Junk E-Mail Type Language Last Name Last Saved Time Location Mailing Address Mailing Address Indicator Manager's Name Meeting Status Message Message Class Message Flag Middle Name Mileage Mobile Phone Modified Meeting Workspace URL Nickname Notes Office Location **Optional Attendees** Organizational ID Number Organizer Other Address Other Address City Other Address Country Other Address PO Box Other Address Postal Code Other Address State Other Address Street Other Fax Other Phone **Outlook Internal Version**

N/A Language LastName N/A Location MailingAddress N/A **ManagerName MeetingStatus Body MessageClass** FlagStatus **MiddleName** Mileage MobileTelephoneNumber LastModificationTime **MeetingWorkspaceURL NickName Body** OfficeLocation **OptionalAttendees** OrganizationalIDNumber Organizer **OtherAddress OtherAddressCity OtherAddressCountry OtherAddressPostOfficeBox OtherAddressPostalCode OtherAddressState** OtherAddressStreet **OtherFaxNumber** OtherTelephoneNumber **OutlookInternalVersion**

Outlook Version Owner

Pager Personal Home Page Phone *n* Selected Phone *n* Selector PO Box **Primary Phone** Priority Private Profession **Radio Phone** Read Received Recurrence **Recurrence** Pattern **Recurrence Range End Recurrence Range Start** Recurring **Referred By Remind Beforehand** Reminder Reminder Override Default **Reminder Sound Reminder Sound File Reminder** Time **Reminder Topic Remote Status** RequestStatus **Requested By Required Attendess** Resources

OutlookVersion Owner PagerNumber PersonalHomePage N/A N/A **HomeAddressPostOfficeBox** PrimaryTelephoneNumber Importance Sensitivity Profession RadioTelephoneNumber UnRead **ReceivedTime RecurrencePattern**.RecurrenceType N/A **RecurrencePattern**.PatternEndDate **RecurrencePattern**.PatternStartDate IsRecurring **ReferredBy ReminderMinutesBeforeStart ReminderSet** ReminderOverrideDefault ReminderPlaySound ReminderSoundFile

ReminderTime N/A RemoteStatus N/A N/A RequiredAttendees

Resources

Response Requested Retrieval Time Role Schedule+ Priority Send Plain Text Only Sensitivity Sent Show Time As Size Spouse Start Start Date State Status Street Address Subject Suffix Team Task Telex Title То **Total Work Tracking Status** TTY/TDD Phone User Field 1 User Field 2 User Field 3 User Field 4 Web Page **ZIP**/Postal Code

ResponseRequested N/A Role **SchedulePlusPriority** N/A Sensitivity SentOn **BusyStatus** Size Spouse Start **StartDate** HomeAddressState **Status** HomeAddressStreet Subject Suffix TeamTask **TelexNumber** Title То **TotalWork TrackingStatus** TTYTDDTelephoneNumber User1 User2 User3 **User4 WebPage** HomeAddressPostalCode

Understanding the forms cache

The forms cache is a folder located in the user's C:\Windows\...\Application Data\Forms folder that serves as a storage location for forms. The forms cache improves the load time of a form because commonly used forms are loaded from the hard disk rather than downloaded from the server. When a form is activated for the first time, the form definition file is copied from its forms library to the Forms folder. The forms cache keeps a temporary copy of the form definition in a subfolder whose name roughly matches the name of the form.

The form table, Frmcache.dat, also located in the Forms folder, is used to locate a form and to prevent multiple instances of the same form from being loaded in the cache. When a form is activated, Microsoft Outlook checks to see if a form with the same message class is already in the cache. If not, it copies the form definition to the cache. In addition, if a change has been made to a form, Outlook copies the new form definition to the cache.

Outlook looks for forms in the following order. When a match is found, Outlook opens the form and does not search further.

- 1. Standard Outlook forms, such as Note, Post, and Contact, in the Application Forms Library
- 2. Forms already cached in the form cache
- 3. Forms published in the folder that's currently selected
- 4. Forms in the Personal Forms Library
- 5. Forms in the Organizational Forms Library

Because Outlook caches forms, you should try to avoid having more than one form with the same name or publishing the same form to more than one forms library. Forms used in a folder-based solution should be published only in the folder. If you're developing a solution based on mail message forms, you can temporarily publish the forms in your Personal Forms Library. Once the form is finalized, you should publish the form to the Organizational Forms Library on the Microsoft Exchange Server and delete the form from your Personal Forms Library after making a backup of the form. If, for some reason, you need to publish a form in more than one location, you should be sure to keep all forms libraries up-to-date with the current version of the form.

Note It is not possible to disable the forms cache. If you set the size of the form cache to 0, Outlook will not be able to open any custom forms.



Create custom forms by using Visual Basic Scripting Edition

Microsoft VBScript is a subset of Visual Basic for Applications. You can use VBScript to create procedures that control Microsoft Outlook folders, objects, <u>items</u>, and properties. VBScript in Outlook requires a special object syntax that has some differences from referencing objects in Visual Basic for Applications.

Learn about the Outlook object model.

You can choose the Outlook item on which to base your custom form. Learn about <u>choosing an item</u>.

You can extend Outlook forms by using custom controls from the <u>Control</u> <u>Toolbox</u>. Outlook forms can use most of the properties and methods that come with the controls. Since controls cannot store values, to store the value you need to <u>bind</u> the control to an Outlook field.

Learn about **binding**.

The Outlook object browser displays the classes, properties, methods, events, and <u>constants</u> available from the Outlook object library. The object browser lets you view and use objects in the Microsoft Script Editor and obtain information about the syntax for using the object.

Learn about viewing and using the object browser.

The Microsoft Script Debugger provides you with a comprehensive debugging environment for testing and correcting errors in the VBScript code for your forms.

Learn about the Script Debugger.



Troubleshooting forms

I can't find the form I created.

- The form may be visible only for message responses. To trigger the response form, create a form, such as a new message form, and then send it to yourself. Open the response form, click the (**Properties**) page, and then clear the **Use form only for responses** check box.
- The form may not have been saved, or it may not have been saved to a forms library.
- Based on the item type used, switching to a table view may make it easier to see the form.

My data is not being saved with the form.

You probably used the Control Toolbox to add controls to a form. When you add controls from the Control Toolbox, if the control will be used to store data, you should bind the control to a field. Data is stored in fields, not controls. If a control is not bound to a field, the data will be lost when the item is saved or sent.

Note If you drag a field from the **Field Chooser**, the control that is created will automatically be bound to the field.

The form is always saved with the item.

If Microsoft Visual Basic Scripting Edition (VBScript) in a form changes the form itself, Microsoft Outlook may save the form definition with the item, creating a one-off item.

I can't change the default e-mail message form.

You cannot set a custom form as the default form for the **Inbox**. All incoming email uses the default mail form or a form specified by the sender.

However, you can change the default mail message form that Microsoft Outlook

will use in all folders. For more information, see Help.

New items aren't using my custom form.

You need to set the default form for the folder to your custom form.

Old items aren't using my custom form.

You need to change the **Message Class** field of all the older items in the folder.

Received e-mail messages or post forms don't appear customized when read.

E-mail messages and post forms can be viewed in either Compose or Read mode. You can disable this capability so that there is only one mode for the form, or you can modify the Read and Compose pages separately. These options are on the **Form** menu when in design time.

An incorrect version of a form is being used.

You probably published this form to more than one forms library. Make sure that the form is published in only one place.

I can't find the message or notes property for an item.

The **Message** or **Notes** field on a form is referred to as the **Body** property in the Microsoft Outlook object model.

I cannot open a password-protected form.

Remember that the password is case-sensitive.

If a form has been password-protected and the password is no longer known, Microsoft cannot help you circumvent the password protection.

Error: A field on this form requires a value.

Check whether controls on the form have field validation enabled in the

Properties dialog box of the control.

If the error still appears, you must delete any unused custom fields that exist in the item, regardless of whether they appear on the form.

Use an Outlook Form in Microsoft Exchange.

Microsoft Exchange Client does not recognize Microsoft Outlook forms. A Microsoft Exchange Client user who opens an Outlook form sees only the standard message form properties; that is, all custom fields are not visible. However, Outlook can open forms designed with the Microsoft Exchange Forms Designer.

The form commands are missing from the **Tools** menu, so I can't display the design environment.

If you are using Microsoft Word as your e-mail editor, the form's design environment isn't available. Turn off Word as your e-mail editor, and then reopen the form.

- 1. In the main Microsoft Outlook window, on the **Tools** menu, click **Options**, and then click the **Mail Format** tab.
- 2. Clear the **Use Microsoft Word to edit e-mail messages** check box.

My solution doesn't run on other computers.

Use the following troubleshooting tips to help troubleshoot problems if your forms or programming solution works on some computers, but not others.

Microsoft Visual Basic Scripting Edition (VBScript) version If your solution involves forms that use VBScript, you may need to make sure that all computers are using the same version of VBScript that is being used on your development computer. VBScript is a shared component. Installing other software, such as a newer version of Microsoft Internet Explorer, may result in newer versions of VBScript being installed. For the latest information about VBScript versions, go to http://www.microsoft.com/scripting.

Controls If your solution uses any nonstandard controls, check to see if the controls are properly installed on all of the user's computers. If you are using any

control other than one of the Forms 2.0 controls that are installed by Microsoft Office, you should provide your users with a Setup program to ensure that all of your controls are installed correctly.

Permissions or user rights Make sure that any user experiencing problems has proper permissions or rights to use any public folders or other resources that your solution uses.

Mail forms If your solution sends forms from one user's computer to another, did you select the **Send form definition with item** check box on the form's **Properties** page when in design mode? If not, you or the users must properly install the form so that it is available for users. Also, make sure that the computers are configured with the same type of mail accounts. Sometimes the type of message format (Microsoft Rich Text, HTML, or plain text) can play a role, as well as whether or not Microsoft Word is the e-mail editor. When the message is received, is the type of message format the same as when it was sent? Is it possible the format was converted during the send process?

VBScript code in the form does not run.

For security reasons, Outlook does not run Microsoft Visual Basic Scripting Edition (VBScript) in a form if the form is a one-off form. Ideally, you should avoid one-off forms and publish forms to a form library instead. If you are using Microsoft Exchange, you can implement custom security settings in the Microsoft Outlook Security Settings folder to re-enable VBScript in one-off forms.

The macro virus warning keeps appearing.

There is Microsoft Visual Basic Scripting Edition (VBScript) code in the item. To prevent this message from appearing, the VBScript code should be located in a published form. Published forms are considered safe, and therefore the macro virus warning is not displayed if the code exists in the published form.

If you published the form but still receive the warning message, this usually means one of two things: the **Send form definition with item** check box is selected on the form's **Properties** page; or the item has become a one-off form and the form definition, including VBScript code, has become part of the item itself.

The **Click** event of a control doesn't fire.

The **Click** event doesn't fire for controls bound to a field. Because the controls are bound to a field, when the value of the field changes you can use the **PropertyChange** or **CustomPropertyChange** event.

Overview of Appointment Forms

The default pages of an appointment form are not customizable. Because these pages provide much of the Microsoft Outlook appointment scheduling functionality, it is recommended that you do not hide these pages, but add additional functionality to a new page on the form.

Note Meeting Request forms are not designed to be customized. Instead, customize an appointment form, and the custom form will be used when the appointment is converted to a meeting.

Overview of Contact Forms

The **General** page of a contact form can be modified, but the other default pages cannot be customized. You can hide these pages and create new pages on the form if you choose. Learn about <u>using contact item selector fields</u>.

Note On the **General** page, you cannot modify the address, phone and e-mail "selector" lists that contain the names of standard Microsoft Outlook fields. Also, the **FileAs** list offers two choices if only a contact's first and last name have been entered, or five choices if a company name has also been entered. This list of choices cannot be modified.



Using contact item selector fields

You can customize a contact form using specially bound controls that display dialog boxes that let users change fields in the item.

Most of these controls are available directly from the **Field Chooser**. The following lists these controls.

Address Selector Address Selected E-mail Selector E-mail Selected Phone *x* Selector Phone *x* Selected

To use these fields on a contact form page, open the form in <u>design mode</u> and then use the **Field Chooser** to drag the field onto the page. This will automatically create the appropriate controls and labels on the page.

Overview of Distribution List Forms

The default pages of distribution list form are not customizable. Because these pages provide much of the Microsoft Outlook distribution list functionality, it's recommended that you do not hide these pages, but add additional functionality to a new page on the form.

Overview of Journal Forms

The default page of the journal form cannot be modified, but you can hide this page and create new pages on the form if you choose.

Note The list of **Entry Type** entries cannot be customized.

Overview of Mail Forms

All pages of an e-mail form are customizable.

Like post forms, mail forms have separate **Compose** and **Read** pages. When a user opens a new e-mail message, the form is displayed in Compose mode and pages are displayed so that the user can fill out the form to be sent. When the recipient opens the item, the form is displayed in Read mode, and a different set of pages and controls can be used to display the data in the fields. When designing an e-mail message form, you can choose to disable the separate **Read** and **Compose** pages so that the form is identical at all times. If you use separate pages, you can often copy and paste the controls from the one set of pages to the other.

Learn about making separate Compose and Read pages.

If you are customizing a mail form for use in an organization that is using Microsoft Exchange Server, the form should be published in the Organizational Forms Library. If you do this, then you can clear the **Send form definition with item** check box on the form's **Properties** page so that only the field data is transmitted when the item is sent. This will greatly reduce network traffic and also make updating the form much easier in the future because each client will be using the shared form that is published on the server.

When designing a mail form, it is important to consider which type of message format the form should be in. Choices are Microsoft Rich Text, HTML and plain text. You can also specify that Microsoft Word must be used as the e-mail editor if you enable the option on the **Properties** tab of the form. Careful consideration should be given to what versions of Microsoft Outlook or Word potential recipients will be using. To programmatically set HTML in a mail form, see **HTMLBody** Property. To programmatically access the Word object model to manipulate a Word e-mail message, see **WordEditor** Property.

Note Some of the properties and objects related to mail formats may behave differently depending on whether Outlook 98 or Outlook 2000 was running in Corporate Workgroup mode or Internet Mail Only mode. Outlook 2002 and later do not distinguish between the Corporate Workgroup mode and Internet Mail

Only mode.



Make separate compose and read versions for a form

You can create and edit separate compose and read versions for each <u>page</u> of a form. The compose version is what the sender sees— for example, what you see when you create a message. The read version is what the recipient sees— for example, what you see when you read your mail.

With the form in design mode:

1. To create or edit the compose page, click **Edit Compose Page** on the **Form** menu.

To create or edit the read page, click **Edit Read Page** on the **Form** menu.

2. Customize pages on the form.

<u>**How**</u>?

Note You can quickly switch between the two versions by clicking the **Edit Compose Page** and **Edit Read Page** buttons on the **Form Design** toolbar. If these buttons are not available, you set compose and read versions to be the same. To have separate compose and read versions, click **Separate Read Layout** on the **Form** menu.

Overview of Post Forms

Like mail forms, all post form pages are customizable, and post forms also have separate **Compose** and **Read** pages that you can modify.

Learn about making separate Compose and Read pages.

A folder containing posts is usually grouped by the **Conversation** field to keep related topics together. The **Conversation** field, also known as the **ConversationTopic** property in the Microsoft Outlook object model, is set based on the **Subject** field of a new post in the folder. All replies to this item will inherit the **Conversation** field from the parent item. However, if the user changes the **Subject** field, the **Conversation** field will be set to the new subject, creating a new conversation.

Overview of Task Forms

The default pages of a task form are not customizable. However, you can hide these pages and create new pages on the form if you choose.

When tasks are used in a personal folder, you can assign tasks to other people. However, when task items are placed in a Microsoft Exchange public folder, they cannot be assigned. Instead, the Owner field can be modified to reflect who currently owns the task. It may be advantageous to create a custom view so that the tasks are grouped by their owner.

Set properties for a form

With the form in design mode:

- 1. Click the **(Properties)** page.
- 2. To protect your form with a password, select the **Protect form design** check box, and then type a password in the **Password** and the **Confirm** boxes. Click **OK**.
- 3. To be able to send a copy of the form in e-mail, select the **Send form definition with item** check box.
- 4. To have the form show only for replies to messages, select the **Use form only for responses** check box.
- 5. Select other options that you want.



Set the attributes of a form

The attributes of a form are set in design mode on the **Properties** and **Actions** pages. On the **Properties** page, the information such as Description, Category, and Form Number helps users find and identify your form in the **New Form** dialog box. If you do not want users to modify your form, you can protect a form by assigning a password to it.

One very important attribute of your form is whether to send the form definition with the form. The form definition includes all the fields and the code you add to the form. If you do not plan to publish your form to a forms library, then you must select the **Send form definition with item** check box (**Properties** page). The form definition includes all the new data you add to the form when you send the form to another user. When you only plan to use your form one time and do not want to publish your form, this type of form is commonly referred to as a one-off form.

If you plan to publish your form to a forms library to which other users have access, such as an **Organizational Forms Library** or a <u>public folder</u> library, you do not need to send the form definition; it is stored in the library. The form definition can add considerable size to your form.

If you want to change how users reply to your form, click the **Actions** page. The **Actions** page lists the default Reply forms that are available. You can add your own custom Reply forms as well. For example, forms based on a new mail message have built-in Reply, Reply to All, Forward, and Reply to Folder forms. When users receive your form, the form will have buttons and menu commands so users can respond to the form. You can disable some or all of these default forms and set attributes for how these Reply forms appear.



Set default properties for a form

With your form in design mode, click the tab for the **Properties** page.

- **Category** You can specify a category for your form that will help to organize the forms in the **New Form** dialog box when you are selecting a form.
- **Sub-Category** You can further refine the category by specifying a sub-category.
- Always use Microsoft Word as the e-mail editor Lets you specify that Microsoft Word will be the editor for the message portion (or control) of your form. This will add all the formatting options that are available with Word, such as spell checking and thesaurus. In order for these options to be available, the recipients of your form must have Word installed.

Note This feature has not been changed from earlier versions of Microsoft Office. Selecting this option uses an older architecture for using Word as the e-mail editor and does not provide exactly the same user experience as enabling Word as the e-mail editor by clicking **Options** on the **Tools** menu.

- **Template** You can specify the Word template that is used to format the text in the message control of the form.
- **Contact** When you click **Contact**, you have access to the <u>Address Book</u>. This allows you to select the names of people who are responsible for maintaining, upgrading, or providing information about this form. The contact information that you provide will show in the **Forms Manager** dialog box and the form **Properties** page.
- **Description** You can type a description of your form. This could include instructions for the use of the form as well as a full description of the form's purpose. This information will be displayed in the **About** dialog box on the **Help** menu of the form as well as in the **Properties** dialog box for the form.
- Version Allows you to set a version number for this form. This is a free-

form text field and does not affect Microsoft Outlook behavior in any way.

- **Form Number** Allows you to set a form number. This is a free-form text field and does not affect Outlook behavior in any way.
- **Change Large Icon** When you click this button, a **File Open** dialog box allows you to select a different large icon for your form. Large icons appear in the form **Properties** dialog box.
- **Change Small Icon** When you click this button, a **File Open** dialog box allows you to select a different small icon for your form. Small icons appear in the Outlook folder to represent an <u>item</u> of the type created with the form.
- **Protect Form Design** When you select this box, you gain access to the **Password** dialog box. You can type a password of your choice in the **Password** field, and then retype it in the **Confirm** field. If you make a mistake when retyping, a message box will inform you and you will have the opportunity to type the confirmation again. Setting a password for your form means that others are prevented from changing any of the attributes of your form after it is published. If you clear the check box, you no longer have password protection.
- **Set Password** When you click this button, the **Password** dialog box will allow you to set a password for the protection of your form. This has the affect of selecting the **Protect Form Design** check box.

Note The password can be accessed using the Outlook object model so it is not fully secure. Setting a password on a custom form only serves as a deterrent so that users cannot easily access the form in design mode.

• Send form definition with item Specifies that the form definition is included when you send the form. This causes the form to be much larger than if this option is not selected. Selecting this check box creates a self-contained form that will allow your recipients to view the form even if they don't have access to the same forms library as the sender.

If you don't specify that the form definition should be sent with the item, the recipient will receive a regular form instead of the one that you have created. Setting this option allows recipients to open the item in the form on their computer, even though the form has not been installed on their Outlook system. This is especially useful for one-off forms that have been created for one-time use.

Outlook does not run VBScript code if the form definition is included with the item. In most cases, it is better to publish a form rather than to include the form definition with the item. If you do send the form with the item, you can re-enable the VBScript code if you use custom security settings in the "Outlook 11 Security Settings" folder on the Microsoft Exchange public folder. In that case, if you send a form with this box checked, the recipients will see a **Warning** dialog box. They have the option of disabling the macros since the form is not published. Harmful macros could delete or copy their files, or send mail from their mailbox to another user.

If network or file transfer time is an issue, and you cannot publish the form for some reason, an alternative to sending the form definition is to save the form and send it as an attachment to another form. Recipients can take the attached form and publish it in their own forms library.

• Use form only for responses Hides a form when it is published to a forms library. This option is useful in situations when you have created a form that is intended only for replies. In another form, you can specify that your reply form will be used instead of the default reply form.

To use your form only for responses, select the **Use form only for responses** check box, and then <u>publish your form</u>. Open a second form in design mode. On the **Actions** page of the second form, you can specify your published form in the Reply or Reply to All action. To use your form as the default reply form, double-click the **Reply** action in the second form. You can select the name of your published reply form in the **Form name**: field of the **Form Action Properties** dialog box. This will cause your reply form to be used instead of the default reply form.



Add Help to a form

A Help file cannot be associated with a form, so you cannot use the **HelpContextID** property to create Help for a Microsoft Outlook form. Instead, you can use the following steps to add a page of explanatory text to your form.

- 1. Click the page to which you want to add text.
- 2. On the **Form** menu, click **Rename Page**.
- 3. In the **Page name** box, type **Help**, and then click **OK**.
- 4. Drag a **TextBox** control from the **Control Toolbox** to the form.
- 5. Right-click the **TextBox** control, and then click **Properties** on the <u>shortcut</u> <u>menu</u>.
- 6. On the **Display** page, select **Multi-line** check box.
- 7. Type the explanatory text that you want in the **TextBox** control.
- 8. Right-click the **TextBox** control, and then click **Properties** on the shortcut menu.
- 9. On the **Display** page, select the **Read only** check box.

Note If the **Compose** page is different from the **Read** page, you may need to create separate Help pages.

Change the icons associated with your form

- 1. In design mode, click the **Properties** page.
- 2. Click Change Large Icon or Change Small Icon.
- 3. Select the icon you want to use.

Note Sometimes you may find that even if you specify that a custom form should use a specific icon, when you view an item in a folder, a standard icon is used instead. This may happen in the following scenarios:

- You published the form in a location that is not accessible to everyone. For example, when they receive a custom e-mail message form, they do not have access to the custom form. Therefore, Microsoft Outlook displays the standard icon.
- The item has become a one-off form. In this case, the message class of the item changes to the default message class for that particular type of item and the icon reverts back to the default icon for that type of item.
- You replied to or forwarded an item in a folder and the new item does not use the custom icon. In this case, the form action that started the new item did not specify a custom message class. Therefore, the new item is a standard form that will not have a custom icon.
- You replied to or forwarded an item in a folder. In this case, Outlook replaces the custom icon with the standard icon so that the reply or forward indicator arrows can be displayed.

The following resolutions correspond to the causes listed above:

- Publish the form to a location that is accessible to everyone using the form, typically the Organizational Forms Library.
- Reset the message class of the item.
- Disable the standard **Reply**, **ReplyAll**, and **Forward** actions and create custom **Reply**, **ReplyAll**, or **Forward** actions that start your custom form instead.
- This is a design limitation of Outlook. The forward or reply icons will always be used instead of your custom icon.

Show or hide fields when printing and saving a form

- 1. In design mode, right-click the field that you want, and then click **Properties** on the shortcut menu.
- 2. Click the **Validation** page.
- 3. Select or clear the **Include this field for Printing and Save As** check box.

Hide or show form pages

While in design mode:

- 1. Click the page that you want to hide or show.
- 2. On the **Form** menu, click **Display This Page** to clear or select the check mark.

Notes

- Parentheses around a page name indicate the page is hidden. You cannot show the **(Properties)** or **(Actions)** pages.
- If you drag a field or control to a hidden page, the page automatically shows.
- To change what appears on the default or first page of an item, hide the first page, and then create a new version on one of the custom pages.
- At least one page must be visible in a form .

Require a value for a field or control

- 1. In design mode, right-click the field or control that you want, and then click **Properties** on the shortcut menu.
- 2. Click the **Validation** page.
- 3. Select the **A value is required for this field** check box.



About standard fields in Microsoft Outlook

Microsoft Outlook provides standard fields for each standard <u>item</u>. Some fields are available in individual items, and you can add and remove all fields from a table or card <u>view type</u>.

The value for fields with the Yes/No data type is saved as -1 or 0. The value for fields with the Duration data type is saved as minutes.

What do you want more information about?

<u>Standard fields in a **Calendar** item</u>

Field	Edit in view	Data type and meaning
All Day Event	Yes	Yes/No. If set to Yes, the Duration field is set to 24 hours (1440 minutes).
Attachment	No	Yes/No.
Billing Information	Yes	Text.
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Contacts	No	Text. Names of contacts linked to this item. Multiple names are separated by commas.
Conversation	ı No	Text. Value of the Subject field in the original message in a conversation.
Created	No	Date/Time. Date and time the Calendar item is created.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the Calendar item.
Duration	No	Duration. 24 hours (1440 minutes) if the All Day Event field is set to Yes. Otherwise, the difference between the values of the End and Start fields. Saved as minutes.
End	Yes	Date/Time. End date and time of a Calendar item.
Importance	Yes	The following settings apply. 0 Low importance 1 Normal importance 2 High importance
Icon	Yes	Internal data type.
In Folder	No	Text. Name of the folder that contains the Calendar item.
Location	Yes	Text. Location of a meeting or appointment.
		The following settings apply. 0 None

Meeting Status	No	 Meeting organizer Tentatively accepted Accepted Declined Not yet accepted
Message Class	No	Specifies the <u>message class</u> for the type of item.
Mileage	Yes	Text.
Modified	No	Date/Time. Last time the Calendar item was modified.
NetMeeting AutoStart	Yes	Yes/No. Specifies whether an online meeting starts immediately when the reminder appears.
NetMeeting Office Document Path	Yes	Text. Specifies the path of the Microsoft Office document used for online-meeting collaboration.
NetMeeting Organizer E- mail	Yes	Text. E-mail address of the online-meeting organizer.
NetMeeting Server	Yes	Text. Name of the NetMeeting server for the online meeting.
NetMeeting Type	No	The following settings apply. 0 NetMeeting 1 NetShow
NetMeeting URL	Yes	Text. The URL of the online meeting.
Notes	No	Text. Value of the text box of the appointment.
Online Meeting	Yes	Yes/No.
Online Meeting Type	Yes	The following settings apply. 0 NetMeeting 1 NetShow
Optional Attendees	No	Text. Names of optional attendees for a meeting or appointment. Multiple names are separated by semicolons.
Organizer Outlook	No	Text. Name of the organizer of a meeting or appointment.

Internal Version	No	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the Calendar item is created in.
Read	No	Yes/No. Specifies whether the Calendar item has been marked as read.
Recurrence	No	The following settings apply. 0 None 1 Daily 2 Weekly 3 Monthly 4 Yearly
Recurrence Pattern	No	Text. Combination of the values of the Recurrence , Start , and End fields.
Recurrence Range End	No	Date/Time. Last date and time of a recurring Calendar item.
Recurrence Range Start	No	Date/Time. First date and time of a recurring Calendar item.
Recurring	No	Yes/No. Specifies whether the Calendar item recurs.
Remind Beforehand	Yes	Number. Minutes before the reminder runs prior to a meeting or appointment.
Reminder	Yes	Yes/No. If the start time for the meeting or appointment has already passed, the Reminder field cannot be set.
Reminder Override Default	Yes	Yes/No. If set to Yes, the Remind Beforehand , Reminder Sound , and Reminder Sound File fields are used to control the reminder for the item. If set to No, the options in the Reminders Options dialog box (on the Tools menu, click Options , click the Other tab, click Advanced Options , and then click Reminder Options) are used.
Reminder Sound	Yes	Yes/No. Specifies whether to play the sound file as a reminder.
Reminder Sound File	Yes	Text. Path of the sound file to be played as a reminder.
Required Attendees	No	Text. Names of required attendees for a meeting or appointment. Multiple names are separated by semicolons.
Resources	No	Text. Names of resources for a meeting or appointment.

Response Requested	Yes	Multiple names are separated by semicolons. Yes/No. In a meeting request, specifies whether the recipient has been asked to respond.
Sensitivity	No	The following settings apply. 0 Normal 1 Personal 2 Private 3 Confidential
Show Time As	Yes	The following settings apply. 0 Free 1 Tentative 2 Busy 3 Out of Office
Size	No	Number. Number of bytes used by the Calendar item.
Start	Yes	Date/Time. Start time of a Calendar item.
Subject	Yes	Text.

Standard fields in a contact

Field	Edit in view	Data type and meaning
Account	Yes	Text.
Address Selected	Yes	Displays the address text that was entered into the Address field based on the value of the Address Selector field.
Address Selector	Yes	The following settings apply. Home Business Other
Anniversary	Yes	Date/Time. When the Anniversary field has a value, a Calendar item is attached to the contact, and the Attachment field is set to Yes.
Assistant's Name Assistant's	Yes	Text.

Phone	Yes	Text.
Attachment	No	Yes/No. Set to Yes when the Anniversary or Birthday field is a non-empty field.
Billing Information	Yes	Text.
Birthday	Yes	Date/Time. When the Birthday field has a value, a Calendar item is attached to the contact, and the Attachment field is set to Yes.
Business Address	Yes (Card view only)	Text.
Business Address City	Yes	Text.
Business Address Country	Yes	Text.
Business Address PO Box	Yes	Text.
Business Address Postal Code	Yes	Text.
Business Address State	Yes	Text.
Business Address Street	Yes (Card view only)	Text.
Business Fax	Yes	Text.
Business Home Page	Yes	Text.
Business Phone		Text.
Business Phone 2	Yes	Text.

Callback	Yes	Text.
Car Phone	Yes	Text.
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Children	Yes	Text.
City	Yes	Text.
Company	Yes	Text.
Company Main Phone	Yes	Text.
Computer Network Name	Yes	Text.
Contacts	No	Text. Names of contacts linked to this item. Multiple names are separated by commas.
Country/Region	Yes	Text.
Created	No	Date/Time. The date and time the contact is created.
Customer ID	Yes	Text.
Department	Yes	Text.
E-mail	No	Text.
E-mail 2	No	Text.
E-mail 3	No	Text.
E-mail Selected	Yes	Displays the e-mail address that was entered based on the value of the E-mail Selector field.
		The following settings apply.
E-mail Selector	Yes	0 E-mail 1 E-mail 2 2 E-mail 3
E-mail Display As	Yes	Alternate text that represents the e-mail address stored in the E-mail field. This text displays on the To line when addressing a message or appointment.
E-mail2 Display As	Yes	Alternate text that represents the e-mail address stored in the E-mail 2 field. This text displays on the To line when addressing a message or appointment.
E-mail3		Alternate text that represents the e-mail address stored

Display As	Yes	in the E-mail 3 field. This text displays on the To line when addressing a message or appointment.
File As	No	Text. Value of the Full Name field, unless modified by the user.
First Name	Yes	Text.
Follow Up Flag	Yes	Text.
FTP Site	Yes	Text. FTP site name for the contact.
Full Name	Yes	Text. Value of the Title , First , Middle , Last , and Suffix fields of the item, separated by spaces. Any changes made to the Full Name field are reflected in its component fields.
Gender	Yes	The following settings apply. 0 Unspecified 1 Female 2 Male
Government ID Number	Yes	Text.
Hobbies	Yes	Text.
Home Address	Yes (Card view only)	Text.
Home Address City	Yes	Text.
Home Address Country	Yes	Text.
Home Address PO Box	Yes	Text.
Home Address Postal Code	Yes	Text.
Home Address State	Yes	Text.
Home Address Street	Yes (Card view only)	Text.

Home Fax	Yes	Text.
Home Phone	Yes	Text.
Home Phone 2	Yes	Text.
Icon	Yes	Internal Data Type.
IM Address	Yes	Instant Messaging address. (Instant Messaging is a feature of the Microsoft MSN Messenger Service and Microsoft Exchange Instant Messaging Service.)
In Folder	Yes	Text. Name of the folder that contains the contact.
Initials	Yes	Text.
Internet Free- Busy Address	Yes	Text. 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.
ISDN	Yes	Text. Phone number for ISDN connection.
Job Title	Yes	Text.
Journal	Yes	Yes/No. Specifies whether activities are automatically recorded in the Journal for the contact.
Language	Yes	Text.
Last Name	Yes	Text.
Location	Yes	Text.
Mailing Address	Yes (Card view only)	Text.
Mailing Address Indicator	Yes	Yes/No. Indicates whether the address specified by the Address Selector field is the mailing address.
Manager's Name	Yes	Text.
Message Class		Text. Specifies the <u>message class</u> for the type of item.
Middle Name	Yes	Text.
Mileage	Yes	Text.
Mobile Phone	Yes	Text.
Modified	No	Date/Time. Last date and time the contact was modified.
Nickname	Yes	Text.
Notes	No	Text. Value of the text box of the contact.

Office Location	Yes	Text.
Organizational ID Number	Yes	Text.
Other Address	Yes (Card view only)	Text.
Other Address City	Yes	Text.
Other Address Country	Yes	Text.
Other Address PO Box	Yes	Text.
Other Address Postal Code	Yes	Text.
Other Address State	Yes	Text.
Other Address Street	Yes (Card view only)	Text.
Other Fax	Yes	Text.
Other Phone	Yes	Text.
Outlook Internal Version	No	For administrator use only
Outlook Version	No	Text. Version of Outlook that the contact is created in.
Pager	Yes	Text.
Personal Home Page	Yes	Text.
Phone 1 Selected (through Phone 8 Selected)		Displays the phone number that was selected in the corresponding Phone Selector field.
Phone 1		The following settings apply:

Selector (through Phone 8 Selector)		Business, Home, Business Fax, Mobile, Radio, Car, Other, and ISDN
PO Box	Yes	Text.
Primary Phone	Yes	Text.
Private	Yes	Yes/No. Indicates whether a specific contact is visible to others who have access to the Contacts folder.
Profession	Yes	Text.
Radio Phone	Yes	Text.
Read	No	Yes/No. Specifies whether the contact has been marked as read.
Referred By	Yes	Text.
Reminder	Yes	Yes/No.
Reminder Time	Yes	Date/Time. Date and time that a reminder is run for a contact.
Reminder Topic	Yes	Text. Caption displayed with reminder flag.
Sensitivity	No	The following settings apply. 0 Normal 1 Personal 2 Private 3 Confidential
Size	No	Number. Number of bytes used by the contact.
Send Plain Text Only		
Spouse	Yes	Text.
State	Yes	Text.
Street Address	Yes (Card view only)	Text.
Subject	Yes	Text. Value of the Full Name field. If the Full Name field is empty, the value of the File As field is used.
Suffix	Yes	Text. Name suffix, such as Jr. or Ph.D.
Telex	Yes	Text.

Title	Yes	Text. Name title, such as Mr., Ms., or Mrs.
TTY/TDD Phone	Yes	Text. Phone number for TTY/TDD connection.
User Field 1 – 4 Yes		Text. User-defined fields provided for compatibility with other programs.
Web Page	Yes	Text.
ZIP/Postal Code	Yes	Text.

Standard fields in a distribution list

Edit in view	Data type and meaning	
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Distribution List Name	Yes	Text. The name of the distribution list.
Icon	Yes	Internal data type.
In Folder	No	Text. Name of the folder that contains the distribution list item.
Message Class	No	Specifies the message class for the type of item.
Modified	No	Date/Time. Last time the distribution list item was modified.
Outlook Internal Version	No	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the distribution list item is created in.
Sensitivity	No	The following settings apply. 0 Normal 1 Personal 2 Private 3 Confidential

Size No Number. Number of bytes used by the distribution list item.

Standard fields in a **Journal** entry

Edit in view	Data type and meaning	
Attachment	No	Yes/No.
Billing Information	Yes	Text.
Categories	Yes	Text. User-defined field used to group and find related items. Multiple categories are separated by commas.
Company	Yes	Text.
Contact	No	Text. Name of the contact the Journal entry is recorded for.
Contacts	No	Text. Names of contacts linked to this item. Multiple names are separated by commas.
Created	No	Date/Time. Date and time the Journal entry is created.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the Journal entry.
Duration	No	Duration. Saved as minutes.
End	Yes	Date/Time. End date and time set for the Journal entry.
Entry Type	No	Text. Type of entry made for the Journal entry.
Icon	Yes	Internal data type.
In Folder	No	Text. Name of the folder that contains the Journal entry.
Message Class	No	Text. Specifies the <u>message class</u> for the type of item.
Mileage	Yes	Text.
Modified	No	Date/Time. Last time the Journal entry was modified.

Notes	No	Text. First 255 characters in the body of the Journal entry.
Outlook Internal	No	For administrator use only.
Version	INU	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the Journal entry is created in.
Read	No	Yes/No. Specifies whether the Journal entry has been marked as read.
		The following settings apply.
Sensitivity	No	0 Normal 1 Personal
ochonevity	110	2 Private
		3 Confidential
Size	No	Number. Number of bytes used by the Journal entry.
Start	Yes	Date/Time. Start time for the Journal entry.
Subject	Yes	Text.

Standard fields in an e-mail message

Field	Edit in view	Data type and meaning
Attachment	No	Yes/No.
BCC	No	Text. Names in the Bcc box of a message.
Billing Information	Yes	Text.
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
CC	No	Text. Names in the Cc box of a message.
Contacts	No	Text. Names of contacts linked to this item. Multiple names are separated by commas.
Conversatior	n No	Text. Value of the Subject field in the original message in a conversation.
Created	No	Date/Time. Date and time the message is created.

Defer Until	Yes	Date/Time. Date and time a message is to be delivered. The server delays delivery of the message.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the message.
Due By	Yes	Date/Time. Date and time the action associated with a Message Flag is set to be completed by. When a value is entered for Due By, the Flag Status field is set to 2.
Expires	Yes	Date/Time. Date and time a message expires.
Flag Status	Yes	The following settings apply. 0 Normal 1 Completed 2 Flagged
Follow Up Flag	Yes	Text.
From	No	Text. Names in the From box in a message.
Have Replies Sent To	No	Text. Names in the Have replies sent to box in a message.
Header status	No	Indicates the download state of the message.
status		C
Icon	Yes	Internal data type.
	Yes Yes	Internal data type. The following settings apply. 0 Low importance 1 Normal importance
Icon		Internal data type. The following settings apply. 0 Low importance
Icon Importance	Yes	Internal data type. The following settings apply. 0 Low importance 1 Normal importance 2 High importance
Icon Importance In Folder Junk E-mail	Yes No	Internal data type. The following settings apply. 0 Low importance 1 Normal importance 2 High importance Text. Name of the folder that contains the message.
Icon Importance In Folder Junk E-mail type	Yes No Yes	Internal data type. The following settings apply. 0 Low importance 1 Normal importance 2 High importance Text. Name of the folder that contains the message. Internal data type.
Icon Importance In Folder Junk E-mail type Message Message	Yes No Yes No	Internal data type. The following settings apply. 0 Low importance 1 Normal importance 2 High importance Text. Name of the folder that contains the message. Internal data type. Text. Value of the text box in a message.
Icon Importance In Folder Junk E-mail type Message Message Class Message	Yes No Yes No No	Internal data type. The following settings apply. 0 Low importance 1 Normal importance 2 High importance Text. Name of the folder that contains the message. Internal data type. Text. Value of the text box in a message. Specifies the message class for the type of item. Text. Action associated with a Message Flag. When a value

Outlook Internal Version	No	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the message is created in.
Read	No	True/False. Specifies whether the message has been marked as read.
Receipt Requested	No	Yes/No. Indicates whether message was sent with a read or delivery receipt requested.
Received	No	Date/Time. Date and time the message is received.
Relevance	Yes	Number. User-defined significance.
Remote Status	No	Specifies the status of Remote Mail header. The following settings apply. 0 None 1 Marked 2 Marked for download 3 Marked for copy
Retrieval Time	No	Duration. Specifies the time it takes to download the message with Remote Mail. Saved as minutes.
Sensitivity	No	The following settings apply. 0 Normal 1 Personal 2 Private 3 Confidential
Sent	No	Date/Time. Date and time the message is sent.
Size	No	Number. Number of bytes used by the message.
Subject	Yes	Text.
То	No	Text. Names in the To box in a message.
Tracking Status	No	Tracking status of a message. The following settings apply. 1 Delivered 5 Read 6 Not Read

Standard fields in a note

Data type

Edit in view	and meaning	
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Color	Yes	The following settings apply. 0 Blue 1 Green 2 Pink 3 Yellow 4 White
Content	No	Text. Value of the text box of a note.
Created	No	Date/Time. Date and time the note is created.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the note.
Icon	Yes	Internal data type.
In Folder	No	Text. Name of the folder that contains the note.
Message Class	No	Text. Specifies the <u>message class</u> for the type of item.
Modified	No	Date/Time. Last date and time the note was modified.
Outlook Internal Version	No	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the note is created in.
Read	No	Yes/No. Specifies whether the note has been marked as read.
Size	No	Number. Number of bytes used by the note.
Subject	No	Text.

<u>Standard fields in a task</u>

Edit in view	Data type and meaning	
% Complete	Yes	Percent.

Actual Work	Yes	Duration. Time spent on a task. Saved as minutes.
Assigned	No	The following settings apply. 0 Not assigned 1 Assigned by me 2 Assigned to me
Attachment	No	Yes/No.
Billing Information	Yes	Text.
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Company	Yes	Text.
Complete	Yes	Yes/No. Specifies whether the task is marked as completed.
Contacts	No	Text. Names of contacts linked to this item. Multiple names are separated by commas.
Conversation	No	Text. Value of the Subject field of the original task.
Created	No	Date/Time. Date and time the task is created.
Date Completed	No for	Date/Time. Date and time the task is completed. If the Date Completed field has a value, the Complete field is set to Yes, and the % Complete field is set to 100.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the task.
Due Date	Yes	Date/Time.
Icon	Yes	Internal data type.
In Folder	No	Text. Name of the folder that contains the task.
Message Class	No	Text. Specifies the <u>message class</u> of the type of item.
Mileage	Yes	Text.
Modified	No	Date/Time. Last date and time the task was modified.
Notes	Yes	Text. Value of the text box of the task.
Outlook Internal	No	For administrator use only.

Version		
Outlook Version	No	Text. Version of Outlook that the task is created in.
Owner	Yes, if task is in a public folder; otherwise, No.	Text. Owner of the task.
Priority	Yes	The following settings apply. 0 Low priority 1 Normal priority 2 High priority
Read	No	Yes/No. Specifies whether the task has been marked as read.
Recurring	No	Yes/No. Specifies whether the task recurs.
Reminder	No	Yes/No. Specifies whether a reminder has been set for the task.
Reminder Override Default	Yes	Yes/No.
Reminder Sound	Yes	Yes/No. Specifies whether to play the sound file as a reminder.
Reminder Sound File	Yes	Text. Path of the sound file to be played as a reminder.
Reminder Time	Yes	Date/Time. Date and time that a reminder is run for a task.
Request Status	No	The following settings apply. 0 No setting 1 Not responded 2 Accepted 3 Declined
Requested By	No	Text. In a task request, the person's name who assigned the task.
Role	Yes	Text.
Schedule+ Priority	Yes	Text.
		The following settings apply.

Sensitivity	No	0 Normal 1 Personal 2 Private 3 Confidential
Size	No	Number. Number of bytes used by the task.
Start Date	Yes	Date/Time. Start date and time for the task.
Status	Yes	The following settings apply. 0 Not Started 1 In Progress 2 Completed 3 Waiting on someone else 4 Deferred
Subject	Yes	Text.
Team Task	Yes	Yes/No.
То	No	The owner of an assigned task.
Total Work	Yes	Duration. Time the task is expected to take. Saved as minutes.

Standard fields in a post

Field	Edit in view	Data type and meaning
Attachment	No	Yes/No.
Billing Information	Yes	Text.
Categories	Yes	Text. Field used to group and find related items. Multiple categories are separated by commas.
Conversation	No	Text. Value of the Subject field in the original item in a conversation.
Created	No	Date/Time. Date and time the message is created.
Defer Until	Yes	Date/Time. Date and time a message is to be delivered. The server delays delivery of the message.
Do Not AutoArchive	Yes	Yes/No. Specifies whether to archive the message.
Expires	Yes	Date/Time. Date and time a message expires.

From Header status	No No	Text. Names in the From box in a message. Indicates the download state of the message.
Icon	Yes	Internal data type.
Importance	Yes	The following settings apply. 0 Low importance 1 Normal importance 2 High importance
In Folder	No	Text. Name of the folder that contains the message.
Message	No	Text. Value of the text box in a message.
Message Class	No	Specifies the message class for the type of item.
Mileage	Yes	Text.
Modified	No	Date/Time. Last date and time the message was modified.
Outlook Internal Version	No	For administrator use only.
Outlook Version	No	Text. Version of Outlook that the message is created in.
Read	No	True/False. Specifies whether the message has been marked as read.
Received	No	Date/Time. Date and time the message is received.
Remote Status	No	Specifies the status of Remote Mail header. The following settings apply. 0 None 1 Marked 2 Marked for download 3 Marked for copy
Retrieval Time	No	Duration. Specifies the time it takes to download the message with Remote Mail. Saved as minutes.
Sensitivity	No	The following settings apply. 0 Normal 1 Personal 2 Private 3 Confidential
Sent	No	Date/Time. Date and time the message is sent.
Size	No	Number. Number of bytes used by the message.

Subject Yes Text.

Change the value of a field

To change the value of an Outlook field, use the property name of the associated standard field. For example, to change the value of the Subject field, use the following code.

Item.Subject = "New Subject"

To change the value of a custom field, use the following code to refer to a custom field.

Item.UserProperties.Find("MyProperty").Value = "New Value"



Use fields with controls

When you drag a field from the **Field Chooser**, the field automatically <u>binds</u> to the appropriate control. Unless you have a special requirement to use a standard control from the **Control Toolbox**, you should use the **Field Chooser** to provide access to fields on your forms.

When you put a control from the **Control Toolbox** on a form, you must bind the control to a form if you want to save a value to or from a control. In most cases, you bind **TextBox**, **CheckBox**, **ListBox**, **ComboBox**, and **OptionButton** controls to fields. Other controls, such as **Label** and **Image** controls, that contain static information with which the user does not interact are generally not bound to a field.

To bind a control to a field, right-click the control, and then click **Properties** on the shortcut menu. Click the **Value** tab. Click **Choose Field**, and then click a field or click **New** to create a custom field. Outlook fields are based on MAPI properties. In this way, the values of fields are stored with the item when you save or send the item. The controls from the **Control Toolbox** are only the visual containers for a field on a form . You can set the appearance of the control using its properties, but you cannot save a value. Controls only exist when the specific form appears that contains the controls. Fields can be used on any form. If you change a field value in one place, this value changes everywhere the field is used.

For example, to change the value of a custom field called Fax, you use the following code:

```
Item.UserProperties.Find("Fax").Value = "555-1234"
```

Note that since this is a field, you do not need to specify the page or the control the field is bound to. In the following code example, a control called txtFax is made invisible. When you work with a control, you must specify the page and the control name.

```
Item.GetInspector.ModifiedFormPages("General").("txtFax").Visible =
False
```

You can bind a control to a field at run time by using the internal property named **ItemProperty**. The following example binds a **TextBox** to a field named Business Address.

Item.GetInspector.ModifiedFormPages("General").Textbox1.ItemProperty
= "Business Address"

Note If you create a control by dragging a plain text field to a form, you cannot bind the control to a field of a different type. For example, you cannot drag a Subject field to a form and then bind it to a field containing an Email type (such as the To field).

Active controls and selected controls

In design mode, all controls have an active state and a selected state. When a control is active, it means you are working with the contents of the control; when a control is selected, it means you are working with the control itself.

Most controls are automatically selected when you put them on the form . In design mode, sizing handles appear around the control border when the control is selected. If you deselect the control, you can select it again by clicking the control.

Clicking a control that is selected puts the control in the active state. In this state, you can directly edit the control <u>Caption</u> or <u>Value</u> property, depending on the control.

In both the selected and the active state, you can use DEL, CTRL+X, and CTRL+C for the **Delete, Cut**, and **Copy** commands, respectively. In the selected state, these commands are available on the shortcut menu and affect the control itself. In the active state, these commands affect whatever text is selected inside the control; if no text is selected, these commands have no effect. These commands are not available on the shortcut menu for active controls.

About custom fields and data types

You can use custom fields in several ways in Outlook.

- You can create new data-entry fields for a view or form so users can add their own custom information. For example, you can create a Date/Time field named "Last Talked To" in a contact folder and then add the field to a Phone List view.
- You can create new views by combining the standard fields in a single column. For example, you can create a column that combines the State and Country fields in an address list to save space.
- You can create a formula field to show information in a new way. For example, you can create a field that displays "Large" when a message is more than 10,000 bytes and "Small" when a message is under the same limit.

Any field that you create is stored in the folder in which you create it. To use a field in more than one folder, you must recreate the field in each folder.

You can create and view custom fields in table views and card views. You can create custom fields with the following data types in Outlook.

Data type	Use to represent		
Combination	Combinations of values of fields and text in a column (table) or row (card). You can specify whether to show each field or show only the first non-empty field. You can also combine text with fields without the use of quotation marks.		
Currency	Numeric data as currency or mathematical calculations that involve money.		
Date/Time	Date and time data.		
Duration	Numeric data. You can enter a duration as minutes, hours, or days. Values are saved as minutes.		
Formula	Calculations based on standard and custom fields. You can use any appropriate functions and operators to complete the formula.		

Integer	Non-decimal numeric data.	
Keywords	When filled in, this user-defined field is used to group and find related items similar to the way the Categories field is used in Outlook. Text with multiple values to be separated by commas. Each value can be grouped individually in a view.	
Number	Numeric data or mathematical calculations except those that involve money. (For money, use Currency data type.)	
Percent	Numeric data as a percentage.	
Text	Text or combinations of text and numbers, such as addresses. Can be up to 255 characters long.	
Yes/No	Data that contains only one of two values, such as Yes/No, True/False, On/Off.	

Each of the data types except Combination, Formula, and Keywords has a series of standard formats you can use to show the values of the fields.

Difference between a field and a column

Usually, a column in a table contains the value of a single field, such as the Received field. However, you can show several fields in a column by using a Formula or Combination field. Fields can also be shown as rows in a card view or as controls on a form. A column is just one way to show the contents of a field.



Create a simple field for a folder

- 1. Select the folder for which you want to create the field.
- 2. On the **View** menu, point to **Arrange By**, then select **Current View**, and click **Customize Current View**.
- 3. On the **View Summary** dialog box, click **Fields**.
- 4. Click New Field.
- 5. In the **Name** box, type a name for the field.
- 6. In the **Type** box, click the data type you want for the field. Do not select **Combination** or **Formula**.
- 7. In the **Format** box, select the format you want for the field.
- 8. Click **OK**.
- To position the field between the other fields in the <u>view</u>, click Move Up or Move Down.
- 10. To remove the field from the view, click **Remove**.

Examples of standard formats for data types

You can format fields that you create with different formats based on the data type. The following table provides a list of some of the formats that Microsoft Outlook includes.

For a complete list of standard formats available for a field in a view, click **Arrange By** on the **View** menu, then click **Current View**, and then click **Format Columns**. Select the field you want to format in the **Available fields** box, and then select a format in the **Format** list. If you don't find the format you want, you can create a custom format with a Formula data type and the **Format** function.

Data type	Standard format examples	
Currency	\$12,345.60 or (\$12,345.60)	
Currency	\$12,346 or (\$12,346)	
	Monday, March 03, 1997 8:00 AM	
	3/03/97 8:00 AM	
Date/Time	March 3, 1997	
	Mon 3/3 8:00 AM	
	8:00 AM	
	12h	
Duration	12 hours	
Duration	12h (Work Time) (see the Note below)	
	12 hours (Work Time) (see the Note below)	
Integer	1,234	
Keywords Text		
	All Digits: 1,234.567 or -1,234.567	
Number	Truncated: 1,235 or -1,235	
Number	Scientific: 1235E+03 or -1235E+03	
	Computer: 64 K or 128MB or 1 GB	
	65.4321%	
Percent	65.43% or -65.43%	

	65% or -65%	
Text	Text	
	Yes or No	
Yes/No	On or Off	
	True or False	

Note Values entered in a field of a Duration data type with a regular format assume a 24-hour day.



Create a combination field for a folder

- 1. Select the folder for which you want to create the combination field.
- 2. On the **View** menu, point to **Arrange By**, then select **Current View**, and click **Customize Current View**.
- 3. On the **View Summary** dialog box, click **Fields**.
- 4. Click **New Field**.
- 5. In the **Name** box, type a name for the field.
- 6. In the **Type** box, click **Combination**.
- 7. Click **Edit**.
- 8. To append each value for the fields that you want to use together, click **Joining fields and any text fragments to each other**.

To show only the first field that is not blank, click **Showing only the first non-empty field, ignoring subsequent ones**.

- 9. Click **Field**, point to the field set that you want, and then click the field that you want to appear.
- 10. Repeat step 9 to add each field that you want to appear.
- 11. Click **OK** twice.
- 12. To position the field between the other fields in the <u>view</u>, click **Move Up** or **Move Down**.

To remove the field from the view, click **Remove**.

Notes

- Combination fields appear with the default format of the data type used. To display a data type with a custom format, you must create a formula field and use the **Format** function.
- You cannot sort, group, or filter the contents of a combination field.



Create a formula field for a folder

- 1. Select the folder for which you want to create the formula field.
- 2. On the **View** menu, point to **Arrange By**, then select **Current View**, and click **Customize Current View**.
- 3. In the **View Summary** dialog box, click **Fields**.
- 4. Click **New Field**.
- 5. In the **Name** box, type a name for the field.
- 6. In the **Type** box, click **Formula**.
- 7. Click **Edit**.
- 8. To insert a function in the formula, click **Function**, point to the function set that you want, and then click the formula that you want.
- 9. To insert a field in the formula, select the argument that you want to replace, click **Field**, point to the field set that you want, and then click the field that you want.
- 10. Click **OK** twice.
- 11. To position the field between the other fields in the <u>view</u>, click **Move Up** or **Move Down**.
- 12. To remove the field from the view, click **Remove**.

Notes

- Formula fields are updated with any change to a view. For example, if you change the width of a column, the formula fields are updated.
- You cannot sort, group, or filter the contents of a formula field.



Examples of Formula and Combination fields

What you want to show	Custom field	Result in custom field
Number of days since the item was received. (Formula field)	DateValue (Now())- DateValue ([Received]) & " Day(s)"	6 Day(s)
Description of a meeting or appointment in Calendar. (Formula field)	"This meeting occurs " & [Recurrence Pattern] & " in " & [Location]	The meeting occurs every day from 12:00 P.M. to 1:30 P.M. in room 1231
Amount to be charged for a phone call recorded in the Journal at .75 a minute. (Formula field)	IIF ([Entry Type] = "Phone call" , Format ([Duration] * .75, "Currency"), "None")	\$1.50
Description of a <u>Message Flag</u> . (Formula field)	IIF ([Flag Status] = "2", [Message Flag] & " " & [Due By],"")	Follow up 3/5/97 8:00:00 A.M.
The first phone number recorded for a contact, in order of appearance in the formula. (Combination field)	[Business Phone] [Business Phone 2] [Home Phone] [Home Phone 2] [Car Phone]	(555) 555-1234
A description of a field combined with the field itself. (Combination field)	Task Due: [Due Date]	Task Due: 3/5/97 8:00:00 A.M.



Troubleshoot custom fields

The custom field I created is not visible.

- Custom fields are stored in the folder in which you create them. Open the folder in which you created the custom field to see if it is visible. You must recreate a custom field for each folder in which you want to use it.
- Custom fields are stored in the field set called **User-defined** fields in the **Field Chooser** and in the **Show Fields** dialog box. To see all the custom fields in a folder, open the folder. On the **View** menu, select **Arrange By**, then select **Current View**, and click **Customize Current View**. On the **View Summary** dialog box, click **Fields**. In the **Select available fields from** box, click **User-defined fields in folder**.

When I try to sort, group, or filter a formula field or combination field, I receive an error message.

You cannot sort, group, or filter a formula field or combination field in Microsoft Outlook.

Set properties for controls

There are two types of properties for a control: Outlook properties and advanced properties. Outlook properties are the basic properties needed to design a control. Advanced properties offer additional functionality.

Outlook properties are set in the **Properties** dialog box, and must be set individually. Advanced properties are set in the **Advanced Properties** dialog box and can be set for multiple controls at once.

- To set properties for a control, right-click the control, and then click **Properties** on the shortcut menu. Select the options that you want.
- To set advanced properties for controls, select the controls that you want, right-click one of the controls, and then click **Advanced Properties** on the shortcut menu. Select the options that you want.

Change the name property of a control

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, type a name in the **Name** box.

Note Each control should have a unique name.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to set the **Name** property of a **CheckBox** on a page named "Test" to "Selection."

Item.GetInspector.ModifiedFormPages("Test").Checkbox1.Name =
"Selection"



Validate user input to a field

You can require that any field, or any control that is <u>bound</u> to a field, have a value. For example, you can set the exact text, range of numbers, or any other criteria for the field. You can also show a custom message (based on a validation formula) that tells users how to fill in the field if their entries do not match the value you want. For example, you can limit the number of characters that can be entered in a <u>TextBox</u>.

What do you want to do?

Require a value for a field or control

Create a validation formula and message

Examples of validation formulas

Use To evaluate

- <50 Numbers less than fifty
- >50 Numbers greater than fifty
- =50 Equal to fifty
- ? The number of characters



About the Control Toolbox

The <u>**Control Toolbox**</u> identifies the controls you can add to a <u>**Frame**</u> or page of a form.

You can customize the **Control Toolbox** in many ways including the following:

- Add pages to the **Control Toolbox**.
- Move controls from one page in the **Control Toolbox** to another.
- Rename **Control Toolbox** pages.
- Add other controls, including ActiveX controls, to the **Control Toolbox**.
- Copy modified controls from the form to the **Control Toolbox**.

For example, **OK** and **Cancel** buttons are special types of **CommandButtons**. If you copy **OK** and **Cancel** templates to the **Control Toolbox**, you can quickly add them to other forms.

Note When you add a control to a form by using the **Control Toolbox**, the control is not <u>bound</u> to a field.

About ToolTips and control tips

Both ToolTips and control tips are short, descriptive phrases that appear when the user holds the mouse pointer briefly over a control or another part of the user interface. The difference between ToolTips and control tips is that Microsoft Outlook provides ToolTips to developers at design time, and developers provide control tips to end-users at run time.

You can customize ToolTips for controls and for the **Control Toolbox**.

A control tip is set with the **<u>ControlTipText</u>** property.

Add a page to the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🕅 .
- 2. Right-click the label of any page in the **Control Toolbox**, and then click **New Page** on the shortcut menu.

Add custom controls to the Control Toolbox

You can add a modified control (based on modifications made to the advanced properties) to the <u>Control Toolbox</u>. You can also add other custom controls to the **Control Toolbox**, such as ActiveX controls that are not part of Outlook.

You can use a variety of custom controls in Outlook forms, but there are some limitations. Although Outlook supports most ActiveX properties and methods, it does not support custom event handling. The <u>Click</u> event is the only event for which you can write code. To access the methods of an ActiveX control, use your VBA Object Browser to browse ActiveX control methods.

What do you want to do?

Add a modified control to the Control Toolbox

Add other custom controls to the Control Toolbox

Change the name and control tip of a page in the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🕅 .
- 2. Right-click the label of the page that you want in the **Control Toolbox**, and then click **Rename** on the shortcut menu.
- 3. In the **Caption** box, type a new name for the page.
- 4. In the **Control Tip Text** box, type new text for the control tip.

Change the order of pages in the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🕅 .
- 2. Right-click the label of any page in the **Control Toolbox**, and then click **Move** on the shortcut menu.
- 3. In the **Page Order** box, click the name of a page that you want to move.
- 4. Click **Move Up** or **Move Down**.

Customize an icon for a control in the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🕅 .
- 2. Right-click the icon for the control you want in the **Control Toolbox**, and then click **Customize** *control name* on the shortcut menu.
- 3. To change the ToolTip, enter text in the **Tool Tip Text** box.
- 4. To edit the icon, click **Edit Picture**. In the **Color** box, click the color that you want to use, and then click the pixel in the **Picture** box where you want to apply the color. Click **OK**.
- 5. To assign a new bitmap, click **Load Picture**. In the **File name** box, type a name for the file that contains the bitmap you want to use as the icon. The bitmap cannot be larger than an icon.

Delete a control from the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🖄 .
- 2. In the **Control Toolbox**, right-click the control that you want to delete, and then click **Delete** *control name* on the shortcut menu.

Delete a page in the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** \bigotimes .
- 2. Right-click the label of the page that you want to delete in the **Control Toolbox**, and then click **Delete Page** on the shortcut menu.

Note All controls on a page are deleted when the page is deleted.

Import or export a Control Toolbox page

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🕅 .
- 2. To import a page, right-click the label of any page in the **Control Toolbox**, and then click **Import Page** on the shortcut menu.

To export a page, right-click the label of the page, and then click **Export Page** on the shortcut menu.

- 3. In the **File name** box, enter the name of the page that you want to import or enter the name of the file to which you want to export a page.
- **Note** Exporting a page does not remove it from the **Control Toolbox**.

Move a control to another page in the Control Toolbox

- 1. If the **Control Toolbox** is not visible, click **Control Toolbox** 🖄 .
- 2. Drag a control on any page to the label of another page. Hold the mouse pointer over the label until the page appears on top, and then drag on to the page.

Note If the page to which you want to move a control is not visible, increase the width of the **Control Toolbox** to show all the pages, and then drag the control to the appropriate page.

Show or hide the Control Toolbox

• In design mode, click **Control Toolbox** 🕅 .

About groups of controls

A group is two or more controls on a form that you can work with as a single unit. You can include any control on the form in a group. Once controls belong to a group, you can work with the entire group, or you can select a single control.

There are many ways to work with groups and the controls in groups. After you select a group, you can do any of the following.

- Size all controls in the group at the same time.
- Separate controls in the group so each is independent of the others.
- Show the group shortcut menu for quick access to commands that affect the group.
- Select a single control within the group without separating controls in the group. You can then change property settings of the selected control without affecting any other control in the group.

Add a control to a form

Use any of the following methods to add a control from the <u>Control Toolbox</u> to your form . You can also use these methods to insert a control in a <u>Frame</u>, <u>TabStrip</u>, or <u>MultiPage</u> on the form.

- Click a control in the **Control Toolbox** and then click in the form. The control appears in its default size. You can then drag the border of the control to change its size.
- Drag a control from the **Control Toolbox** to the form. The control appears in its default size.
- Double-click the control in the **Control Toolbox**, and then click in the form once for each control you want to create. For example, to create four command buttons, double-click the **CommandButton** control in the **Control Toolbox**, and then click four times in the form.



Align controls

1. Select the controls that you want to align to part of the <u>dominant control</u>.

How?

2. On the **Layout** menu, point to **Align**, and then click one of the following. Click То Align controls to the left edge of the dominant control. Left Align the center of controls to the vertical center of the dominant Center control. Align controls to the right edge of the dominant control. Right Тор Align controls to the top edge of the dominant control. Align the center of controls to the horizontal center of the Middle dominant control. **Bottom** Align controls to the bottom edge of the dominant control. Align the upper-left corner of controls with their nearest grid То point. Note that this option is not based on the position of the Grid dominant control.

Align text in a control

This procedure is only available for the <u>**ComboBox**</u>, <u>**Label**</u>, and <u>**TextBox**</u> controls.

- 1. Right-click the control you want, and then click **Advanced Properties** on the shortcut menu.
- 2. To set the **TextAlign** property, click the property and enter a value in the **Apply** box.

Set the TextAlign property to	То
Left	Align the text with the left edge of the control
Right	Align the text with the right edge of the control
Center	Center the text relative to the length of the control

Allow multiple lines of text in a control

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, select the **Multi-line** check box.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to set the multi-line property of a **TextBox** on a page called "Test."

Item.GetInspector.ModifiedFormPages("Test").Textbox1.MultiLine
= True



Assign a control tip, caption, and accelerator key to a control

- 1. Right-click the control that you want, and then click **Advanced Properties** on the shortcut menu.
- 2. To set the property, click the property and enter a value in the **Apply** box.

Set the property	To assign	
ControlTipText	The string you want to use as the <u>control tip</u>	
<u>Caption</u>	The string you want to use as the <u>caption</u>	
<u>Accelerator</u>	A single character as the value for ALT+Key entry. If you use a character from the caption of the control, the character is underlined	

Note You can also set the value of these properties with code. For example, to set the caption of the **Label1** control on the **Message** page, use:

Item.GetInspector.ModifiedFormPages("Message").Label1.Caption = "New



Create a set of OptionButton controls

By default, all **OptionButton** controls in a <u>container</u> are part of a single option group. This means that selecting one of the buttons automatically sets all other option buttons on the page to **False**.

If you want more than one option group on the page, there are two ways to create additional groups:

- Use the **GroupName** property to identify related buttons. This method reduces the number of controls required on the form , which can reduce the hard disk space required and improve the performance of the form. If you want to create an option group in a **TabStrip** (which is not a container), you must use the **GroupName** property.
- Put related buttons in a **<u>Page</u>**, **<u>MultiPage</u>**, or <u>**Frame**</u> on the form.

What do you want to do?

<u>Create a set of OptionButtons using the GroupName property</u>

Add a control to a form

Enable or disable a control on a page

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, select or clear the **Enabled** check box.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to set the **Enabled** property of a **CheckBox** on a page named "Test."

Item.GetInspector.ModifiedFormPages("Test").Checkbox1.Enabled =
False

Give a control a 3-D appearance

- 1. In design mode, right-click the control you want, and then click **Advanced Properties** on the shortcut menu.
- 2. Click **SpecialEffect**, and then click the option you want in the box above.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to set the **SpecialEffect** property of a **CheckBox** on a page named "Test."

Item.GetInspector.ModifiedFormPpages("Test").Checkbox1.SpecialEf
= 2

Hide or show a control

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, select or clear the **Visible** check box.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to set the **Visible** property of a **CheckBox** on a page named "Test."

Item.GetInspector.ModifiedFormPages("Test").Checkbox1.Visible =
False

Make a control read-only

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, select the **Read only** check box.

The following code example uses the **ModifiedFormPages** property of the current **Inspector** object to make TextBox1 read-only.

Item.GetInspector.ModifiedFormPages("P.2").TextBox1.ReadOnly =
True

Move a control one unit at a time

- 1. In design mode, right-click the control you want, and then click **Properties** on the shortcut menu.
- 2. On the **Display** page, type a number in the **Top** and **Left** boxes.

Show the shortcut menu for controls and groups of controls

To show the shortcut menu for a	Right-click
page	the caption of the appropriate page.
MultiPage	anywhere in the control, but not on the caption of any page in the control.
Group of controls (groups created with the Group command)	anywhere inside the rectangle that surrounds the group.
tab	the caption of the appropriate tab.
TabStrip	anywhere in the control, but not on the caption of any tab in the control.
Any other control or container	the control or container.

Note The commands on a shortcut menu vary based on the object you select. For example, if you select multiple controls that aren't in a group, the shortcut menu includes the **Group** command; the shortcut menu for a text box does not include this command.



Ways to protect sensitive information

Many applications use data that is for authorized users only. The following list suggests ways you can protect sensitive information in a <u>form</u>. Note however, that the fields associated with a form can still be exposed in a view.

• Write code that makes a control (and its data) invisible to unauthorized users. To identify a particular user in code, use the following code.

Application.GetNameSpace("MAPI").CurrentUser

The **<u>Visible</u>** property makes a control visible or invisible.

- Write code that sets the control foreground and background to the same color when unauthorized users run the application. This hides the information from unauthorized users. The **ForeColor** and **BackColor** properties determine the <u>foreground color</u> and the <u>background color</u>.
- Disable the control when unauthorized users run the application. The **Enabled** property determines when a control is disabled.
- Require a password to alter a form.
- Require a password for access to the application or a specific control. You can use <u>placeholders</u> as the user types each character. The <u>PasswordChar</u> property defines placeholder characters.
- If the form only appears in a public folder, you can set permissions for the folder to determine who can open the form.

Note Using passwords or any other techniques listed above can improve the security of your application, but these methods do not guarantee the prevention of unauthorized access to your data.



Center controls in a form

1. Select the controls or groups of controls that you want to center. The <u>dominant control</u> does not affect this procedure.

How?

2. On the **Layout** menu, point to **Center in Form**, and then click **Horizontally** or **Vertically**.

Right-align a control on a form

- 1. Drag a <u>Label</u> control from the <u>Control Toolbox</u> to the left of the spot where you would like to align a control.
- 2. Remove any text from the **<u>Caption</u>** property.
- 3. Right-click the label, and then click **Properties** on the shortcut menu.
- 4. On the **Display** page, select the **Resize with form** check box.
- 5. Create the control you want to align.
- 6. Place the control to the right of the **Label** control.



Space controls

- 1. Select the controls for which you want to adjust spacing.
- 2. On the **Layout** menu, point to **Horizontal Spacing** or **Vertical Spacing**, and then click one of the following.

Click	То	
Make Equal	Make all horizontal and vertical spaces between controls the same size. The amount of horizontal and vertical space will vary based on the area available to show controls and the combined width of all controls.	
Increase	Increase the space between controls by one grid block.	
Decrease Decrease the space between controls by one grid block.		
Remove	Remove the space between controls. The controls do not overlap, but are immediately adjacent to each other.	



Size controls

1. Select the controls that you want to size to the <u>dominant control</u>.

How?

2. On the **Layout** menu, point to **Make Same Size**, and then click one of the following.

ClickToWidthMake all selected controls the same width as the dominant control.HeightMake all selected controls the same height as the dominant control.

Both Make all selected controls the same height and width as the dominant control.

Note To size a control to fit its contents (both its picture and any text assigned to the **Caption** or **Text** property), select the control, and then click **Size to Fit** on the **Layout** menu.

Automatically resize controls

You can set controls to resize horizontally or vertically when a user changes the size of the form. The size of the control changes in proportion to the size of the form. For example, when a user changes the size of a message, the body resizes automatically.

Resize a control horizontally with the form

Resize a control vertically with the form

Resize a control horizontally and vertically with the form



Resize a control horizontally and vertically with the form

This procedure sets an invisible property. After you complete the procedure, you should remove the code from the Script Editor.

- 1. In <u>design mode</u>, select the control, and then size it relative to the form. For example, if you want the control to always be same as the height of the form, make the bottom edge of the control equal to the bottom edge of the form.
- 2. On the **Form** menu, click **View Code**.
- 3. Add the following code to the Script Editor. Replace "TextBox1" with the name of the control you want to resize, and replace "P. 2" with the name of your page. Although this example uses the **Write** event, you can use any event to run the code.

```
Sub Item_Write
Item.GetInspector.ModifiedFormPages("P.2").TextBox1.LayoutFlags
= 65
```

End Sub

- 4. On the **Form** menu of the form, click **Run This Form**.
- 5. On the **File** menu of the form, click **Save**.

Note To reset the **LayoutFlags** property, right-click the control, and then click **Properties** on the shortcut menu. On the **Display** page, if the **Resize with form** check box is selected, clear it; if the **Resize with form** check box is cleared, select it and then clear it.



Resize a control vertically with the form

This procedure sets an invisible property. After you complete the procedure, you can remove the code from the Script Editor.

- 1. In <u>design mode</u>, select the control, and then size it relative to the form. For example, if you want the control to always be the height of the form, make the bottom edge equal to the bottom edge of the form.
- 2. On the **Form** menu, click **View Code**.
- 3. Add the following code to the Script Editor. Replace "TextBox1" with the name of the control you want to resize, and replace "P. 2" with the name of your page. Although this example uses the **Write** event, you can use any event to run the code.

```
Sub Item_Write
Item.GetInspector.ModifiedFormPages("P.2").TextBox1.LayoutFlags
= 68
```

End Sub

- 4. On the **Form** menu of the form, click **Run This Form**.
- 5. On the **File** menu of the form in <u>run mode</u>, click **Save**.

Note To reset the **LayoutFlags** property, right-click the control, and then click **Properties** on the shortcut menu. On the **Display** page, if the **Resize with form** check box is selected, clear it; if the **Resize with form** check box is cleared, select it and then clear it.

Select multiple controls and the dominant control

You can select more than one control in three ways. In addition, when you select multiple controls, one of the controls becomes a reference for the rest and is called the dominant control.

Selection method	What is selected	Dominant control
	All controls in an invisible rectangle around the selected controls.	First control you select
CTRL+CLICK	Individual controls, one at a time.	Last control you select.
Select Objects pointer in Control Toolbox	All controls that fall within or touch a rectangle you draw.	Control nearest the mouse pointer when you begin drawing the rectangle.

Tips

- Occasionally the CTRL+CLICK method may select additional controls that are near to or adjacent to the selected controls. For more accuracy, use the **Select Objects** pointer method.
- If you CTRL+CLICK twice on a selected control, that control becomes the dominant control.



Use z-order to layer controls

- 1. Select the controls you want to reposition in the <u>z-order</u>.
- 2. On the **Layout** menu, point to **Order**, and then click one of the following.

ClickTo move controlBring to FrontTo the top of the z-order.Send to BackTo the bottom of the z-order.Bring ForwardUp one position in the z-order.Send BackwardDown one position in the z-order.

Notes

- You can't Undo or Redo layering commands, such as **Send to Back** or **Bring to Front**.
- If the form includes any ListBox, Frame, or MultiPage controls, these controls automatically move as close as possible to the top of the stack. For example, applying Send Backward to a ListBox, Frame, or MultiPage control moves the control below other ListBox, Frame, or MultiPage controls, but does not move the control below any other type of control in the stack. Similarly, applying Bring Forward to a control closer to the top of the stack, but does not move the control above any ListBox, Frame, or MultiPage control in the stack.

Visually, this means that if a **ListBox**, **Frame**, or **MultiPage** control and any other Microsoft Forms controls are in the same location on a form, the **ListBox**, **Frame**, or **MultiPage** control always appears on top of the other control(s). If a **ListBox**, **Frame**, or **MultiPage** control is in the same place as another **ListBox**, **Frame**, or **MultiPage** control, the z-order of the controls determines which control appears on top of the other.

Show or hide the grid

• In design mode, click **Show Grid** on the **Layout** menu.



Set the grid size

- 1. In design mode, click **Set Grid Size** on the **Form** menu.
- 2. In the **Height** and **Width** boxes, type the dimensions that you want.

Tip

If you use the **Arrange** command (**Layout** menu) to position command buttons in your application, use small grid settings so that you can position the command buttons closer to the edge of the form.

Turn on or off sizing controls to the grid

• In design mode, click **Snap to Grid** on the **Layout** menu or click the Snap to Grid icon on the toolbar.



Create a group of controls

1. Select each control that you want to include in the group. The <u>dominant</u> <u>control</u> does not affect this procedure.

How?

2. On the **Layout** menu, click **Group**.

Select and edit a control within a group

- 1. Select the group of controls.
- 2. Select a single control within the group. The sizing handles around the group become lighter, and dark sizing handles appear on the selected control.
- 3. Make the changes that you want to the selected control properties. Any change you make affects only the selected control.
- 4. When you're finished, click anywhere in the group, but don't click on the selected control. The group is still selected.
- 5. Select another control in the group or go on to another task.

Separate controls in a group

- 1. Select the group of controls.
- 2. On the **Layout** menu, click **Ungroup**.



Assign a control tip, caption, and accelerator key to a page or tab

This procedure sets properties on a **<u>Page</u>** or **<u>Tab</u>** in a **<u>MultiPage</u>** or **<u>TabStrip</u>** control only.

1. In design mode, select a page or tab in a **MultiPage** or **TabStrip** control.

How?

Be sure to select an individual page or tab, not the corresponding **MultiPage** or **TabStrip**. When a page or tab is selected, a rectangle appears around its caption.

- 2. Right-click the caption of the selected page or tab, and then click **Rename** on the shortcut menu.
- 3. In the **Control Tip Text** box, type the string you want to use as the <u>control</u> <u>tip</u>.
- 4. In the **Caption** box, type the string you want to use as the caption.
- 5. In the **Accelerator Key** box, type a single character. Use a character from the caption of the control. Note that the selected character is underlined in the control caption.

Tip

To assign a control tip for a **MultiPage** or **TabStrip**, use the <u>ControlTipText</u> property. If you assign a control tip to a **MultiPage** or a **TabStrip**, control tips for the individual page or tab objects within the **MultiPage** do not appear.

Change the order of pages in a MultiPage or TabStrip

This procedure sets properties on a <u>Page</u> or <u>Tab</u> *only* in a <u>MultiPage</u> or <u>TabStrip</u> control.

1. In design mode, select any page in the **MultiPage** or **TabStrip** control.

How?

- 2. Right-click the caption of the page, and then click **Move** on the shortcut menu.
- 3. In the **Page Order** box, click the page you want to move.
- 4. Click **Move Up** or **Move Down**.

Note You can also change the page order by changing the <u>Index</u> property. The index of the first page is 0; the index of the second page is 1, and so on.

Set tab order of controls on a page

- 1. In design mode, right-click on a page in a form, but not on a control, and then click **Tab Order** on the shortcut menu.
- 2. Select the name of a control you want to reposition in the tab order.
- 3. Click **Move Up** or **Move Down**.

Note You can also set the tab order by changing the **TabIndex** property. The tab index of the first control in the tab order is 0; the tab index of the second control is 1, and so on.

Should I use a MultiPage or a TabStrip?

If you use a single layout for data, use a **TabStrip**, and then map each set of data to its own tab. If you need several layouts for data, use a **MultiPage**, and then assign each layout to its own page.

Unlike a page of a **MultiPage**, the client region of a **TabStrip** is not a separate form but a portion of the form that contains the **TabStrip**. The border of a **TabStrip** defines a region of the form that you can associate with the tabs. When you place a control in the client region of a **TabStrip**, you are adding a control to the form that contains the **TabStrip**.

Create a list box

- 1. In design mode, drag the <u>ListBox</u> control from the <u>Control Toolbox</u> to the form .
- 2. Right-click the list box, and then click **Advanced Properties** on the shortcut menu.
- 3. To create a standard list box, set the **ListStyle** property to **Plain**. To set the property, click the property and enter a value in the **Apply** box.

To create a list box with option buttons or check boxes, set the **ListStyle** property to **Option**.

4. Fill the list at design time or run time.

How?

Note When the **ListStyle** property is set to **Option**, the **MultiSelect** property determines whether check boxes or option buttons appear in the list. When **MultiSelect** is set to **Single**, option buttons appear in the list. When **MultiSelect** is set to **Multi** or **Extended**, check boxes appear in the list.

Add items to a list

You can add items to a list box or a combo box at design time and at run time. To save its contents in an item, a **ListBox** and a **ComboBox** control must be bound to a field.

What do you want to do?

Add items to a list at design time

Add items to a list at run time



Add items to a list at run time

In a **ListBox** or **ComboBox** with a single column, use the **AddItem** method to add an individual entry to the list.

In a multicolumn list box or combo box, you can use the **List** and **Column** properties to load the list from a two-dimensional <u>array</u>, as shown in the following steps.

- 1. Create a multicolumn **ListBox** or **ComboBox** control.
- 2. In VBScript, create a two-dimensional <u>array</u> that contains the items you want to put in the list.
- 3. Set the **ColumnCount** property of the list box or combo box to match the number of entries in the list. To set the property, click the property and enter a value in the **Apply** box.
- 4. Do one of the following:
 - Assign the array as the value of the **List** property. The contents of the list box will match the contents of the array exactly.
 - Assign the array as the value of the **Column** property. **Column** transposes rows and columns, so each row of the list box matches the corresponding column of the array.

Match entries in a list to entries typed by users

- 1. In design mode, drag the <u>ListBox</u> or <u>ComboBox</u> control from the <u>Control</u> <u>Toolbox</u> to the form .
- 2. Right-click the list box or combo box, and then click **Advanced Properties** on the shortcut menu.
- 3. To set the **MatchEntry** property, click the property and enter a value in the **Apply** box.

Set the MatchEntry property to	То
No matching	Provide no matching
First letter	Compare the most recently typed letter to the first letter of each entry in the list (the first match in the list is selected)
Complete	Compare the user's entry and an exact match in an entry from the list

Notes

- The matching feature resets after 2 seconds (6 seconds in the East Asian version). For example, if you have a list of the 50 United States and you type "CO" quickly, you will find "Colorado." But if you type "CO" in longer than 2 seconds, you will find "Ohio" because the auto-complete search resets between letters.
- If you select **Complete** matching, it is a good idea to sort the list entries alphabetically (you can use the <u>TextColumn</u> property to do this). If the list is not sorted alphabetically, matching may not work correctly. For example, if the list includes Alabama, Louisiana, and Alaska, respectively, then "Alabama" is considered a complete match if the user types "ala." In fact, this result is not complete because there are two entries in the list that could

match what the user entered. Sorting alphabetically eliminates this.

Things you can do with a multicolumn ListBox or ComboBox

To control the column widths of a multicolumn **ListBox** or **ComboBox**, you can specify the width, in points, for all the columns in the **ColumnWidths** property. Specifying zero for a specific column hides that column of information.

If you want to hide all but one column of a **ListBox** or **ComboBox** from the user, you can identify the column of information to appear by using the <u>TextColumn</u> property.

Similarly, you can control which column of values is used for the control when the user makes a selection by specifying the column number in the **BoundColumn** property.

Align a picture on a control

- 1. Right-click the control that has the bitmap you want to align, and then click **Advanced Properties** on the shortcut menu.
- 2. Set one or all of the following properties. To set the property, click the property and enter a value in the **Apply** box.

Set the property	То
PictureAlignment	Center the picture within the Image or align any corner to f the picture with the corresponding corner of the Image .
PictureSizeMode	Clip, stretch, or zoom the picture within the Image (stretching can distort the picture but zooming does not).
PictureTiling	Show multiple copies of the picture within the Image .

Assign a picture to a control

- 1. Right-click the control you want to assign a picture to, and then click **Advanced Properties** on the shortcut menu.
- 2. Set the **<u>Picture</u>** property. To set the property, click the property and enter a value in the **Apply** box.
- 3. In the **Load Picture** dialog box, enter the name of the picture in the **File name** box.

Note If the picture is larger than the control, Outlook scales the picture to fit the control.

Assign a picture to a form

- 1. On the form, right-click the page you want to assign a picture to, and then click **Advanced Properties** on the shortcut menu.
- 2. To set the property, click the property and enter a value in the Apply box.
 Set the property To the value
 Picture
 PictureAlignment
 TopLeft
 PictureSizeMode
 Clip
 PictureTiling
 True



Change the appearance of a control

Outlook includes several properties you can use to define the appearance of controls in your form: **ForeColor**, **BackColor**, **BackStyle**, **BorderColor**, **BorderStyle**, and **SpecialEffect**.

- **ForeColor** determines the <u>foreground color</u>. The foreground color applies to any text associated with the control, such as the caption or the control contents.
- **BackColor** and **BackStyle** apply to the control background. The background is the area within the control boundaries, such as the area surrounding the text in a control, but not the control border. **BackColor** determines the background color . **BackStyle** determines whether the background is transparent . A transparent control background is useful if your form has a background picture.

For **ForeColor** and **BackColor**, you can use the color scheme defined by your system, or you can use a custom color that you pick from the color palette. Using a system color, such as Menu Text, ensures that your form matches the colors and palette used by your applications. Custom colors do not always appear the same across systems and screen resolutions, but they do offer the widest choice of colors.

• BorderColor, BorderStyle, and SpecialEffect apply to the control border. You can use BorderStyle or SpecialEffect to choose a border type. Only one of these two properties can be used at a time. When you assign a value to one of these properties, the system sets the other property to None. With SpecialEffect, you can choose one of several border styles, but you can only use system colors for the border. BorderStyle supports only one border style, but you can choose any color that is a valid setting for BorderColor. BorderColor specifies the color of the control border and is only valid when you use BorderStyle to create the border.

Outlook supports transparency, the display of whatever is behind an object instead of its background, in two areas: the background of certain controls and in bitmaps used on certain controls.

You can show a bitmap on many controls. Certain controls support transparent bitmaps— that is, bitmaps in which one or more <u>background colors</u> are transparent. Bitmap transparency is not controlled by any control property; it is controlled by the color of the lower-left pixel in the image. Outlook does not provide a way to edit a bitmap and make it transparent; you must use a picture editor for this purpose.

Bitmaps are always transparent on the following controls: <u>CheckBox</u>, <u>CommandButton</u>, <u>Label</u>, <u>OptionButton</u>, and <u>ToggleButton</u>. In Outlook, the following do not support transparent bitmaps: the form, <u>Frame</u> control, <u>Image</u> control, and <u>MultiPage</u> control.

Transparent pictures sometimes have a hazy appearance. If you do not like this appearance, show the picture on a control that supports opaque images. If you use a transparent bitmap on a control that does not support transparent bitmaps, the bitmap appears correctly, but you can't see what's behind the bitmap.

What do you want to do?

Use a system color for a background or foreground

Use a custom color for the background or foreground of a control

Set the background color of a form

Make a control transparent

Delete a picture from a control

- 1. Right-click the control, and then click **Advanced Properties** on the shortcut menu.
- 2. Highlight the value of the **Picture** property (the word "bitmap").
- 3. Press DELETE.

Test a form in run mode

To switch between the design mode and the run mode (as it appears when used) of the form , on the **Form** menu, click **Run This Form**.

Note You can test the form outside the Outlook design environment. Open a new instance of the form.

Enter information in the form, send or post it, and then open the form to make sure it appears correctly. If the form is not correct, you can make changes to it.



Save and distribute a form

There are three ways to save a form that you have created.

- Save the form in a forms library. Use this option when you want to save a form to a location (library) for easy access. There are three forms libraries.
 - **Personal Forms Library** Forms saved in this library are accessible only to you and are stored in your mailbox. These forms are available on the item menu from the **Choose Form** command. Use this library when you have created a form for your own personal use, such as a form to track mileage.
 - **Outlook Folders** Forms saved in this library can be accessible to everyone (in a <u>public folder</u>) or only to you (in a private folder). If the form is saved in a library on your hard disk, it is accessible to you only while you are working in the folder. These forms are available on the item menu while you are in the folder. Use this library when saving forms to a specific folder, generally a public folder. For example, in a customer-tracking public folder, you could have a meeting report form and a customer profile form.
 - **Organizational Forms Library** Forms saved in this library are accessible to everyone in your organization and are stored on the server. You must have write permission to save to the server. For more information, see your administrator. These forms are available on the item menu from the **Choose Form** command. Use this library when you want to make the form available to everyone in your organization, such as a form to report vacation time. Saving to this library provides a quick and easy way to distribute and update forms.
- Save the form in the open folder. Use this option when you want to send a form to others by using e-mail or when you do not plan to share the form.
- Save the form as a file. Use this option when you want to work with the form in another program or you want to save the form as a template. You do not plan to share the form with others, or you want to send the form to others by using e-mail.

What do you want to do?

Save a form in the open folder Save a form as a file Save a form in a forms library Send a form to others by using e-mail

Save a form with the item (one-off forms)

When you are creating solutions using custom forms in Microsoft Outlook, it is important to understand how Outlook is using your custom form in relation to the items in a folder.

For instance, if you want to create a custom contact form to replace the default Outlook contact form, you would typically follow these steps:

- 1. Start with a new, default contact item to use as the basis for your custom form.
- 2. Modify the form to suit your needs.
- 3. Publish the form to the **Contacts** folder.
- 4. Set the form as the default form for the **Contacts** folder by changing the folder properties.

In this typical scenario, information about the form (the form definition) is not saved with each item. Instead, the form is stored in the location where it was published and is referenced using the **Message Class** field. This way, each item only stores the data associated with it, and its size is relatively small.

However, it is possible to have Outlook store the form definition within individual items in a folder. These items are called one-off items and will always use the form definition that is stored within the item instead of the published form.

In most situations, the form definition should not be stored within the item. The most common exception to this is a custom e-mail message form. If you are using Microsoft Exchange Server, you can publish a custom e-mail message form to the Organizational Forms Library so that it will always be available for use by everyone within the organization. In this way, you would not have to store the form definition within the item. However, if you are not using Exchange server, or if you are sending the form to another organization where the form is not available, you should select the **Send form definition with item** check box on the **Properties** page of the form when in design mode. This will

ensure that the recipient will be able to view the mail message with the custom form.

Note If the recipient is still unable to view your custom form, be sure that you customized or disabled the **Read** page of the custom e-mail form.

If the custom form contains Microsoft Visual Basic Scripting Edition (VBScript), Outlook will display the macro virus warning unless the form is published in the Microsoft Exchange Server Organizational Forms Library.

The following scenarios commonly result in items becoming one-off items.

• You have a folder-based solution whereby the form is published in the folder and the items are using the published form. You open an existing item in a folder, make changes to the form in design mode, and then save the item.

Because the form definition has changed and the form was not republished, Outlook will save the new form definition with the item. If you want to change the form for all of the items in the folder, instead of opening an existing item, follow these steps:

- 1. Open a new item based on your custom form.
- 2. Make form design changes to that item.
- 3. Republish the form with the same name.
- 4. Close and do not save changes to the item.

All of the items in the folder will now use the updated custom form the next time the items are opened, because the message class of the items still refers to the published form.

• VBScript code in the custom form changed the form definition of the item.

If VBScript code within an item programmatically changes the form, in many cases this will result in the form definition being saved with the item. The following Outlook object model methods most commonly cause this behavior:

• UserProperties .Add

• Methods and properties of the **FormDescription** object.

- Some methods or properties of controls, such as **Enabled**.
- Methods and properties of the <u>Actions</u> collection object.

Although solutions and situations vary greatly, there are some signs you should be aware of that can indicate that an item has become a one-off item.

- VBScript code in the form does not run, or a macro virus warning unexpectedly appears, indicating that the item itself, and not just a published form, contains VBScript code.
- The size of an item increases unexpectedly.
- An item's icon changes unexpectedly.



Using Visual Basic with Outlook

You can use Visual Basic to customize and extend Microsoft Outlook. Outlook allows you to control Outlook by using Visual Basic, Visual Basic for Applications and VBScript. Which you use depends on what you want your program to do.

Visual Basic is a full-featured programming language you can use to create stand-alone applications or dynamic-link libraries (DLLs) that extend other applications. Visual Basic for Applications is a subset of Visual Basic that is run within an application to extend its capabilities. VBScript is a simplified version of Visual Basic for Applications and is run within an Outlook <u>item</u>. In all cases, these programming languages control Outlook through its object model.

Learn about the **Outlook object model**.

If you want to create a separate application that accesses data stored by Outlook and uses Outlook to send and receive messages, use Visual Basic to create the application (you can also use other programming languages, such as C++, to control Outlook through its object model). You can also use Visual Basic to create a DLL that can extend Outlook as a COM add-in.

You use Visual Basic for Applications in one of two ways: You can use Visual Basic for Applications in other applications (such as Microsoft Excel or Microsoft Word) to automate Outlook, or you can use Visual Basic for Applications within Outlook to control Outlook. If you expect your users to be using another application most of the time, and you want to give them the ability to send a message using Outlook or to access information stored by Outlook, write Visual Basic for Applications programs in that application that control Outlook through the Outlook object model. If, on the other hand, you want to write Visual Basic code that customizes how Outlook works (like a macro), use Visual Basic for Applications within Outlook.

You can extend the functionality of Outlook forms by using VBScript. VBScript programs are stored within a <u>form</u>. Because the program code is contained within the form, it can be sent with an item to another user. An important consideration in choosing which kind of the Visual Basic programming language

you will use is the type of events you want your program to respond to. Because VBScript code is associated with a particular item, code that responds to events in specific items (such as when a particular item is opened or a value in a field is changed) is easiest to write using VBScript. If, on the other hand, you want your program to respond to events that occur in the application, in Windows Explorer, in folders, or in all items, then you should write your program using Visual Basic or Visual Basic for Applications.

Code written for Visual Basic or Visual Basic for Applications often does not work in VBScript without modification. For example, you must replace all builtin constants written in Visual Basic for Applications with the literal numeric values of those constants in VBScript. And VBScript uses only the **Variant** data type.

Learn about constants and variables in VBScript.

In Outlook Visual Basic for Applications and VBScript, when you reference the **Application** object to use **CreateObject** or **GetObject**, you simply use **Application**. For example, the following code displays the Tasks folder:

```
Set olMAPI = Application.GetNameSpace("MAPI")
olMAPI.GetDefaultFolder(13).Display
```

In Visual Basic or Visual Basic for Applications in other applications, you must explicitly create the **Application** object:

```
Set myOlApp = CreateObject("Outlook.Application")
Set olMAPI = myOlApp.GetNameSpace("MAPI")
olMAPI.GetDefaultFolder(olFolderTasks).Display
```

About using VBScript in Outlook

Microsoft VBScript is a powerful scripting language based on Microsoft Visual Basic that enables you to control objects, folders, forms, items, and controls within a form. For example, you can change properties and values of controls on a page, modify the default Microsoft Outlook item events, and even create automated procedures, such as mailing a notice to all the contacts in a Contacts folder.

You add VBScript code to an Outlook form to respond to <u>Click</u> events that are fired by controls on the form, or to respond to events fired by the items that have the same message class as the form. VBScript makes it especially easy to respond to item events because the VBScript code executes in the context of the item, so you don't have to set an object variable to point to the item. In addition, VBScript code is compact and can be contained within a form sent to other users.

With VBScript, you have full access to the Microsoft Outlook object model, except for two areas: VBScript code cannot respond to events other than item and form events, and you cannot use named constants defined in the Outlook object type library.

You can also use Visual Basic for Applications in Outlook to respond to Outlook events and to create macros that automate procedures. Unlike VBScript code, however, Visual Basic for Applications code cannot be contained in a form and so cannot accompany an item that is sent to other users.

For more information about using VBScript, see <u>Create custom forms by using</u> <u>VBScript</u>.



How can I prevent the VBScript code from running?

To prevent any VBScript code from running, hold down the SHIFT key. For example, hold down SHIFT while you open an <u>item</u> to prevent the VBScript code for the Open event from running.

Variants supported in VBScript

Microsoft VBScript in Outlook uses only the Variant data type.

A **Variant** is a special kind of data type that can contain different kinds of information, depending on how it's used. Because **Variant** is the only data type in VBScript, it's also the data type returned by all functions in VBScript.

At its simplest, a **Variant** can contain either numeric or string information. A **Variant** behaves as a number when you use it in a numeric context and as a string when you use it in a string context. That is, if you're working with data that looks like numbers, VBScript assumes that it is numbers and does the thing that is most appropriate for numbers. Similarly, if you're working with data that can only be string data, VBScript treats it as string data. Of course, you can always make numbers behave as strings by enclosing them in quotation marks (" ").



Using the Script Editor

The Microsoft Script Editor allows you to add VBScript procedures that respond to events generated by <u>items</u> or form controls.

To open the Script Editor

- 1. Open an item of the type to which you want to add code.
- 2. On the **Tools** menu, point to **Forms** and then click **Design This Form**.
- 3. On the **Form** menu, click **View Code**.

The Script Editor makes it easy to insert the template for an item event handler.

To insert a blank item event handler

- 1. On the **Script** menu, click **Event Handler**.
- 2. Select the event you want to respond to, and then click **OK**.

The Script Editor can move the insertion point to a specific line of code. This makes it easy to debug the script when Microsoft Outlook reports an error at a specific location.

To move to a specific line

- 1. On the **Edit** menu, click **Go To**.
- 2. In the **Line Number** box, type in the number of the line of code to which you want to go.

Learn about <u>the Script Debugger</u>.

Learn about the Outlook object browser.



About the Outlook object browser

The Microsoft Outlook object browser displays the <u>classes</u>, properties, methods, and events available from the Outlook object library. The object browser lets you view and insert these objects into the Script Editor and obtain information about the syntax for using the object.

Learn about viewing and using the object browser.



About the Outlook script debugger

The Microsoft Script Debugger provides you with a comprehensive debugging environment for testing and correcting errors in the VBScript code that you created for your Microsoft Outlook <u>forms</u>. The Script Debugger is a shared component that can also be used to track down errors in any Microsoft ActiveXenabled scripting language and to debug Java applets, beans, and ActiveX components.

The Microsoft Script Debugger lets you debug both <u>client scripts</u> and <u>server</u> <u>scripts</u>.

The Microsoft Script Debugger works the way many debuggers do, by allowing you to:

- View the source code of the script that you are debugging.
- Control the pace of the script execution.
- View and change variable and property values.
- View and control script flow.

The script debugger is only available in Outlook at <u>run time</u>.

Learn about <u>using the Script Debugger in Outlook</u>.

Common things you can do with VBScript

The following are common things you can do with VBScript in an Outlook Form.

What do you want to do?

Create an Outlook item

Add a Click event for a control

Add an Outlook event

Reference the active user

Change the value of a field

Reference a folder

Use properties from the Microsoft Exchange Server global address list

Put a list of fields and values in the message body



Create custom forms by using VBScript

Microsoft VBScript is a subset of Visual Basic for Applications. You can use VBScript to create procedures that control Microsoft Outlook folders, objects, <u>items</u>, and properties. VBScript in Outlook requires a special object syntax that has some differences from referencing objects in Visual Basic for Applications.

Learn about the Outlook object model.

You can choose the Outlook item on which to base your custom form.

You can extend Outlook forms by using custom controls from the **Control Toolbox**. Outlook forms can use most of the properties and methods that come with the controls. Since controls cannot store values, to store the value you need to <u>bind</u> the control to an Outlook field.

The Outlook object browser displays the <u>classes</u>, properties, methods, events, and <u>constants</u> available from the Outlook object library. The object browser lets you view and use objects in the Microsoft Script Editor and obtain information about the syntax for using the object.

Learn about viewing and using the object browser.

The Microsoft Script Debugger provides you with a comprehensive debugging environment for testing and correcting errors in the VBScript code for your forms.

Learn about <u>the Script Debugger</u>.

Set global variables for a form

A global <u>variable</u> is available to any procedure in a form while the script is running. To set a global variable, assign the value to the variables before any procedures.



Referencing controls on an Outlook form

If you need to refer to a control on an Outlook form within a procedure, you must also reference the inspector, page, and controls collection that contains the control, even if you are referencing the control with its own event procedure. The following example shows how to change the caption of a command button when it's clicked. To test this example, in <u>design mode</u> create a command button with the default name CommandButton1 on the page P.2.

```
Sub CommandButton1_Click
   Set myButton = Item.GetInspector.ModifiedFormPages("P.2")_
        .Controls("CommandButton1")
   myButton.Caption = "New Caption"
End Sub
```

Form Events

Form events occur when something happens to an item displayed in a form, such as when it's saved or opened or when a user-defined action is started.

Most often, form events are handled by VBScript code within the form itself.

Some events can be cancelled. That is, your event handler can prevent Microsoft Outlook from performing the default action associated with the event. For example, you can write an event handler for the **Forward** event to prevent an item from being sent to recipients who are not on a list of approved recipients. Learn about <u>canceling an event</u>.

The following table lists the form events supported by Outlook.

Event	Cancelable?	Description
AttachmentAdd	No	Occurs when an attachment has been added to the item
AttachmentRead	No	Occurs when an attachment has been opened for reading
BeforeAttachmentSave	Yes	Occurs before an attachment is saved
BeforeCheckNames	Yes	Occurs before Outlook starts resolving names in the recipients collection of the item
Close	Yes	Occurs before Outlook closes the inspector displaying the item
CustomAction	Yes	Occurs before Outlook executes a custom action of an item
CustomPropertyChange	No	Occurs when a custom item property has changed
Forward	Yes	Occurs before Outlook executes the Forward action of an item
Open	Yes	Occurs before Outlook opens an inspector to display the item
PropertyChange	No	Occurs when an item property has

		changed
Read	No	Occurs when an item is opened for editing by a user
Reply	Yes	Occurs before Outlook executes the Reply action of an item
ReplyAll	Yes	Occurs before Outlook executes the Reply to All action of an item
Send	Yes	Occurs before Outlook sends the item
Write	Yes	Occurs before Outlook saves the item in a folder

Control Events

Outlook form controls support only one event, the <u>Click</u> event.

A control bound to a field does not fire the **Click** event. You must handle the appropriate <u>field event</u> to detect a user's interaction with a control bound to a field.

The following controls fire the **Click** event whenever a user clicks anywhere in the control.

CheckBox

CommandButton

Frame

Image

<u>Label</u>

OptionButton

ToggleButton

The following controls fire the **Click** event when the user selects an item in the list.

ComboBox

ListBox

The following controls do not support the **Click** event.

MultiPage

ScrollBar

SpinButton

TabStrip

TextBox

While the **MultiPage** control itself does not support the **Click** event, an individual **Page** on a **MultiPage** control will fire the **Click** event if the user clicks inside the client area of the page, but not if the user clicks the tab associated with the page.

To detect a change in a **TextBox** control, bind the control to a field and then handle the appropriate <u>field event</u>.

Field Events

Microsoft Outlook provides two events to notify your program that a field (property) in an item has changed. The **PropertyChange** event is fired whenever a standard Outlook field in an item has changed. Outlook fires the **CustomPropertyChange** event whenever a user-defined field changes.

A control that is bound to a field does not fire the <u>Click</u> event, whether the control was selected from the **Control Toolbox** and subsequently bound to a field, or was selected from the **Field Chooser**. Consequently, you must use the **PropertyChange** or **CustomPropertyChange** event to detect user interaction with a bound control.

Add an Outlook event

1. Open the Script Editor.

How?

- 2. On the **Script** menu, click **Event Handler**.
- 3. In the **Insert Events Handler** box, click the event that you want to use, and then click **Add**.

Canceling an event

Microsoft Outlook calls event handlers in your program to allow your program to respond to such events as actions that the user takes or changes in the message store. Each event is accompanied by a default action that Outlook performs as a result of the event. For example, when the **Open** event occurs for an item, by default Outlook displays the item in an inspector window.

Some events only notify your program that a particular event has occurred. For these events, your event handler simply responds to the event. With other events, Outlook allows your event handler to cancel the event, that is, to instruct Outlook not to perform the default action associated with the event. In the case of the **Open** event, for example, your program can prevent Outlook from displaying the item in an inspector. If an event can be cancelled, the reference topic describing the event indicates how to cancel the event.

If an event can be cancelled, an event handler written in Microsoft Visual Basic or Microsoft Visual Basic for Applications receives a parameter that it sets before returning to indicate whether the event should be cancelled. For example, an event handler for the **Open** event written in Visual Basic for Applications might look like this. This example assumes that the value of OpenOK is set elsewhere.

```
Sub myItem_Open(byRef Cancel as Boolean)
    If OpenOK Then
        Cancel = False ' Outlook performs default action
    Else
        Cancel = True ' Outlook does not perform default action
    EndIf
End Sub
```

Because of limitations in VBScript, however, this syntax cannot be used. An event handler for the **Open** event in the script of an item must be written as a function. To cancel the event, the value of the function is set to **False** before returning, as in the following example.

```
Function Item_Open()
If OpenOK Then
```

Item_Open = True ' Outlook performs default action Else Item_Open = False ' Outlook does not perform default action End If End Function



Test tips for using VBScript in Outlook

There are several ways to test code in Outlook. If there is a syntax error in the code, when you run the form, an error message appears with the line number of the code. You can then go to the specific line of code. You can also use the Script Debugger and other methods to isolate <u>run-time</u> errors.

What do you want to do?

Go to a line of code

<u>Use the Script Debugger</u>

Use message boxes to return values

Set global variables for a form

<u>Use the PropertyChange event to test procedures</u>

About the object environment

There are two ways to write code for Outlook:

- From outside the application, such as by using Microsoft Visual Basic or Microsoft Visual Basic for Applications in Microsoft Excel or another application.
- From inside the application, such as by using Visual Basic for Applications or by using VBScript with an Outlook form.

Learn more about <u>the differences between using Visual Basic for Applications</u> <u>and VBScript</u>

The major components of the Outlook object model are:

Application	The top of the object hierarchy that represents the entire application the application and create items and objects. For example, this co- Basic for Applications or VBScript: Application.CreateItem(1).Display
NameSpace	Represents the MAPI message store where all the Outlook items and off Outlook and for referencing the default folders such as M example, this code references the active user in Outlook Visual B Application.GetNameSpace("MAPI").CurrentUser
Explorer	Represents the Outlook window. Enables you to show, return, and code shows the active Outlook window in Outlook Visual Basic f Application.ActiveExplorer.Display
Folders	There are two folder objects, the Folders collection object that en and the MAPIFolder object that enables you to work with a sing collection of folders named Personal Folders in Outlook Visual B Application.GetNameSpace("MAPI").Folders("Personal Folc
Outlook items	There are two item objects, the Items collection object that enable the item objects that represents the standard item types in Outlool message. In VBScript, the active item is assumed, so you do not r For example, this code sets the Subject field of the active message Item.Subject = "New Subject"
Inspector	References forms. Use to show forms and pages. For example, the Outlook Visual Basic for Applications or VBScript:

Application.ActiveInspector.SetCurrentFormPage("Options

AddressEntry	Each AddressEntry object in the AddressEntries collection hole process to which the messaging system can deliver messages.
AddressList	The AddressList object is an address book that contains a set of <i>i</i> available through the parent AddressLists collection.
Exception	The Exception object holds information about one instance of an exception to a recurring series. Unlike most of the other Outlook object.
Control	There are two control objects, the Controls collection object that page and the specific control object that enables you to work with Caption of a CommandButton control named "CommandButton
	Item.GetInspector.ModifiedFormPages("Test").Controls("(



Use the Outlook Object Browser

The Outlook object browser displays the <u>classes</u>, properties, methods, events, and constants available from the Outlook object library. The object browser lets you view and use objects in the Script Editor and obtain information about the syntax for using the object.



How to use the Outlook object browser

To view the Outlook object browser:

1 Open a <u>form</u> in <u>design mode</u>.

2 In the **Form** menu, click **View Code** to view the Script Editor.

3 In the Script Editor, click **Object Browser** on the **Script** menu or press **F2**.

All of the available Outlook objects are listed in the **Classes** pane of the object browser in alphabetical order.

To view the members of an object, select the object in the **Classes** pane. The members of this object appear in alphabetical order in the **Members of** pane. The heading at the top of this pane will reflect the name of the object that you select. For example, if you select the **AppointmentItem** object in the **Classes** pane, the heading of the **Members of** pane will appear as **Members of AppointmentItem**.

The details pane shows the definition of the selected member. This text is readonly and cannot be copied and pasted into the Script Editor.

To insert an item from the object browser into the Script Editor:

1 In the Script Editor, position your cursor at the location for insertion.

2 Select the desired object in the **Classes** pane.

3 Select the desired member of this object in the **Members of** pane.

4 Click the **Insert** button.

Note The **Insert** button remains unavailable until a member of the object is selected.



How to use the Outlook script debugger

To use the script debugger, open a <u>form</u> in <u>design mode</u>.

To add VBScript to the form, on the **Form** menu, click on **View Code**. In the Script Editor window, add the necessary VBScript for your form.

On the **Form** menu, click **Run This Form**.

While your form is in <u>run mode</u>, on the **Tools** menu, select **Forms** and click **Script Debugger**.

You can also use the Stop statement in code to launch the debugger.

Note If there is no VBScript for this form, the **Script Debugger** menu item will be unavailable.

The VBScript for this form will appear in a read-only window. You cannot change text in this window.

On the Help menu of the Microsoft Script Debugger, select Help Topics for more detailed information about:

- Debugging scripts
- Viewing source code
- Controlling program execution
- Viewing and changing values
- Viewing and controlling program flow

Constants and variables in VBScript

In VBScript, constants must be referenced by their numeric values. The constant string does not work and returns a value of 0, which gives unpredictable results.

There are two types of variables. Procedure-level variables that are used only within a procedure and script-level variables that are available to all the procedures within your script. Declare script-level variables at the top of your script. Declare procedure-level variables inside procedures. You can use procedure-level variables with the same name in different procedures because each variable is recognized only by the procedure in which it's declared. When the procedure exits, the variable ends. Variables that refer to Outlook objects can be either procedure-level or script-level variables. However, the value of the variable must be set within a procedure. Do not attempt to access Outlook objects outside of a procedure.

Rules about variables:

- Must begin with an alphanumeric character.
- Cannot contain an embedded period.
- Cannot exceed 255 characters.
- Cannot use more than 127 procedure-level variables (arrays count as a single variable).
- Cannot use more than 127 script-level variables.

Referencing fields

When you need to access the fields in an item, the method you use depends on whether the field is a standard, built-in Microsoft Outlook field, or a custom field.

In either case, you do not access the field directly. Instead, you refer to the field as a property of the item you' re working with.

For example, to retrieve the text from the Subject field of a mail message, you use the **Subject** property of the item, as shown in the following VBScript example.

```
mySubject = Item.Subject
```

If the field is a custom (user-defined) field, you access it using the **UserProperties** property of the item, as shown in the following VBScript example. This example assumes that the item already contains a custom field named ReferredBy.

```
MyReferral = Item.UserProperties("ReferredBy")
```

Reference a folder

To reference a folder by the name of the folder, use the following code.

Application.GetNameSpace("MAPI").Folders("Personal Folders").Folders

To reference a folder by a number, use the following code. In this example, the first folder in the folder collection Personal Folders is referenced.

Application.GetNameSpace("MAPI").Folders("Personal Folders").Folders

To reference any of the default Outlook folders, use the **GetDefaultFolder** method. Use the value from the table below to specify the folder you want to create.

Application.GetNameSpace("MAPI").GetDefaultFolder(6)

Default folder Has the value

Deleted Items	3
Outbox	4
Sent Items	5
Inbox	6
Calendar	9
Contacts	10
Journal	11
Notes	12
Tasks	13

Creating a new item

To create a new item, use the **CreateItem** method of the **Application** object. This method returns an object that you can then use to work with the item.

The following Microsoft Visual Basic for Applications example shows how to create a mail message, add text to its subject and body, and display it. To use this sample, create a command button named Command1 on a form.

```
Private Sub Command1_Click()
   Dim myOLApp As New Outlook.Application
   Dim myOLItem As Outlook.MailItem
   Set myOLItem = myOLApp.CreateItem(olMailItem)
   With myOLItem
        .Subject = "Sample item"
        .Body = "This is a sample message."
   End With
   myOLItem.Display
End Sub
```

The following example shows how to perform the same task using VBScript in a form.

```
Sub CommandButton1_Click()
   Set myOLItem = Application.CreateItem(0)
   myOLItem.Subject = "Sample item"
   myOLItem.Body = "This is a sample message."
   myOLItem.Display
End Sub
```



Create an Outlook item

This procedure uses the <u>Click</u> event to call **CreateItem** to create and show an appointment when a user clicks CommandButton1.

In <u>design mode</u>:

- 1. Using the **Control Toolbox**, place a **<u>CommandButton</u>** on your <u>form</u>.
- 2. Open the Script Editor.

How?

3. Enter the following code, using the value from the table below to specify the type of <u>item</u> that you want to create.

```
Sub CommandButton1_Click
    Application.CreateItem(1).Display
End Sub
```

Type of item	Has the value
MailItem	0
AppointmentItem	1
ContactItem	2
TaskItem	3
JournalItem	4
NoteItem	5
PostItem	6
DistListItem	7

Referencing existing items in a folder

There are a number of ways you can reference existing items in a folder using Microsoft Visual Basic. This topic provides information about:

- Using a For ... Next or For Each ... Next loop
- Using the **Items** collection
- Using the **Find** method
- Using the **Restrict** method

Using a For...Next or For Each...Next Loop

Typically these statements are used to loop through all of the items in a folder. The **Items** collection contains all the items in a particular folder, and you can specify which item to reference by using an index with the **Items** collection. This is typically used with the For I = 1 to n programming construct.

You can use For Each...Next to loop through the items in the collection without specifying an index. Both approaches achieve the same result.

The following examples use For...Next to loop through all the contacts in the Contacts folder and display the Full Name field in a dialog box.

```
' Visual Basic/Visual Basic for Applications code example.
Set ol = New Outlook.Application
Set olns = ol.GetNameSpace("MAPI")
' Set MyFolder to the default contacts folder.
Set MyFolder = olns.GetDefaultFolder(olFolderContacts)
' Get the number of items in the folder.
NumItems = MyFolder.Items.Count
' Set MyItem to the collection of items in the folder.
Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact
' Loop through all of the items in the folder.
For I = 1 to NumItems
   MsqBox MyItems(I).FullName
Next
' VBScript code example.
Set olns = Item.Application.GetNameSpace("MAPI")
' Set MyFolder to the default contacts folder.
Set MyFolder = olns.GetDefaultFolder(10)
```

The following examples use For Each...Next to achieve the same result as the preceding examples:

' VBScript code example. Set olns = Item.Application.GetNameSpace("MAPI") ' Set MyFolder to the default contacts folder. Set MyFolder = olns.GetDefaultFolder(10) ' Set MyItem to the collection of items in the folder. Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact For Each SpecificItem in MyItems

```
MsgBox SpecificItem.FullName
```

Using the Items Collection

You can also use the **Items** collection and specify a text string that matches the Subject field of an item. The following examples display an item in the Inbox whose subject contains "Please help on Friday!"

```
' Visual Basic/Visual Basic for Applications code example.
Set ol = New Outlook.Application
Set olns = ol.GetNameSpace("MAPI")
' Set MyFolder to the default Inbox.
Set MyFolder = olns.GetDefaultFolder(olFolderInbox)
```

```
Set MyItem = MyFolder.Items("Please help on Friday!")
MyItem.Display
' VBScript code example.
Set olns = Item.Application.GetNameSpace("MAPI")
' Set MyFolder to the default Inbox.
Set MyFolder = olns.GetDefaultFolder(6)
Set MyItem = MyFolder.Items("Please help on Friday!")
MyItem.Display
```

Using the Find Method

Use the **Find** method to search for an item in a folder based on the value of one of its fields. If the search is successful, you can then use the **FindNext** method to check for additional items that meet the same search criteria.

The following examples search to see if you have any high priority tasks.

```
' Visual Basic/Visual Basic for Applications code example.
Set ol = New Outlook.Application
Set olns = ol.GetNamespace("MAPI")
Set myFolder = olns.GetDefaultFolder(olFolderTasks)
Set MyTasks = myFolder.Items
' Importance corresponds to Priority on the task form.
Set MyTask = MyTasks.Find("[Importance] = ""High""")
If MyTask Is Nothing Then ' the Find failed
   MsgBox "Nothing important. Go party!"
Else
   MsgBox "You have something important to do!"
End If
' VBScript code example.
Set olns = Item.Application.GetNamespace("MAPI")
Set myFolder = olns.GetDefaultFolder(13)
Set MyTasks = myFolder.Items
  Importance corresponds to Priority on the task form.
Set MyTask = MyTasks.Find("[Importance] = ""High""")
If MyTask Is Nothing Then ' the Find failed
   MsgBox "Nothing important. Go party!"
Else
   MsgBox "You have something important to do!"
End If
```

Using the Restrict Method

The **Restrict** method is similar to the **Find** method, but instead of returning a single item, it returns a collection of items that meet the search criteria. For example, you could use this method to find all contacts that work at the same company.

The following examples display all of the contacts that work atProseWare Corporation:

```
' Automation code example.
Set ol = New Outlook.Application
Set olns = ol.GetNameSpace("MAPI")
Set MyFolder = olns.GetDefaultFolder(olFolderContacts)
Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact
MyClause = "[CompanyName] = ""ProseWare"""
Set MyPWItems = MyItems.Restrict(MyClause)
For Each MyItem in MyPWItems
   MyItem.Display
Next
' VBScript code example.
Set olns = Item.Application.GetNameSpace("MAPI")
Set MyFolder = olns.GetDefaultFolder(10)
Set myItems = myFolder.Items.Restrict("[MessageClass] = 'IPM.Contact
MyClause = "[CompanyName] = ""ProseWare"""
Set MyPWItems = MyItems.Restrict(MyClause)
For Each MyItem in MyPWItems
   MyItem.Display
Next
```

Filtering items in a collection

You can use the Microsoft Outlook object model to return information about all items in a folder. Often, however, the desired objective is to search for a specific item or to retrieve a subset of the items in the folder. Consider the following examples:

- You are developing a Microsoft Access database. When the user enters a new contact record, you want to give the user the ability to click a button to check whether a contact with the same name already exists in Outlook. If a match is found, you can retrieve all the fields for the contact and automatically fill in the Access database record. In this situation, if the user filled in the first and last name fields on the Access form, you can use the **Find** method in the Outlook object model to search for a match against the Outlook Full Name field. If you want to make sure there are no additional contacts in Outlook with the same name, you can then use the **FindNext** method to conduct the same search again.
- You are writing a Microsoft Visual Basic program to automatically schedule appointments in users' calendars. In order to do this, you need to retrieve a user's appointments for a given day. In this case, you would use the **Restrict** method to retrieve all appointments that fall on a particular day.

While the **Find** and **Restrict** methods perform different functions, the syntax for both is similar. Following are some tips for using these methods:

- The filters used with **Find** and **Restrict** are not case-sensitive.
- It is not possible to use these methods if you need to search for a string that's contained within an Outlook field, commonly called a "Contains" operation. If you need to perform a Contains operation, you can iterate through all of the items in the folder and use the Visual Basic **InStr** function to see if your search string is contained within an Outlook field.
- To create a filter that performs a "begins with" operation, use the >, < , >=, and <= operators. For example, to search for all contacts whose last names begin with "Mc", use this filter: ""[LastName] >= ""Mc"" and [LastName] < ""Md"""
- You can use the contents of a variable as part of the filter.

Working with command bars

In Microsoft Outlook 2000, toolbars, menu bars, and shortcut menus are all controlled programmatically as one type of object: command bars. All the following items are represented in Microsoft Visual Basic by **CommandBar** objects:

- Menu bars, toolbars, and shortcut menus
- Menus on menu bars and toolbars
- Submenus on menus, submenus, and shortcut menus

You can modify any built-in menu bar or toolbar, and you can create and modify custom toolbars, menu bars, and shortcut menus to deliver with your Visual Basic application. You can present the features of your application as buttons on toolbars or as groups of command names on menus. Because toolbars and menus are both command bars, you use the same kind of controls on both of them. Menu bars and toolbars can both contain menus.

In Visual Basic, buttons and menu items are represented by **CommandBarButton** objects. The pop-up controls that display menus and submenus are represented by **CommandBarPopup** objects.

You can also add text boxes, drop-down list boxes, and combo boxes to any command bar. These three types of controls are all represented in Visual Basic by **CommandBarComboBox** objects.

Note Although they share similar appearances and behaviors, command bar controls and ActiveX controls are not the same. You cannot add ActiveX controls to command bars, and you cannot add command bar controls to documents or forms.

The built-in command bar controls in Outlook are also represented by **CommandBarButton**, **CommandBarPopup**, and **CommandBarComboBox** objects, but their appearances and behaviors may be different from those you can add yourself. You can modify the location and appearance of built-in controls, you cannot modify their built-in behavior.

Reference the active user

Use **GetNameSpace** to return the Outlook **NameSpace** object from the **Application** object, and then use the **CurrentUser** property to return a **Recipient** object repesenting the active user, as shown in the following example.

Set myUser = Application.GetNameSpace("MAPI").CurrentUser

Use current user properties from the Microsoft Exchange Server global address list

In code, open an OLE messaging session and log on, and then use the following table to reference the property you want to use.

```
Set olemSession = Application.CreateObject("MAPI.Session")
ReturnCode = olemSession.Logon( Application.GetNameSpace("MAPI").Cur
myPage = Item.GetInspector.ModifiedFormPages("Message")
Set myUser = olemSession.CurrentUser
Item.UserProperties.Find("Name") = myUser.Name
Item.UserProperties.Find("Messaging Address") = myUser.Address
Item.UserProperties.Find("MAPI First Name") = myUser.Fields.item(&h3
```

Address Book Proper	ty Reference
PR_GIVEN_NAME	&h3a06001e
PR_INITIALS	&h3a0a001e
PR_SURNAME	&h3a11001e
PR_7BIT_DISPLAY_NAME	&h39ff001e
PR_STREET_ADDRESS	&h3a29001e
PR_LOCALITY	&h3a27001e
PR_STATE_OR_PROVINCE	&h3a28001e
PR_POSTAL_CODE	&h3a2a001e
PR_COUNTRY	&h3a26001e
PR_TITLE	&h3a17001e
PR_COMPANY_NAME	&h3a16001e
PR_DEPARTMENT_NAME	&h3a18001e
PR_OFFICE_LOCATION	&h3a19001e
PR_ASSISTANT	&h3a30001e
PR_BUSINESS_TELEPHONE_	NUMBER &h3a08001e

Use the PropertyChange event to test procedures

Perform the following steps to test simple procedures. Replace the code below with the code that you want to test. Each time a user changes the value of the Importance field, or any other default field, the code runs.

1. Open the Script Editor.

How?

- 2. On the Script menu, click Event Handler.
- 3. In the **Events** box, double-click **PropertyChange**.
- 4. Add the following code in the event:

MsgBox "This is my test procedure"

- 5. On the **Form** menu, click **Run This Form**.
- 6. Click the ! icon on the toolbar. The message box will appear.
- 7. Click **OK** to close the message box.



CheckBox control

Displays the selection state of an item.

Remarks

Use a **CheckBox** to give the user a choice between two values such as *Yes/No*, *True/False*, or *On/Off*. When the user selects a **CheckBox**, it displays a special mark (such as an **X**) and its current setting is *Yes*, *True*, or *On*. If the user does not select the **CheckBox**, it is empty and its setting is *No*, *False*, or *Off*. Depending on the value of the **TripleState** property, a **CheckBox** can also have a <u>null</u> value.

If a **CheckBox** is <u>bound</u> to a <u>data source</u>, changing the setting changes the value of that source. A disabled **CheckBox** shows the current value, but is dimmed and does not allow changes to the value from the user interface.

You can also use check boxes inside a group box to select one or more of a group of related items. For example, you can create an order form that contains a list of available items, with a **CheckBox** preceding each item. The user can select a particular item or items by checking the corresponding **CheckBox**.

The default property of a **CheckBox** is the **<u>Value</u>** property.

Note The **ListBox** also lets you put a check mark by selected options. Depending on your application, you can use the **ListBox** instead of using a group of **CheckBox** controls.



ComboBox control

Combines the features of a **ListBox** and a **TextBox**. The user can enter a new value, as with a **TextBox**, or the user can select an existing value as with a **ListBox**.

Remarks

If a **ComboBox** is <u>bound</u> to a <u>data source</u>, the **ComboBox** inserts the value entered or selected by the user into that data source. If a multicolumn combo box is bound, then the <u>BoundColumn</u> property determines which value is stored in the bound data source.

The list in a **ComboBox** consists of rows of data. Each row can have one or more columns, which can appear with or without headings. Some applications do not support column headings, others provide only limited support.

The default property of a **ComboBox** is the <u>Value</u> property.

Note If you want more than a single line of the list to appear at all times, you might want to use a **ListBox** instead of a **ComboBox**. If you want to use a **ComboBox** and limit values to those in the list, you can set the **Style** property of the **ComboBox** so the control looks like a drop-down list box.

CommandButton control

Starts, ends, or interrupts an action or series of actions.

Remarks

Requires VBScript.

Syntax

Sub CommandButton_Click()

'write event code here

End Sub

The macro or event procedure assigned to the **CommandButton's** <u>Click</u> event determines what the **CommandButton** does. For example, you can create a **CommandButton** that opens another form. You can also display text, a picture, or both on a **CommandButton**.

The default property of a **CommandButton** is the <u>Value</u> property.

The only event for a **CommandButton** is the **Click** event.

Font Object

Defines the characteristics of the text used by a control.

Each control has its own **Font** object to let you set its text characteristics independently of the characteristics defined for other controls. Use font properties to specify the font name, to set bold, italic, or underlined text, or to adjust the size of the text.

The default property for the **Font** object is the **Name** property.



Frame control

Creates a functional and visual <u>control group</u>.

Remarks

All option buttons in a **Frame** are mutually exclusive, so you can use the **Frame** to create an option group. You can also use a **Frame** to group controls with closely related contents.For example, in an application that processes customer orders, you might use a **Frame** to group the name, address, and account number of customers.

You can also use a **Frame** to create a group of **<u>ToggleButtons</u>**, but the toggle buttons are not mutually exclusive.

To create a group of mutually exclusive **<u>OptionButton</u>** controls, you can put the buttons in a **Frame**on your form, or you can use the **<u>GroupName</u>** property.



Image control

Displays a picture on a form.

Remarks

The **Image** control lets you display a picture as part of the data in a form. For example, you might use an **Image** to display employee photographs in a personnel form.

The **Image** lets you crop, size, or zoom a picture, but does not allow you to edit the contents of the picture. For example, you cannot use the **Image** to change the colors in the picture, to make the picture <u>transparent</u>, or to refine the image of the picture. You must use image editing software for these purposes.

The **Image** supports the following file formats:

- *.bmp
- *.cur
- *.gif
- *.ico
- *.jpg
- *.wmf

Note You can also display a picture on a **Label**. However, a **Label** does not let you crop, size, or zoom the picture.<P>

Label control

Displays descriptive text.

Remarks

A **Label** control on a form displays descriptive text such as titles, captions, pictures, or brief instructions. For example, labels for an address book might include a **Label** for a name, street, or city.

The default property for a **Label** is the **<u>Caption</u>** property.

Note You can also display a picture on a **Label**. However, a **Label** does not let you crop, size, or zoom the picture.



ListBox control

Displays a list of values and lets you select one or more.

Remarks

If the **ListBox** is <u>bound</u> to a <u>data source</u>, the **ListBox** stores the selected value in that data source.

The **ListBox** can either appear as a list or as a group of **OptionButton** controls or **CheckBox** controls.

The default property for a **ListBox** is the **<u>Value</u>** property.

The default event for a **ListBox** is the <u>Click</u> event.

Note You can't drop text into a drop-down **ListBox**.

ListBox styles

You can choose between two presentation styles for a **ListBox**. Each style provides different ways for users to select items in the list.

If the style is **Plain**, each item is on a separate row; the user selects an item by highlighting one or more rows.

If the style is **Option**, an **OptionButton** or **CheckBox** appears at the beginning of each row. With this style, the user selects an item by clicking the option button or check box. Check boxes appear only when the **MultiSelect** property is **True**.



MultiPage control

Presents multiple screens of information as a single set.

Remarks

A **MultiPage** is useful when you work with a large amount of information that can be sorted into several categories. For example, use a **MultiPage** to display information from an employment application. One page might contain personal information such as name and address; another page might list previous employers; a third page might list references. The **MultiPage** lets you visually combine related information, while keeping the entire record readily accessible.

New pages are added to the right of the currently selected page rather than adjacent to it.

A **MultiPage** is a control that contains a <u>collection</u> of one or more pages.

Each **Page** of a **MultiPage** is a form that contains its own controls, and as such, can have a unique layout. Typically, the pages in a **MultiPage** have tabs so the user can select the individual pages.

By default, a **MultiPage** includes two pages, called Page1 and Page2. Each of these is a **Page** object, and together they represent the **Pages** collection of the **MultiPage**. If you add more pages, they become part of the same **Pages** collection.

The default property for a **MultiPage** is the <u>Value</u> property, which returns the index of the currently active **Page** in the **Pages** collection of the **MultiPage**.

Note The **MultiPage** control does not support the <u>Click</u> event.



OptionButton control

Shows the selection status of one item in a group of choices.

Remarks

Use an **OptionButton** to show whether a single item in a group is selected. Note that each **OptionButton** in a **Frame** is mutually exclusive.

If an **OptionButton** is <u>bound</u> to a <u>data source</u>, the **OptionButton** can show the value of that data source as either *Yes/No*, *True/False*, or *On/Off*. If the user selects the **OptionButton**, the current setting is *Yes*, *True*, or *On*. If the user does not select the **OptionButton**, the setting is *No*, *False*, or *Off*. For example, an **OptionButton** in an inventory-tracking application might show whether an item is discontinued. If the **OptionButton** is bound to a data source, then changing the setting changes the value of that data source. A disabled **OptionButton** is dimmed and does not show a value.

Depending on the value of the **<u>TripleState</u>** property, an **OptionButton** can also have a <u>null</u> value.

You can also use an **OptionButton** inside a group box to select one or more of a group of related items. For example, you can create an order form with a list of available items, with an **OptionButton** preceding each item. The user can select a particular item by checking the corresponding **OptionButton**.

The default property for an **OptionButton** is the <u>Value</u> property.

Pages Collection

A **Pages** collection includes all the pages of a **MultiPage**.

Remarks

Each **Pages** collection provides the features to manage the number of pages in the collection and to identify the page that is currently in use.

The default value of the **Pages** collection identifies the current **Page** of a collection.

You can reference a **Page** by its index value. The index value reflects the ordinal position of the **Page** within the collection. The index of the first **Page** in a collection is 0; the index of the second **Page** is 1; and so on.

Page Object

One page of a **MultiPage** or a single member of a **Pages** collection.

Remarks

Each **Page** object contains its own set of controls and does not necessarily rely on other pages in the collection for information. A **Page** inherits some properties from its container; the value of each inherited property is set by the container.

You can reference a **Page** by its index value. The index value reflects the ordinal position of the **Page** within the collection. The index of the first **Page** in a collection is 0; the index of the second **Page** is 1; and so on.

The default name for the first **Page** is Page1. The default name for the second **Page** is Page2.

ScrollBar control

Returns or sets the value of another control based on the position of the scroll box.

Remarks

Requires VBScript.

A **ScrollBar** is a stand-alone control you can place on a form. It is visually like the scroll bar you see in certain objects such as a **ListBox** or the drop-down portion of a **ComboBox**. However, unlike the scroll bars in these controls, the stand-alone **ScrollBar** is not an integral part of any other control.

To use the **ScrollBar** to set or read the value of another control, you must write code that uses the **ScrollBar's Value** property. For example, to use the **ScrollBar** to update the value of a **TextBox**, you can write code that reads the **Value** property of the **ScrollBar** and then sets the **Value** property of the **TextBox**.

The default property for a **ScrollBar** is the <u>Value</u> property.

Note To create a horizontal or vertical **ScrollBar**, drag the sizing handles of the **ScrollBar** horizontally or vertically on the form.

SpinButton control

Increments and decrements numbers.

Remarks

Requires VBScript.

Clicking a **SpinButton** changes only the value of the **SpinButton**. You can write code that uses the **SpinButton** to update the displayed value of another control. For example, you can use a **SpinButton** to change the month, the day, or the year shown on a date. You can also use a **SpinButton** to scroll through a range of values or a list of items, or to change the value displayed in a text box.

To display a value updated by a **SpinButton**, you must assign the value of the **SpinButton** to the displayed portion of a control, such as the <u>Caption</u> property of a <u>Label</u> or the <u>Text</u> property of a <u>TextBox</u>. To create a horizontal or vertical **SpinButton**, drag the sizing handles of the **SpinButton** horizontally or vertically on the form.

The default property for a **SpinButton** is the <u>Value</u> property.

Tab Object

A **Tab** is an individual member of a **<u>Tabs</u>** collection.

Remarks

Visually, a **Tab** object appears as a rectangle protruding from a larger rectangular area, or as a button adjacent to a rectangular area.

In contrast to a **Page**, a **Tab** does not contain any controls. Controls that appear within the region bounded by a **TabStrip** are contained on the form, as is the **TabStrip**.

You can reference a **Tab** by its index value. The index value reflects the ordinal position of the **Tab** within the collection. The index of the first **Tab** in a collection is 0; the index of the second **Tab** is 1; and so on.

Tabs Collection

A **Tabs** collection includes all **<u>Tabs</u>** of a <u>**TabStrip**</u>.

Remarks

Each **Tabs** collection provides the features to manage the number of tabs in the collection and to identify the tab that is currently in use.

The default value of the **Tabs** collection identifies the current **Tab** of a collection.

You can reference a **Tab** by its index value. The index value reflects the ordinal position of the **Tab** within the collection. The index of the first **Tab** in a collection is 0; the index of the second **Tab** is 1; and so on.



TabStrip control

Presents a set of related controls as a visual group.

Remarks

You can use a **TabStrip** to view different sets of information for related controls.

A **TabStrip** is a control that contains a <u>collection</u> of one or more tabs.

Each <u>**Tab**</u> of a **TabStrip** is a separate object that users can select. Visually, a **TabStrip** also includes a client area that all the tabs in the **TabStrip** share.

By default, a **TabStrip** includes two pages, called Tab1 and Tab2. Each of these is a **Tab** object, and together they represent the **Tabs** collection of the **TabStrip**. If you add more pages, they become part of the same **Tabs** collection.

For example, the controls might represent information about a daily schedule for a group of individuals, with each set of information corresponding to a different individual in the group. Set the title of each tab to show one individual's name. Then, you can write code that, after you click a tab, updates the controls to show information about the person identified on the tab.

Note The **TabStrip** is implemented as a <u>container</u> of a **Tabs** collection, which in turn contains a group of **Tab** objects. The **TabStrip** control does not support the <u>Click</u> event.

Note The default property for a **TabStrip** is the <u>SelectedItem</u> property.



TextBox control

Displays information from a user or from an organized set of data.

Remarks

A **TextBox** is the control most commonly used to display information entered by a user. Also, it can display a set of data, such as a table, query, worksheet, or a calculation result. If a **TextBox** is <u>bound</u> to a <u>data source</u>, then changing the contents of the **TextBox** also changes the value of the bound data source.

Formatting applied to any piece of text in a **TextBox** will affect all text in the control. For example, if you change the font or point size of any character in the control, the change will affect all characters in the control.

The default property for a **TextBox** is the <u>Value</u> property.

Tips on using text boxes

The **TextBox** is a flexible control governed by the following properties: **Text**, **MultiLine**, **WordWrap**, and **AutoSize**.

Text contains the text that's displayed in the text box.

MultiLine controls whether the **TextBox** can display text as a single line or as multiple lines. Newline characters identify where one line ends and another begins. If **MultiLine** is **False** (the default value), the text is truncated instead of wrapped.

WordWrap allows the **TextBox** to wrap lines of text that are longer than the width of the **TextBox** into shorter lines that fit. The default value is **True**.

If you do not use **WordWrap**, the **TextBox** starts a new line of text when it encounters a newline character in the text. If **WordWrap** is turned off, you can have text lines that do not fit completely in the **TextBox**. The **TextBox** displays the portions of text that fit inside its width and truncates the portions of text that do not fit. **WordWrap** is not applicable unless **MultiLine** is **True**. **AutoSize** controls whether the **TextBox** adjusts to display all of the text. When using **AutoSize** with a **TextBox**, the width of the **TextBox** shrinks or expands according to the amount of text in the **TextBox** and the font size used to display the text. The default value is **False**.

AutoSize works well in the following situations:

- Displaying a caption of one or more lines.
- Displaying the contents of a single-line **TextBox**.
- Displaying the contents of a multiline **TextBox** that is read-only to the user.

Note Avoid using **AutoSize** with an empty **TextBox** that also uses the **MultiLine** and **WordWrap** properties. When the user enters text into a **TextBox** with these properties, the **TextBox** automatically sizes to a long narrow box one character wide and as long as the line of text.



ToggleButton control

Shows the selection state of an item.

Remarks

Use a **ToggleButton** to show whether an item is selected. If a **ToggleButton** is bound to a data source, the **ToggleButton** shows the current value of that data source as either *Yes/No*, *True/False*, *On/Off*, or some other choice of two settings. If the user selects the **ToggleButton**, the current setting is *Yes*, *True*, or *On*. If the user does not select the **ToggleButton**, the setting is *No*, *False*, or *Off*. If the **ToggleButton** is bound to a data source, changing the setting changes the value of that data source. A disabled **ToggleButton** shows a value, but is dimmed and does not allow changes from the user interface.

You can also use a **ToggleButton** inside a **Frame** to select one or more of a group of related items. For example, you can create an order form with a list of available items, with a **ToggleButton** preceding each item. The user can select a particular item by selecting the appropriate **ToggleButton**.

The default property of a **ToggleButton** is the <u>Value</u> property.

The only event for a **ToggleButton** is the <u>Click</u> event.



About the order of events

The following events occur in the order specified when a user completes an action.

Events	When	
Open	A form is opened to compose an <i>item</i> .	
Send, Write, Close	An item is sent.	
Write, Close	An item is posted.	
Write	An item is saved.	
Close	An item is closed.	
Read, Open	An item is opened in a folder.	
Reply	A user replies to an item's sender.	
ReplyAll	A user replies to an item's sender and all recipients.	
Forward	The newly-created item is passed to the procedure after the user selects the Forward action for an item.	
PropertyChange	One of the item's standard properties is changed.	
CustomPropertyChange One of the item's custom properties is changed.		
CustomAction	A user-defined action is initiated.	

The <u>Click</u> event occurs only when you have defined it for a control in the Script Editor.

Add a Click event for a control

1. Open the Script Editor.

How?

2. For the control to which you want to add a <u>Click</u> event, add the following, where CommandButton1 is the name of the control.

Sub CommandButton1_Click

End Sub

3. Enter the code that you want in the event.



Click Event

The **Click** event occurs only if it has been defined for a control in the Script Editor.

The <u>**TabStrip</u>** and <u>**MultiPage**</u> controls do not support the **Click** event. However, individual the <u>**Page**</u> objects of the **MultiPage** control do support the click event.</u>

<u>ScrollBars</u> and **<u>SpinButtons</u>** do not support the **Click** event but you can <u>bind</u> them to fields and use the **CustomPropertyChange** event.

Syntax

```
Sub object_Click( )
```

The **Click** event syntax has these parts:

Part Description

object Required. A valid object.

Example:

```
Sub CommandButton1_Click()
    MsgBox "You just clicked my button! "
End Sub
```

<u>Remarks</u>

The **Click** event occurs in one of two cases:

- The user clicks a control with the mouse.
- The user definitively selects a value for a control with more than one possible value.

Of the two cases where the **Click** event occurs, the first case applies to the **<u>CommandButton</u>**, **<u>Frame</u>**, **<u>Image</u>**, **<u>Label</u>**, and **Page**.

The second case applies to the <u>CheckBox</u>, <u>ComboBox</u>, <u>ListBox</u>, and <u>ToggleButton</u>. It also applies to an <u>OptionButton</u> when the value changes to **True**.

The following are examples of actions that initiate the **Click** event:

- Clicking a blank area of a form or a disabled control (other than a list box) on the form.
- Clicking a **CommandButton**.
- Pressing the SPACEBAR when a **CommandButton** has the focus.
- Clicking a control with the left mouse button (left-clicking).
- Pressing ENTER on a form that has a command button whose **Default** property is set to **True**, as long as no other command button has the focus.
- Pressing ESC on a form that has a command button whose **Cancel** property is set to **True**, as long as no other command button has the focus.
- Pressing a control's <u>accelerator key</u>.

For some controls, the **Click** event occurs when the **Value** property changes. However, using the **PropertyChange** or **CustomPropertyChange** event is the preferred technique for detecting a new value for a property. The following are examples of actions that initiate the **Click** event due to assigning a new value to a control:

- Clicking a **CheckBox** or **ToggleButton**, pressing the SPACEBAR when one of these controls has the focus, pressing the accelerator key for one of these controls, or changing the value of the control in code.
- Changing the value of an OptionButton to True. Setting one
 OptionButton in a group to True sets all other buttons in the group to
 False, but the Click event occurs only for the button whose value changes to True.
- Selecting a value for a **ComboBox** or **ListBox** so that it unquestionably matches an item in the control's drop-down list. For example, if a list is not sorted, the first match for characters typed in the edit region may not be the only match in the list, so choosing such a value does not initiate the **Click** event. In a sorted list, you can use entry-matching to ensure that a selected value is a unique match for text the user types.

The **Click** event is not initiated when **Value** is set to **Null**.

Note Left-clicking changes the value of a control, thus it initiates the **Click** event. Right-clicking does not change the value of the control, so it does not initiate the **Click** event.

Also Note If you bind a **ListBox**, **ComboBox**, **OptionButton**, or **CheckBox** to a field, then the **Click** event does not fire. You need to use the **PropertyChange** or **CustomPropertyChange** event to detect the change via code.

Example:

```
Sub Item_PropertyChange(ByVal Name)
Set MyListBox = Item.GetInspector.ModifiedFormPages("Message").Contr
Select Case Name
    Case "Mileage"
        Item.CC = MyListBox.Value
        Item.Subject = MyListBox.Value
        Case Else
End Select
End Sub
```



Add Method

Adds a control by its programmatic identifier (*ProgID*) to a page or adds or inserts a **Tab** or **Page** in a **TabStrip** or **MultiPage**.

Syntax

For **MultiPage**, **TabStrip**

Set Object = object.Add([Name [, Caption [, index]]])

For other controls
Set Control = object.Add(ProgID [, Name [, Visible]])

The **Add** method syntax has these parts:

Part	Description
object	Required. A valid object name.
Name	Optional. Specifies the name of the object being added. If a name is not specified, the system generates a default name based on the rules of the application where the form is used.
Caption	Optional. Specifies the caption to appear on a tab or a control. If a caption is not specified, the system generates a default caption based on the rules of the application where the form is used.
index	Optional. Identifies the position of a page or tab within a Pages or Tabs collection. If an index is not specified, the system appends the page or tab to the end of the Pages or Tabs collection and assigns the appropriate index value.
ProgID	Required. Programmatic identifier. A text string with no spaces that identifies an object class. The standard syntax for a <i>ProgID</i> is <vendor>.<component>.<version>. A <i>ProgID</i> is mapped to a <u>class</u> identifier (CLSID).</version></component></vendor>
Visible	Optional. True if the object is visible (default). False if the object is hidden.

Settings

ProgID values for individual controls are:

<u>CheckBox</u>	Forms.CheckBox.1
<u>ComboBox</u>	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
<u>Label</u>	Forms.Label.1
<u>ListBox</u>	Forms.ListBox.1
<u>MultiPage</u>	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
<u>ScrollBar</u>	Forms.ScrollBar.1
<u>SpinButton</u>	Forms.SpinButton.1
<u>TabStrip</u>	Forms.TabStrip.1
<u>TextBox</u>	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Remarks

For a **MultiPage** control, the **Add** method returns a **Page** object. For a **TabStrip**, it returns a **Tab** object. The index value for the first **Page** or **Tab** of a <u>collection</u> is 0, the value for the second **Page** or **Tab** is 1, and so on.

For the **Controls** collection of an object, the **Add** method returns a control corresponding to the specified *ProgID*. The AddControl event occurs after the control is added.

The following syntax will return the **<u>Text</u>** property of a control added at design time:

userform1.thebox.text

If you add a control at run time, you must use the exclamation syntax to reference properties of that control. For example, to return the **Text** property of a control added at run time, use the following syntax:

userform1!thebox.text

Note You can change a control's **Name** property at <u>run time</u> only if you added that control at run time with the **Add** method.



AddItem Method

For a single-column **ListBox** or **ComboBox**, the **AddItem** method adds an item to the list. For a multicolumn **ListBox** or **ComboBox**, this method adds a row to the list.

Syntax

Variant = object.AddItem([item [, varIndex]])

The **AddItem** method syntax has these parts:

Part	Description	
object	Required. A valid object.	
Item	Optional. Specifies the item or row to add. The number of the first item or row is 0; the number of the second item or row is 1, and so on.	
varIndex	Optional. Integer specifying the position within the object where the new item or row is placed.	

Remarks

If you supply a valid value for *varIndex*, the **AddItem** method places the item or row at that position within the list. If you omit *varIndex*, the method adds the item or row at the end of the list.

The value of *varIndex* must not be greater than the value of the ListCount property.

For a multicolumn **ListBox** or **ComboBox**, **AddItem** inserts an entire row, that is, it inserts an item for each column of the control. To assign values to an item beyond the first column, use the **List** or **Column** property and specify the row and column of the item.

If the control is **bound** to data, the **AddItem** method fails.

Note You can add more than one row at a time to a **ComboBox** or **ListBox** by using **List**.



Clear Method

Removes all objects from an object or <u>collection</u>.

Syntax

object.Clear

The **Clear** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

For a **MultiPage** or **TabStrip**, the **Clear** method deletes individual pages or tabs.

For a **ListBox** or **ComboBox**, **Clear** removes all entries in the list.

For a **Controls** collection, **Clear** deletes controls that were created at <u>run-time</u> with the <u>Add</u> method. Using **Clear** on controls created at <u>design-time</u> causes an error.

Note If the control is <u>bound</u> to data, the **Clear** method fails.

Copy Method

Copies the contents of an object to the Clipboard.

Syntax

object.Copy

The **Copy** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The original content remains on the object.

The actual content that is copied depends on the object. For example, on a **Page**, the **Copy** method copies the currently selected control or controls. On a **TextBox** or **ComboBox**, it copies the currently selected text.

Using **Copy** for a form, **Frame**, or **Page** copies the currently-active control.



Cut Method

Removes selected information from an object and transfers it to the Clipboard.

Syntax

object.Cut

The **Cut** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

For a **ComboBox** or **TextBox**, the **Cut** method removes currently selected text in the control to the Clipboard. This method does not require that the control have the <u>focus</u>.

On a <u>Page</u>, <u>Frame</u>, or form, **Cut** removes currently selected controls to the Clipboard. This method only removes controls created at <u>run time</u>.

DropDown Method

Displays the list portion of a **<u>ComboBox</u>**.

Syntax

object.DropDown

The **DropDown** method syntax has these parts:

PartDescriptionobject Required. A valid object.

Remarks

Use the **DropDown** method to open the list in a combo box.



GetFormat Method

Returns an integer value indicating whether a specific <u>format</u> is on the **DataObject**.

Syntax

Boolean = object.GetFormat(format)

The **GetFormat** method syntax has these parts:

Part

Description

object Required. A valid object.

Required. An integer or string specifying a specific format that might *format* exist in the **DataObject**. If the specified format exists in the **DataObject**, **GetFormat** returns **True**.

Settings

The settings for *format* are:

Value	Description
1	Text format.
A string or any integer	A user-defined DataObject format passed to the
other than 1	DataObject from <u>SetText</u> .

Remarks

The **GetFormat** method searches for a format in the current list of formats on the **DataObject**. If the format is on the **DataObject**, **GetFormat** returns **True**; if not, **GetFormat** returns **False**.

The **DataObject** currently supports only text formats.

GetFromClipboard Method

Copies data from the Clipboard to a **DataObject**.

Syntax

String = object.GetFromClipboard()

The **GetFromClipboard** method syntax has these parts:

PartDescriptionobject Required. A valid object name.

Remarks

The **DataObject** can contain multiple data items, but each item must be in a different format. For example, the **DataObject** might include one text item and one item in a custom format; but cannot include two text items.



GetText Method

Retrieves a text string from the **DataObject** using the specified <u>format</u>.

Syntax

String = object.GetText([format])

The **GetText** method syntax has these parts:

Part

Description

object Required. A valid object name.

format Optional. A string or integer specifying the format of the data to retrieve from the **DataObject**.

Settings

The settings for *format* are:

Value	Description
1	Text format.
A string or any integer other than 1	A user-defined DataObject format passed to the DataObject from <u>SetText</u> .

Remarks

The **DataObject** supports multiple formats, but only supports one data item of each format. For example, the **DataObject** might include one text item and one item in a custom format; but cannot include two text items.

If no format is specified, the **GetText** method requests information in the Text format from the **DataObject**.



Item Method

Returns a member of a <u>collection</u>, either by position or by name.

Syntax

Set Object = object.Item(collectionindex)

The **Item** method syntax has these parts:

PartDescriptionobjectRequired. A valid object.collectionindexRequired. A member's position, or index, within a collection.

Settings

The *collectionindex* can be either a string or an integer. If it is a string, it must be a valid member name. If it is an integer, the minimum value is 0 and the maximum value is one less than the number of items in the collection.

Remarks

If an invalid index or name is specified, an error occurs.

Move Method

Moves a form or control, or moves all the controls in the **Controls** collection..

Syntax

For a form or control *object*.**Move(** [*Left* [, *Top* [, *Width* [, *Height* [, *Layout*]]]]**)**

For the Controls collection *object*.**Move(** X, Y)

The **Move** method syntax has these parts:

Part	Description
object	Required. A valid object name.
Left	Optional. Single-precision value, in points, indicating the horizontal coordinate for the left edge of the object.
Тор	Optional. Single-precision value, in points, that specifies the vertical coordinate for the top edge of the object.
Width	Optional. Single-precision value, in points, indicating the width of the object.
Height	Optional. Single-precision value, in points, indicating the height of the object.
Layout	Optional. A Boolean value indicating whether the Layout event is initiated for the control's parent following this move. False is the default value.
Х, Ү	Required. Single-precision value, in points, that specifies the change from the current horizontal and vertical position for each control in the Controls collection.

Settings

The maximum and minimum values for the *Left*, *Top*, *Width*, *Height*, *X*, and *Y* arguments vary from one application to another.

Remarks

For a form or control, you can move a selection to a specific location relative to the edges of the form that contains the selection.

You can use named arguments, or you can enter the arguments by position. If you use named arguments, you can list the arguments in any order. If not, you must enter the arguments in the order shown, using commas to indicate the relative position of arguments you do not specify. Any unspecified arguments remain unchanged.

For the **Controls** collection, you can move all the controls in this collection a specific distance from their current positions on a **Frame** or **Page**.

Paste Method

Transfers the contents of the Clipboard to an object.

Syntax

object.Paste

The **Paste** method syntax has these parts:

PartDescriptionobject Required. A valid object.

Remarks

Data pasted into a **<u>ComboBox</u>** or **<u>TextBox</u>** is treated as text.

PutInClipboard Method

Moves data from a **DataObject** to the Clipboard.

Syntax

object.PutInClipboard

The **PutInClipboard** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **PutInClipboard** method replaces the contents of the Clipboard with the contents of the **DataObject** that is in Text format.

RedoAction Method

Reverses the effect of the most recent Undo action.

Syntax

Boolean = object.**RedoAction**

The **RedoAction** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

Redo reverses the last Undo, which is not necessarily the last action taken. Not all actions can be undone.

RedoAction returns **True** if it was successful.



Remove Method

Removes a member from a <u>collection</u>; or, removes a control from a <u>Frame</u>, or <u>Page</u>.

Syntax

object.Remove(collectionindex)

The **Remove** method syntax has these parts:

Part	Description
object	Required. A valid object.
collectionindex	Required. A member's position, or index, within a collection. Numeric as well as string values are acceptable. If the value is a number, the minimum value is zero, and the maximum value is one less than the number of members in the collection. If the value is a string, it must correspond to a valid member name.



RemoveItem Method

Removes a row from the list in a list box or combo box.

Syntax

Boolean = object.RemoveItem(index)

The **RemoveItem** method syntax has these parts:

Part

Description

object Required. A valid object.

index Required. Specifies the row to delete. The number of the first row is 0; the number of the second row is 1, and so on.

This method will not remove a row from the list if the **ListBox** is data <u>bound</u> (that is, when the **RowSource** property specifies a <u>data source</u> for the **ListBox**).

Repaint Method

Updates the display by redrawing the frame or page.

Syntax

Boolean = object.**Repaint**

The **Repaint** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **Repaint** method is useful if the contents or appearance of an object changes significantly, and you don't want to wait until the system automatically repaints the area.

Scroll Method

Moves the scroll bar on an object.

Syntax

object.Scroll([ActionX [, ActionY]])

The **Scroll** method syntax has these parts:

Part

Description

object Required. A valid object name.

ActionX Optional. Identifies the action to occur in the horizontal direction. *ActionY* Optional. Identifies the action to occur in the vertical direction.

Settings

The settings for *ActionX* and *ActionY* are:

Value

Description

0 Do not scroll in the specified direction.

Move up on a vertical scroll bar or left on a horizontal scroll bar.

1 Movement is equivalent to pressing the up or left arrow key on the keyboard to move the scroll bar.

Move down on a vertical scroll bar or right on a horizontal scroll bar.

2 Movement is equivalent to pressing the right or down arrow key on the keyboard to move the scroll bar.

Move one pageup on a vertical scroll bar or one page left on a horizontal

3 scroll bar. Movement is equivalent to pressing PAGE UP on the keyboard to move the scroll bar.

Move one pagedown on a vertical scroll bar or one page right on a

- 4 horizontal scroll bar. Movement is equivalent to pressing PAGE DOWN on the keyboard to move the scroll bar.
- 5 Move to the top of a vertical scroll bar or to the left end of a horizontal scroll bar.

6 Move to the bottom of a vertical scroll bar or to the right end of a horizontal scroll bar.

Remarks

The **Scroll** method applies scroll bars that appear on a form, **Frame**, or **Page** that is larger than its display area. This method does not apply to the stand-alone **ScrollBar** or to scroll bars that appear on a **TextBox**.



SetDefaultTabOrder Method

Sets the **<u>TabIndex</u>** property of each control on a frame or page, using a default top-to-bottom, left-to-right <u>tab order</u>.

Syntax

object.SetDefaultTabOrder

The **SetDefaultTabOrder** method syntax has these parts:

Part Description

object Required. A valid object.



SetFocus Method

Moves the <u>focus</u> to this instance of an object.

Syntax

object.SetFocus

The **SetFocus** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

If setting the focus fails, the focus reverts to the previous object and an error is generated.

By default, setting the focus to a control does not activate the control's window or place it on top of other controls.

The **SetFocus** method is valid for an empty **Frame** as well as a **Frame** that contains other controls. An empty **Frame** will take the focus itself, and any subsequent keyboard events apply to the **Frame**. In a **Frame** that contains other controls, the focus moves to the first control in the **Frame**, and subsequent keyboard events apply to the control that has the focus.



SetText Method

Copies a text string to the **DataObject** using a specified <u>format</u>.

Syntax

object.SetText(StoreData [, format])

The **SetText** method syntax has these parts:

Part	Description
object	Required. A valid object.
StoreDate	a Required. Defines the data to store on the DataObject .
format	Optional. An integer or string specifying the format of <i>StoreData</i> . When retrieving data from the DataObject , the format identifies the piece of data to retrieve.

Settings

The settings for *format* are:

ValueDescription1Text format.A string or integer value other than 1 A user-defined DataObject format.

Remarks

The **DataObject** stores data according to its format. When the user supplies a string, the **DataObject** saves the text under the specified format.

If the **DataObject** contains data in the same format as new data, the new data replaces the existing data in the **DataObject**. If the new data is in a new format, the new data and the new format are both added to the **DataObject**, and the previously existing data is there as well.

If no format is specified, the **SetText** method assigns the Text format to the text string. If a new format is specified, the **DataObject** registers the new format with the system.

StartDrag Method

Initiates a drag-and-drop operation for a **DataObject**.

Syntax

DropEffect=Object.StartDrag([Effect as DropEffect])

The **StartDrag** method syntax has these parts:

Part

Description

Object Required. A valid object. *Effect* Optional. Effect of the drop operation on the target control.

Settings

The settings for *Effect* are:

Value

Description

- 0 Does not copy or move the drop source to the drop target.
- 1 Copies the drop source to the drop target.
- 2 Moves the drop source to the drop target.
- 3 Copies or moves the drop source to the drop target.

Remarks

The drag action starts at the current mouse pointer position with the current keyboard state and ends when the user releases the mouse. The effect of the drag-and-drop operation depends on the effect chosen for the drop target.

For example, a control's MouseMove event might include the **StartDrag** method. When the user clicks the control and moves the mouse, the mouse pointer changes to indicate whether *Effect* is valid for the drop target.

UndoAction Method

Reverses the most recent action that supports the Undo command.

Syntax

Boolean = object.**UndoAction**

The **UndoAction** method syntax has these parts:

Part Description

object Required. A valid object.

Remarks

Not all user actions can be undone. If an action cannot be undone, the Undo command is unavailable following the action.

You must apply this method before the form or control is updated. You may want to include this method in a form's PropertyChange event.



ZOrder Method

Places the object at the front or back of the <u>z-order</u>.

Syntax

object.ZOrder([zPosition])

The **ZOrder** method syntax has these parts:

Part

Description

object Required. A valid object.

zPosition Optional. A control's position, front or back, in the container's z-order.

Settings

The settings for *zPosition* are:

Value

Description

- 0 Places the control at the front of the z-order. The control appears on top of other controls (default).
- 1 Places the control at the back of the z-order. The control appears underneath other controls.

Remarks

The z-order determines how windows and controls are stacked when they are presented to the user. Items at the back of the z-order are overlaid by closer items; items at the front of the z-order appear to be on top of items at the back. When the *zPosition* argument is omitted, the object is brought to the front.

In <u>design mode</u>, the Bring to Front or Send To Back commands set the z-order. Bring to Front is equivalent to using the **ZOrder** method and putting the object at the front of the z-order. Send to Back is equivalent to using **ZOrder** and putting the object at the back of the z-order. This method does not affect content or sequence of the controls in the **Controls** collection.

Note You can't Undo or Redo layering commands, such as **Send to Back** or **Bring to Front**. For example, if you select an object and click **Move Backward** on the shortcut menu, you won't be able to Undo or Redo that action.



Accelerator Property

Sets or retrieves the <u>accelerator key</u> for a control.

Syntax

object.Accelerator [= String]

The **Accelerator** property syntax has these parts:

PartDescriptionobject Required. A valid object.String Optional. The character to use as the accelerator key.

Remarks

To designate an accelerator key, enter a single character for the **Accelerator** property. You can set **Accelerator** in the control's property sheet or in code. If the value of this property contains more than one character, the first character in the string becomes the value of **Accelerator**.

When an accelerator key is used, there is no visual feedback (other than <u>focus</u>) to indicate that the control initiated the <u>Click</u> event. For example, if the accelerator key applies to a <u>CommandButton</u>, the user will not see the button pressed in the interface. The button receives the focus, however, when the user presses the accelerator key.

ActiveControl Property

Identifies and allows manipulation of the control that has the focus.

Syntax

object.ActiveControl

The **ActiveControl** property syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **ActiveControl** property is read-only and is set when you select a control in the interface. You can use **ActiveControl** as a substitute for the control name when setting properties or calling methods.

Alignment Property

Specifies the position of a control relative to its caption.

Syntax

object.Alignment [= 0,1]

The **Alignment** property syntax has these parts:

Part Description

object Required. A valid object.

Settings

The Settings for Alignment are:

Value

Description

- 0 Places the caption to the left of the control.
- 1 Places the caption to the right of the control (default).

Remarks

The caption text for a control is left-aligned.

AutoSize Property

Specifies whether an object automatically resizes to display its entire contents.

Syntax

object.AutoSize [= Boolean]

The **AutoSize** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control is resized.

Settings

The settings for *Boolean* are:

Value Description

True Automatically resizes the control to display its entire contents.

False Keeps the size of the control constant. Contents are clipped when they exceed the area of the control (default).

Remarks

For controls with captions, the **AutoSize** property specifies whether the control automatically adjusts to display the entire caption.

For controls without captions, this property specifies whether the control automatically adjusts to display the information stored in the control. In a **ComboBox**, for example, setting **AutoSize** to **True** automatically sets the width of the display area to match the length of the current text.

For a single-line **<u>TextBox</u>**, setting **AutoSize** to **True** automatically sets the width of the display area to the length of the text in the text box.

For a multiline **TextBox** that contains no text, setting **AutoSize** to **True** automatically displays the text as a column. The width of the text column is set to accommodate the widest letter of that font size. The height of the text column is set to display the entire text of the **TextBox**.

For a multiline **TextBox** that contains text, setting **AutoSize** to **True** automatically enlarges the **TextBox** vertically to display the entire text. The width of the **TextBox** does not change.

Note If you manually change the size of a control while **AutoSize** is **True**, the manual change overrides the size previously set by **AutoSize**.



AutoTab Property

Specifies whether an automatic tab occurs when a user enters the maximum allowable number of characters into a **TextBox** or the text box portion of a **ComboBox**.

Syntax

object.AutoTab [= Boolean]

The **AutoTab** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Specifies whether an automatic tab occurs.

Settings

The settings for Boolean are:

Value Description

True Tab occurs.False Tab does not occur (default).

Remarks

The **MaxLength** property specifies the maximum number of characters allowed in a **TextBox** or the text box portion of a **ComboBox**.

You can specify the **AutoTab** property for a **TextBox** or **ComboBox** on a form for which you usually enter a set number of characters. Once a user enters the maximum number of characters, the <u>focus</u> automatically moves to the next control in the <u>tab order</u>. For example, if a **TextBox** displays inventory stock numbers that are always five characters long, you can use **MaxLength** to specify the maximum number of characters to enter into the **TextBox** and **AutoTab** to automatically tab to the next control after the user enters five characters.

AutoWordSelect Property

Specifies whether the basic unit used to extend a selection is a word or a single character.

Syntax

object.AutoWordSelect [= Boolean]

The **AutoWordSelect** property syntax has these parts:

Part Description

object Required. A valid object. *Boolean* Optional. Specifies the basic unit used to extend a selection.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueUses a word as the basic unit (default).FalseUses a character as the basic unit.

Remarks

The **AutoWordSelect** property specifies how the selection extends or contracts in the edit region of a **TextBox** or **ComboBox**.

If the user places the insertion point in the middle of a word and then extends the selection while **AutoWordSelect** is **True**, the selection includes the entire word.



BackColor Property

Specifies the <u>background color</u> of the object.

Syntax

object.BackColor [= Long]

The **BackColor** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. A value or constant that determines the background color of an object.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the **RGB** function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75, as shown in the following example.

RGB(15,200,75)

Remarks

You can only see the background color of an object if the **<u>BackStyle</u>** property is set to **1**.



BackStyle Property

Returns or sets the background style for an object.

Syntax

object.BackStyle [=BackStyle]

The **BackStyle** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.BackStyle Optional. Specifies the control background.

Settings

The settings for *BackStyle* are:

Value Description

- 0 The background is transparent.
- 1 The background is opaque (default).

Remarks

The **BackStyle** property determines whether a control is <u>transparent</u>. If **BackStyle** is 1, the control is not transparent and you cannot see anything behind the control on a form. If **BackStyle** is 0, you can see through the control and look at anything on the form located behind the control. The <u>BackColor</u> property is only valid if the **BackStyle** property is set to 1.

Note does not affect the transparency of bitmaps. You must use a picture editor such as Paintbrush to make a bitmap transparent. Not all controls support transparent bitmaps.



Bold, Italic, Size, StrikeThrough, Underline, Weight Properties

Specifies the visual attributes of text on a displayed or printed form.

Syntax

object.Bold [= Boolean]

object.Italic [= Boolean]

object.Size [= Currency]

object.StrikeThrough [= Boolean]

object.Underline [= Boolean]

object.Weight [= Integer]

The **Bold**, **Italic**, **Size**, **StrikeThrough**, **Underline**, and **Weight** property syntaxes have these parts:

PartDescriptionobjectRequired. A valid object name.BooleanOptional. Specifies the font style.CurrencyOptional. A number indicating the font size.IntegerOptional. Specifies the font style.

The settings for *Boolean* are:

Value

Description

True The text has the specified attribute (that is bold, italic, size, strikethrough or underline marks, or weight).

False The text does *not* have the specified attribute (default).

The **Weight** property accepts values from 0 to 1000. A value of zero allows the system to pick the most appropriate weight. A value from 1 to 1000 indicates a specific weight, where 1 represents the lightest type and 1000 represents the darkest type.

Remarks

These properties define the visual characteristics of text. The **Bold** property determines whether text is normal or bold. The **Italic** property determines whether text is normal or italic. The **Size** property determines the height, in points, of displayed text. The **Underline** property determines whether text is underlined. The **StrikeThrough** property determines whether the text appears with strikethrough marks. The **Weight** property determines the darkness of the type.

The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display with the specified attribute or that isn't installed, Windows substitutes a similar font. The substitute font will be as similar as possible to the font originally requested.

Changing the value of **Bold** also changes the value of **Weight**. Setting **Bold** to **True** sets **Weight** to 700; setting **Bold** to **False** sets **Weight** to 400. Conversely, setting **Weight** to anything over 550 sets **Bold** to **True**; setting **Weight** to 550 or less sets **Bold** to **False**.

The default point size is determined by the operating system.



BorderColor Property

Specifies the color of an object's border.

Syntax

```
object.BorderColor [= Long]
```

The **BorderColor** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. A value that determines the border color of an object.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the **RGB** function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75, as shown in the following example.

RGB(15,200,75)

Remarks

To use the **BorderColor** property, the **<u>BorderStyle</u>** property must be set to a value other than 0.

BorderStyle uses **BorderColor** to define the border colors. The **SpecialEffect** property uses <u>system colors</u> exclusively to define its border colors. For Windows operating systems, system color settings are set using the **Display** icon in Control Panel. In Windows NT 3.51, system color settings are set using the **Color** icon in Control Panel.

BorderStyle Property

Specifies the type of border used by a control or a form.

Syntax

object.BorderStyle [=BorderStyle]

The **BorderStyle** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.BorderStyleOptional. Specifies the border style.

Settings

The settings for *BorderStyle* are:

Value	e Description
0	The control has no visible border line.
1	The control has a single line harder (defer

1 The control has a single-line border (default).

The default value for a <u>**ComboBox**</u>, <u>**Frame**</u>, <u>**Label**</u>, <u>**ListBox**</u> or <u>**TextBox**</u> is 0 (*None*). The default value for an <u>**Image**</u> is 1 (*Single*).

Remarks

For a **Frame**, the **BorderStyle** property is ignored if the **SpecialEffect** property is *None*.

You can use either **BorderStyle** or **SpecialEffect** to specify the border for a control, but not both. If you specify a nonzero value for one of these properties, the system sets the value of the other property to zero. For example, if you set **BorderStyle** to 1, the system sets **SpecialEffect** to zero (*Flat*). If you specify a nonzero value for **SpecialEffect**, the system sets **BorderStyle** to zero.

BorderStyle uses **BorderColor** to define the colors of its borders. To use the **BorderColor** property, the **BorderStyle** property must be set to a value other than 0.



BoundColumn Property

Identifies the source of data in a multicolumn **<u>ComboBox</u>** or **<u>ListBox</u>**.

Syntax

object.BoundColumn [= Variant]

The **BoundColumn** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Variant Optional. Indicates how the BoundColumn value is selected.

Settings

The settings for *Variant* are:

Value

Description

0 Assigns the value of the **ListIndex** property to the control.

1 or Assigns the value from the specified column to the control. Columns are greater numbered from 1 when using this property (default).

Remarks

When the user chooses a row in a multicolumn **ListBox** or **ComboBox**, the **BoundColumn** property identifies which item from that row to store as the value of the control. For example, if each row contains 8 items and **BoundColumn** is 3, the system stores the information in the third column of the currently-selected row as the value of the object.

You can display one set of data to users but store different, associated values for the object by using the **BoundColumn** and the **TextColumn** properties. **TextColumn** identifies the column of data displayed in a **ComboBox** or **ListBox; BoundColumn** identifies the column of associated data values stored

for the control. For example, you could set up a multicolumn **ListBox** that contains the names of holidays in one column and dates for the holidays in a second column. To present the holiday names to users, specify the first column as the **TextColumn**. To store the dates of the holidays, specify the second column as the **BoundColumn**.

If the control is <u>bound</u> to a <u>data source</u>, the value in the column specified by **BoundColumn** is stored in the data source named in the **ControlSource** property.

The **ListIndex** value retrieves the number of the selected row. For example, if you want to know the row of the selected item, set **BoundColumn** to 0 to assign the number of the selected row as the value of the control. Be sure to retrieve a current value, rather than relying on a previously saved value, if you are referencing a list whose contents might change.

The **Column**, **List**, and **ListIndex** properties all use zero-based numbering. That is, the value of the first item (column or row) is zero; the value of the second item is one, and so on. This means that if **BoundColumn** is set to 3, you could access the value stored in that column using the expression Column(2).



Cancel Property

Returns or sets a value indicating whether a <u>**CommandButton**</u> is the Cancel button on a form.

Syntax

object.Cancel [= Boolean]

The **Cancel** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the object is the Cancel button.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueThe CommandButton is the Cancel button.FalseThe CommandButton is not the Cancel button (default).

Remarks

A **CommandButton** or an object that acts like a command button can be designated as the default command button. For <u>OLE container controls</u>, the **Cancel** property is provided only for those objects that specifically behave as command buttons.

Only one **CommandButton** on a form can be the Cancel button. Setting **Cancel** to **True** for one command button automatically sets it to **False** for all other objects on the form. When a **CommandButton's Cancel** property is set to **True** and the form is the active form, the user can choose the command button by clicking it, pressing ESC, or pressing ENTER when the button has the <u>focus</u>.

A typical use of **Cancel** is to give the user the option of canceling uncommitted changes and returning the form to its previous state.

You should consider making the Cancel button the default button for forms that support operations that can't be undone (such as delete). To do this, set both **Cancel** and the **Default** property to **True**.

CanPaste Property

Specifies whether the Clipboard contains data that the object supports.

Syntax

object.CanPaste

The **CanPaste** property syntax has these parts:

Part Description

object Required. A valid object.

Return Values

The **CanPaste** property return values are:

ValueDescriptionTrueThe object can receive information pasted from the Clipboard.FalseThe object cannot receive information pasted from the Clipboard.

Remarks

CanPaste is read-only.

If the Clipboard data is in a format that the object does not support, the **CanPaste** property is **False**. For example, if you try to paste a bitmap into an object that only supports text, **CanPaste** will be **False**.

CanRedo Property

Indicates whether the most recent Undo can be reversed.

Syntax

object.CanRedo

The **CanRedo** property syntax has these parts:

PartDescriptionobject Required. A valid object.

Return Values

The **CanRedo** property syntax return values are:

ValueDescriptionTrueThe most recent Undo can be reversed.FalseThe most recent Undo is irreversible.

Remarks

CanRedo is read-only.

To Redo an action means to reverse an Undo; it does not necessrily mean to repeat the last user action.

CanUndo Property

Indicates whether the last user action can be undone.

Syntax

object.CanUndo

The **CanUndo** property syntax has these parts:

Part Description

object Required. A valid object.

Return Values

The **CanUndo** property syntax return values are:

ValueDescriptionTrueThe most recent user action can be undone.FalseThe most recent user action cannot be undone.

Remarks

CanUndo is read-only.

Many user actions can be undone with the **Undo** command. The **CanUndo** property indicates whether the most recent action can be undone.

Caption Property

Descriptive text that appears on an object to identify or describe it.

Syntax

object.Caption [= String]

The **Caption** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. A string expression that evaluates to the text displayed as the caption.

Settings

The default setting for a control is a unique name based on the type of control. For example, CommandButton1 is the default caption for the first command button in a form.

Remarks

The text identifies or describes the object with which it is associated. For buttons and labels, the **Caption** property specifies the text that appears in the control. For **Page** and **Tab** objects, it specifies the text that appears on the tab. For the **Explorer** object, it specifies the text that appears in the explorer's title bar.

If a control's caption is too long, the caption is truncated. If a form's caption is too long for the title bar, the title is displayed with an ellipsis.

The **ForeColor** property of the control determines the color of the text in the caption.

Tip If a control has both the **Caption** and **<u>AutoSize</u>** properties, setting **AutoSize** to **True** automatically adjusts the size of the control to frame the entire caption.



ClientHeight, ClientLeft, ClientTop, ClientWidth Properties

Define the dimensions and location of the display area of a **TabStrip**.

Syntax

object.ClientHeight [=Single]

object.ClientLeft [=Single]

object.ClientTop [=Single]

object.ClientWidth [=Single]

The **ClientHeight**, **ClientLeft**, **ClientTop**, and **ClientWidth** property syntaxes have these parts:

Part

Description

object Required. A valid object.

Optional. For **ClientHeight** and **ClientWidth**, specifies the height or Single width, in points, of the display area. For **ClientLeft** and **ClientTop**, specifies the distance, in points, from the top or left edge of the **TabStrip's** container.

Remarks

At <u>run time</u>, **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth** automatically store the coordinates and dimensions of the **TabStrip's** internal area, which is shared by objects in the **TabStrip**.



Column Property

Specifies one or more items in a **ListBox** or **ComboBox**.

Syntax

object.Column(column, row) [= Variant]

The **Column** property syntax has these parts:

Part	Description
0	Required. A valid object.
column	Optional. An integer with a range from 0 to one less than the total number of columns.
row	Optional. An integer with a range from 0 to one less than the total number of rows.
Variant	Optional. Specifies a single value, a column of values, or a two- dimensional array to load into a ListBox or ComboBox .

Settings

If you specify both the column and row values, **Column** reads or writes a specific item.

If you specify only the column value, the **Column** property reads or writes the specified column in the current row of the object. For example, MyListBox.Column (3) reads or writes the third column in MyListBox.

Column returns a *Variant* from the cursor. When a built-in <u>cursor</u> provides the value for *Variant* (such as when using the **AddItem** method), the value is a string. When an external cursor provides the value for *Variant*, formatting associated with the data is not included in the Variant.

Remarks

You can use **Column** to assign the contents of a combo box or list box to another control, such as a text box. For example, you can set the **ControlSource**

property of a text box to the value in the second column of a list box.

If the user makes no selection when you refer to a column in a combo box or list box, the **Column** setting is **Null**. You can check for this condition by using the **IsNull** function.

You can also use **Column** to copy an entire two-dimensional <u>array</u> of values to a control. This syntax lets you quickly load a list of choices rather than individually loading each element of the list using **AddItem**.

Note When copying data from a two-dimensional array, **Column** transposes the contents of the array in the control so that the contents of ListBox1.Column(X, Y) is the same as MyArray(Y, X). You can also use **List** to copy an array without transposing it.



ColumnCount Property

Specifies the number of columns to display in a list box or combo box.

Syntax

object.ColumnCount [= Long]

The **ColumnCount** property syntax has these parts:

PartDescriptionobject Required. A valid object.LongOptional. Specifies the number of columns to display.

Remarks

If you set the **ColumnCount** property for a list box to 3 on an employee form, one column can list last names, another can list first names, and the third can list employee ID numbers.

Setting **ColumnCount** to 0 displays zero columns, and setting it to -1 displays all the available columns. For an <u>unbound data source</u>, there is a 10-column limit (0 to 9).

You can use the <u>ColumnWidths</u> property to set the width of the columns displayed in the control.

ColumnHeads Property

Displays a single row of column headings for list boxes, combo boxes, and objects that accept column headings.

Syntax

object.ColumnHeads [= Boolean]

The **ColumnHeads** property syntax has these parts:

Part Description

object Required. A valid object. *Boolean* Optional. Specifies whether the column headings are displayed.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueDisplay column headings.FalseDo not display column headings (default).

Headings in combo boxes appear only when the list drops down.

Remarks

When the system uses the first row of data items as column headings, they can't be selected.



ColumnWidths Property

Specifies the width of each column in a multicolumn **<u>ComboBox</u>** or **<u>ListBox</u>**.

Syntax

object.ColumnWidths [= String]

The **ColumnWidths** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. Sets the column width in points. A setting of –1 or blank results in a calculated width. A width of 0 hides a column. To specify a different unit of measurement, include the unit of measure. A value greater than 0 explicitly specifies the width of the column.

Settings

To separate column entries, use semicolons (;) as list separators. Or use the list separator specified in the Regional Settings section of the Windows Control Panel.

Any or all of the **ColumnWidths** property settings can be blank. You create a blank setting by typing a list separator without a preceding value.

If you specify a -1 in the property page, the displayed value in the property page is a blank.

To calculate column widths when **ColumnWidths** is blank or –1, the width of the control is divided equally among all columns of the list. If the sum of the specified column widths exceeds the width of the control, the list is left-aligned within the control and one or more of the rightmost columns are not displayed. Users can scroll the list using the horizontal scroll bar to display the rightmost columns.

The minimum calculated column width is 72 points (1 inch). To produce

columns narrower than this, you must specify the width explicitly.

Unless specified otherwise, column widths are measured in points. To specify another unit of measure, include the units as part of the values. The following examples specify column widths in several units of measure and describe how the various settings would fit in a three-column list box that is 4 inches wide.

Setting

Effect

90;72;90	The first column is 90 points (1.25 inch); the second column is 72 points (1 inch); the third column is 90 points.
6 cm;0;6 cm	The first column is 6 centimeters; the second column is hidden; the third column is 6 centimeters. Because part of the third column is visible, a horizontal scroll bar appears.
1.5 in;0;2.5 in	The first column is 1.5 inches, the second column is hidden, and the third column is 2.5 inches.
2 in;;2 in	The first column is 2 inches, the second column is 1 inch (default), and the third column is 2 inches. Because only half of the third column is visible, a horizontal scroll bar appears.
(Blank)	All three columns are the same width (1.33 inches).

Remarks

In a **ComboBox**, the system displays the column designated by the **TextColumn** property in the text box portion of the control. Setting **TextColumn** to -1 displays the first column that has a **ColumnWidths** value greater than 0.



ControlTipText Property

Specifies text that appears when the user briefly holds the mouse pointer over a control without clicking.

Syntax

object.ControlTipText [= String]

The **ControlTipText** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. The text that appears when the user holds the mouse pointer over a control.

Remarks

The **ControlTipText** property lets you give users tips about a control in a running form. The property can be set during <u>design time</u> but only appears by the control during <u>run time</u>.

The default value of **ControlTipText** is an empty string. When the value of **ControlTipText** is set to an empty string, no tip is available for that control.



Count Property

Returns the number of objects in a <u>collection</u>.

Syntax

object.Count

The **Count** property syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **Count** property is read only.

Note that the index value for the first page or tab of a collection is zero, the value for the second page or tab is one, and so on. For example, if a **MultiPage** contains two pages, the indexes of the pages are 0 and 1, and the value of **Count** is 2.

CurLine Property

Specifies the current line of a control.

Syntax

object.CurLine [= Long]

The **CurLine** property syntax has these parts:

PartDescriptionobject Required. A valid object.LongOptional. Specifies the current line of a control.

Remarks

The current line of a control is the line that contains the insertion point. The number of the first line is 0.

The **CurLine** property is valid when the control has focus.



CurTargetX Property

Retrieves the preferred horizontal position of the insertion point in a multiline **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.CurTargetX

The **CurTargetX** property syntax has these parts:

Part Description

object Required. A valid object.

Return Values

The **CurTargetX** property retrieves the preferred position, measured in himetric units. A himetric is 0.0001 meter.

Remarks

The target position is relative to the left edge of the control. If the length of a line is less than the value of the **CurTargetX** property, you can place the insertion point at the end of the line. The value of **CurTargetX** changes when the user sets the insertion point or when the **CurX** property is set. **CurTargetX** is read-only.

The return value is valid when the object has focus

You can use **CurTargetX** and **CurX** to move the insertion point as the user scrolls through the contents of a multiline **TextBox** or **ComboBox**. When the user moves the insertion point to another line of text by scrolling the content of the object, **CurTargetX** specifies the preferred position for the insertion point. **CurX** is set to this value if the line of text is longer than the value of **CurTargetX**. Otherwise, **CurX** is set to the end of the line of text.



CurX Property

Specifies the current horizontal position of the insertion point in a multiline **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.CurX [= Long]

The **CurX** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. Indicates the current position, measured in himetrics. A himetric is 0.0001 meter.

Remarks

The **CurX** property applies to a multiline **TextBox** or **ComboBox**. The return value is valid when the object has the <u>focus</u>

You can use **CurTargetX** and **CurX** to position the insertion point as the user scrolls through the contents of a multiline **TextBox** or **ComboBox**. When the user moves the insertion point to another line of text by scrolling the content of the object, **CurTargetX** specifies the preferred position for the insertion point. **CurX** is set to this value if the line of text is longer than the value of **CurTargetX**. Otherwise, **CurX** is set to the end of the line of text.



Cycle Property

Specifies the action to take when the user leaves the last control on a **Frame** or **Page**.

Syntax

object.Cycle [=Cycle]

The **Cycle** property syntax has these parts:

Part

Description

object Required. A valid object.

Cycle Optional. Specifies whether cycling includes controls nested in a **Frame** or **MultiPage**.

Settings

The settings for *Cycle* are:

Value

Description

- 0 Cycles through the controls on the form and the controls of the **Frame** and **MultiPage** controls that are currently displayed on the form. Cycles through the controls on the form, **Frame**, or **MultiPage**. The
- 2 focus stays within the form, **Frame**, or **MultiPage** until the focus is explicitly set to a control outside the form, **Frame**, or **MultiPage**.

If you specify a non-integer value for **Cycle**, the value is rounded up to the nearest integer.

Remarks

The <u>tab order</u> identifies the order in which controls receive the <u>focus</u> as the user tabs through a form or subform. The **Cycle** property determines the action to take when a user tabs from the last control in the tab order.

The 0 setting transfers the focus to the first control of the next **Frame** or **MultiPage** on the form when the user tabs from the last control in the tab order.

The 2 setting transfers the focus to the first control of the same form, **Frame**, or **MultiPage** when the user tabs from the last control in the tab order.



Default Property

Designates the default **<u>CommandButton</u>** on a form.

Syntax

object.Default [= Boolean]

The **Default** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the command button is the default.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueThe CommandButton is the default button.FalseThe CommandButton is not the default button (default).

Remarks

A **CommandButton** or an object that acts like a command button can be designated as the default command button. Only one object on a form can be the default command button. Setting the **Default** property to **True** for one object automatically sets it to **False** for all other objects on the form.

To choose the default command button on an active form, the user can click the button, or press ENTER when no other **CommandButton** has the <u>focus</u>.

Default is provided for <u>OLE container controls</u> that specifically act like **CommandButton** controls.

Tip You should consider making the Cancel button the default button for forms

that support operations that can't be undone (such as delete). To do this, set both **Default** and the <u>Cancel</u> property to **True**.

Delay Property

Specifies the delay on a **<u>SpinButton</u>** or **<u>ScrollBar</u>**.

Syntax

object.Delay [= Long]

The **Delay** property syntax has these parts:

PartDescriptionobject Required. A valid object.LongOptional. The delay, in milliseconds, between events.

Remarks

The **Delay** property affects the amount of time between consecutive SpinUp, SpinDown, and Change events generated when the user clicks and holds down a button on a **SpinButton** or **ScrollBar**. The first event occurs immediately. The delay to the second occurrence of the event is five times the value of the specified **Delay**. This initial lag makes it easy to generate a single event rather than a stream of events.

After the initial lag, the interval between events is the value specified for **Delay**.

The default value of **Delay** is 50 milliseconds. This means the object initiates the first event after 250 milliseconds (5 times the specified value) and initiates each subsequent event after 50 milliseconds.

DragBehavior Property

Specifies whether the system enables the drag-and-drop feature for a **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.DragBehavior [=DragBehavior]

The **DragBehavior** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.DragBehavior Optional. Specifies whether the drag-and-drop feature is enabled.

Settings

The settings for *DragBehavior* are:

Value Description

- 0 Does not allow a drag-and-drop action (default).
- 1 Allows a drag-and-drop action.

Remarks

If the **DragBehavior** property is enabled, dragging in a text box or combo box starts a drag-and-drop operation on the selected text. If **DragBehavior** is disabled, dragging in a text box or combo box selects text.

The drop-down portion of a **ComboBox** does not support drag-and-drop processes, nor does it support selection of list items within the text.

DragBehavior has no effect on a **ComboBox** whose **<u>Style</u>** property is set to 2.

Note You can combine the effects of the **EnterFieldBehavior** property and **DragBehavior** to create a large number of text box styles.

DropButtonStyle Property

Specifies the symbol displayed on the drop button in a **<u>ComboBox</u>**.

Syntax

object.DropButtonStyle [=DropButtonStyle]

The **DropButtonStyle** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.DropButtonStyleOptional. The appearance of the drop button.

Settings

0

The settings for *DropButtonStyle* are:

Value Description

- Displays a plain button, with no symbol.
- 1 Displays a down arrow (default).
- 2 Displays an ellipsis (...).
- 3 Displays a horizontal line like an underscore character.

Remarks

The recommended setting for showing items in a list is 1.



Enabled Property

Specifies whether a control can receive the <u>focus</u> and respond to user-generated events.

Syntax

object.Enabled [= Boolean]

The **Enabled** property syntax has these parts:

Part

Description

object Required. A valid object.

Boolean Optional. Whether the object can respond to user-generated events.

Settings

The settings for *Boolean* are:

Value

Description

True The control can receive the focus and respond to user-generated events, and is accessible through code (default).

The user cannot interact with the control by using the mouse, keystrokes,False accelerators, or hotkeys. The control is generally still accessible through code.

Remarks

Use the **Enabled** property to enable and disable controls. A disabled control appears dimmed, while an enabled control does not. Also, if a control displays a bitmap, the bitmap is dimmed whenever the control is dimmed. If **Enabled** is **False** for an **Image**, the control does not initiate events but does not appear dimmed.

The **Enabled** and **Locked** properties work together to achieve the following

effects:

- If **Enabled** and **Locked** are both **True**, the control can receive focus and appears normally (not dimmed) in the form. The user can copy, but not edit, data in the control.
- If **Enabled** is **True** and **Locked** is **False**, the control can receive focus and appears normally in the form. The user can copy and edit data in the control.
- If **Enabled** is **False** and **Locked** is **True**, the control cannot receive focus and is dimmed in the form. The user can neither copy nor edit data in the control.
- If **Enabled** and **Locked** are both **False**, the control cannot receive focus and is dimmed in the form. The user can neither copy nor edit data in the control.

You can combine the settings of the **Enabled** and the **TabStop** properties to prevent the user from selecting a command button with TAB, while still allowing the user to click the button. Setting **TabStop** to **False** means that the command button won't appear in the <u>tab order</u>. However, if **Enabled** is **True**, then the user can still click the command button, as long as **TakeFocusOnClick** is set to **True**.

When the user tabs into an enabled **MultiPage** or **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **MultiPage** or **TabStrip** is disabled, the first enabled page or tab of that control receives the focus. If all pages or tabs of a **MultiPage** or **TabStrip** are disabled, the control is disabled and cannot receive the focus.

If a **<u>Frame</u>** is disabled, all controls that it contains are disabled.

Clicking a disabled **ListBox** does not initiate the **Click** event.



EnterFieldBehavior Property

Specifies the selection behavior when entering a **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.EnterFieldBehavior [=EnterFieldBehavior]

The **EnterFieldBehavior** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.EnterFieldBehavior Optional. The desired selection behavior.

Settings

The settings for *EnterFieldBehavior* are:

Value

Description

- 0 Selects the entire contents of the edit region when entering the control (default).
- 1 Leaves the selection unchanged. Visually, this uses the selection that was in effect the last time the control was active.

Remarks

The **EnterFieldBehavior** property controls the way text is selected when the user tabs to the control, not when the control receives <u>focus</u> as a result of the <u>SetFocus</u> method. Following **SetFocus**, the contents of the control are not selected and the insertion point appears after the last character in the control's edit region.

Note You can combine the effects of the **EnterFieldBehavior** property and **DragBehavior** to create a large number of text box styles.



EnterKeyBehavior Property

Defines the effect of pressing ENTER in a <u>TextBox</u>.

Syntax

object.EnterKeyBehavior [= Boolean]

The **EnterKeyBehavior** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Specifies the effect of pressing ENTER.

Settings

The settings for *Boolean* are:

Value

Description

True Pressing ENTER creates a new line.

False Pressing ENTER moves the focus to the next object in the tab order (default).

Remarks

The **EnterKeyBehavior** and **MultiLine** properties are closely related. The values described above only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing ENTER always moves the <u>focus</u> to the next control in the <u>tab order</u> regardless of the value of **EnterKeyBehavior**.

The effect of pressing CTRL+ENTER also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+ENTER creates a new line regardless of the value of **EnterKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+ENTER has no effect.



ForeColor Property

Specifies the <u>foreground color</u> of an object.

Syntax

object.ForeColor [= Long]

The **ForeColor** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. A value or constant that determines the foreground color of an object.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the **RGB** function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75, as shown in the following example.

RGB(15,200,75)

Remarks

Use the **ForeColor** property for controls on forms to make them easy to read or to convey a special meaning. For example, if a text box reports the number of units in stock, you can change the color of the text when the value falls below the reorder level.

For a **ScrollBar** or **SpinButton**, **ForeColor** sets the color of the arrows. For a **Frame**, **ForeColor** changes the color of the caption. For a **Font** object, **ForeColor** determines the color of the text.



GroupName Property

Creates a group of mutually exclusive **<u>OptionButton</u>** controls.

Syntax

object.GroupName [= String]

The **GroupName** property syntax has these parts:

Part

Description

object Required. A valid **OptionButton**.

Optional. The name of the group that includes the **OptionButton**. Use *String* the same setting for all buttons in the group. The default setting is an empty string.

Remarks

To create a group of mutually exclusive **OptionButton** controls, you can put the buttons in a **Frame** on your form, or you can use the **GroupName** property. **GroupName** is more efficient for the following reasons:

- You do not have to include a **Frame** for each group. By not using a **Frame**, you reduce the number of controls on the form, and in turn, improve performance and reduce the size of the form.
- You have more design flexibility. If you use a **Frame** to create the group, all the buttons must be inside the **Frame**. If you want more than one group, you must have one **Frame** for each group. However, if you use **GroupName** to create the group, the group can include option buttons anywhere on the form. If you want more than one group, specify a unique name for each group; you can still place the individual controls anywhere on the form.
- You can create buttons with <u>transparent</u> backgrounds, which can improve the visual appearance of your form. The **Frame** is not a transparent control.

Regardless of which method you use to create the group of buttons, clicking one button in a group sets all other buttons in the same group to **False**. All option

buttons with the same **GroupName** within a single <u>container</u> are mutually exclusive. You can use the same group name in two containers, but doing so creates two groups (one in each container) rather than one group that includes both containers.

For example, assume your form includes some option buttons and a **MultiPage** that also includes option buttons. The option buttons on the **MultiPage** are one group and the buttons on the form are another group. The two groups do not affect each other. Changing the setting of a button on the **MultiPage** does not affect the buttons on the form.



Height, Width Properties

The height or width, in <u>points</u>, of an object.

Syntax

object.Height [= Single]

object.Width [= Single]

The **Height** and **Width** property syntaxes have these parts:

Description

object Required. A valid object.

Single Optional. A numeric expression specifying the dimensions of an object.

Remarks

Part

The **Height** and **Width** properties are automatically updated when you move or size a control. If you specify a setting for the **Left** or **Top** property that is less than zero, that value will be used to calculate the height or width of the control, but a portion of the control will not be visible on the form.

If you move a control from one part of a form to another, the setting of **Height** or **Width** only changes if you size the control as you move it. The settings of the control's **Left** and **Top** properties will change to reflect the control's new position relative to the edges of the form that contains it.

The value assigned to **Height** or **Width** must be greater than or equal to zero. For most systems, the recommended range of values is from 0 to +32,767. Higher values may also work depending on your system configuration.

For most systems, the recommended range of values for **Left** and **Top** is from -32,767 to +32,767. Other values may also work depending on your system configuration. For a **ComboBox**, values of **Left** and **Top** apply to the text portion of the control, not to the list portion. When you move or size a control, its new **Left** setting is automatically entered in the property sheet. When you

print a form, the control's horizontal or vertical location is determined by its **Left** or **Top** setting.



HideSelection Property

Specifies whether selected text remains highlighted when a control does not have the <u>focus</u>.

Syntax

object.HideSelection [= Boolean]

The **HideSelection** property syntax has these parts:

Part

Description

object Required. A valid object.

Boolean Optional. Whether the selected text remains highlighted even when the control does not have the focus.

Settings

The settings for Boolean are:

Value

Description

True Selected text is not highlighted unless the control has the focus (default).False Selected text always appears highlighted.

Remarks

You can use the **HideSelection** property to maintain highlighted text when another form or a dialog box receives the focus, such as in a spell-checking procedure.



IMEMode Property

The feature or some of the options described in this Help topic are only available if support for Japanese, Simplified Chinese, Traditional Chinese, or Korean is enabled through Microsoft Office Language Settings.

Specifies the default run-time mode of the <u>Input Method Editor</u> (IME) for a control. This property applies only to applications written for Asian languages and is ignored in other applications.

Syntax

object.IMEMode [= fmIMEMode]

The **IMEMode** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmIMEMod	<i>e</i> Optional. The mode of the Input Method Editor (IME).

Settings

The settings for *fmIMEMode* are:

Value	Description
0	Does not control IME (default).
1	IME on.
2	IME off. English mode.
3	IME off. User can't turn on IME by keyboard.
4	IME on with Full-width Hiragana mode.
5	IME on with Full-width Katakana mode.
6	IME on with Half-width Katakana mode.
7	IME on with Full-width Alphanumeric mode.
8	IME on with Half-width Alphanumeric mode.
9	IME on with Full-width Hangul mode.

10 IME on with Half-width Hangul mode.

A setting of 0 indicates that the mode of the IME does not change when the control receives focus at run time. For any other value, the mode of the IME is set to the value specified by the **IMEMode** property when the control receives focus at run time.

Index Property

The position of a **Tab** object within a **Tabs** collection or a **Page** object in a **Pages** collection.

Syntax

object.Index [= Integer]

The **Index** property syntax has these parts:

Part Description

object Required. A valid object. *Integer* Optional. The index of the currently selected **Tab** object.

Remarks

The **Index** property specifies the order in which tabs appear. Changing the value of **Index** visually changes the order of **Pages** in a **MultiPage** or **Tabs** on a **TabStrip**. The index value for the first page or tab is zero, the index value of the second page or tab is one, and so on.

In a **MultiPage**, **Index** refers to a **Page** as well as the page's **Tab**. In a **TabStrip**, **Index** refers to the tab only.



InsideHeight, InsideWidth Properties

InsideHeight returns the height, in <u>points</u>, of the <u>client region</u> inside a <u>Frame</u>. **InsideWidth** returns the width, in points, of the client region inside a **Frame**.

Syntax

object.InsideHeight

object.InsideWidth

The **InsideHeight** and **InsideWidth** property syntaxes have these parts:

PartDescriptionobject Required. A valid object.

Remarks

The **InsideHeight** and **InsideWidth** properties are read-only.

IntegralHeight Property

Indicates whether a **ListBox** or **TextBox** displays full lines of text in a list or partial lines.

Syntax

object.IntegralHeight [= Boolean]

The **IntegralHeight** property syntax has these parts:

Part Description

object Required. A valid object. *Boolean* Optional. Whether the list displays partial lines of text.

Settings

The settings for *Boolean* are:

Value

Description

True The list resizes itself to display only complete items (default).

FalseThe list does not resize itself even if the item is too tall to display
completely.

Remarks

The **IntegralHeight** property relates to the height of the list, just as the <u>AutoSize</u> property relates to the width of the list.

If **IntegralHeight** is **True**, the list box automatically resizes when necessary to show full rows. If **False**, the list remains a fixed size; if items are taller than the available space in the list, the entire item is not shown.

KeepScrollBarsVisible Property

Specifies whether scroll bars remain visible when not required.

Syntax

object.KeepScrollBarsVisible [=ScrollBars]

The **KeepScrollBarsVisible** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.ScrollBarsOptional. Where scroll bars are displayed.

Settings

The settings for *ScrollBars* are:

Value

Description

- 0 Displays no scroll bars.
- 1 Displays a horizontal scroll bar.
- 2 Displays a vertical scroll bar.
- 3 Displays both a horizontal and a vertical scroll bar (default).

Remarks

If the visible region is large enough to display all the controls on an object such as a **Page** object, scroll bars are not required. The **KeepScrollBarsVisible** property determines whether the scroll bars remain visible when they are not required.

If the scroll bars are visible when they are not required, they appear normal in size, and the scroll box fills the entire width or height of the scroll bar.

If the **KeepScrollBarsVisible** property is **True**, any scroll bar on a form or page

is always visible, regardless of whether the object's contents fit within the object's borders.

LargeChange Property

Specifies the amount of movement that occurs when the user clicks between the scroll box and scroll arrow.

Syntax

object.LargeChange [= Long]

The **LargeChange** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. An integer that specifies the amount of change to the **Value** property.

Remarks

The **LargeChange** property applies only to the **<u>ScrollBar</u>**. It does not apply to the scrollbars in other controls such as a **<u>TextBox</u>** or a drop-down **<u>ComboBox</u>**.

The value of **LargeChange** is the amount by which the **ScrollBar's Value** property changes when the user clicks the area between the scroll box and scroll arrow. The direction of the movement is always toward the place where the user clicks. For example, in a horizontal **ScrollBar**, clicking to the left of the scroll box moves the scroll box to the left. In a vertical **ScrollBar**, clicking above the scroll box moves the scroll box up.

LargeChange does not have units. Any integer is a valid setting for **LargeChange**. The recommended range of values is from –32,767 to +32,767, and the value must be between the values of the **Max** and **Min** properties of the **ScrollBar**.



Left, Top Properties

The distance between a control and the left or top edge of the form that contains it.

Syntax

object.Left [= Single]

object.Top [= Single]

The **Left** and **Top** property syntaxes have these parts:

Part

Description

object Required. A valid object.

Single Optional. A numeric expression specifying the coordinates of an object.

Settings

Setting the **Left** or **Top** property to 0 places the control's edge at the left or top edge of its <u>container</u>.

Remarks

For most systems, the recommended range of values for **Left** and **Top** is from -32,767 to +32,767. Other values may also work depending on your system configuration. For a **ComboBox**, values of **Left** and **Top** apply to the text portion of the control, not to the list portion. When you move or size a control, its new **Left** setting is automatically entered in the property sheet. When you print a form, the control's horizontal or vertical location is determined by its **Left** or **Top** setting.

The <u>Height</u> and <u>Width</u> properties are automatically updated when you move or size a control. If you specify a setting for the **Left** or **Top** property that is less than zero, that value will be used to calculate the height or width of the control, but a portion of the control will not be visible on the form.

If you move a control from one part of a form to another, the setting of **Height** or **Width** only changes if you size the control as you move it. The settings of the control's **Left** and **Top** properties will change to reflect the control's new position relative to the edges of the form that contains it.

The value assigned to **Height** or **Width** must be greater than or equal to zero. For most systems, the recommended range of values is from 0 to +32,767. Higher values may also work depending on your system configuration.

LineCount Property

Returns the number of text lines in a **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.LineCount

The **LineCount** property syntax has these parts:

PartDescriptionobject Required. A valid object.

Remarks

The **LineCount** property is read-only.

Note A **ComboBox** will only have one line.



List Property

Returns or sets the list entries of a **ListBox** or **ComboBox**.

Syntax

object.List(row, column) [= Variant]

The **List** property syntax has these parts:

Part	Description
object	Required. A valid object.
row	Required. An integer with a range from 0 to one less than the number of entries in the list.
column	Required. An integer with a range from 0 to one less than the number of columns.
Variant	Optional. The contents of the specified entry in the ListBox or ComboBox .

Settings

Row and column numbering begins with zero. That is, the row number of the first row in the list is zero; the column number of the first column is zero. The number of the second row or column is 1, and so on.

Remarks

The **List** property works with the **ListCount** and **ListIndex** properties. Use **List** to access list items. A list is a variant <u>array</u>; each item in the list has a row number and a column number.

Initially, **ComboBox** and **ListBox** contain empty lists.

Note To specify items you want to display in a ComboBox or ListBox, use the AddItem method. To remove items, use the **RemoveItem** method.

Use **List** to copy an entire two-dimensional array of values to a control. Use

AddItem to load a one-dimensional array or to load an individual element.

ListCount Property

Returns the number of list entries in a control.

Syntax

object.ListCount

The **ListCount** property syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **ListCount** property is read-only. **ListCount** is the number of rows over which you can scroll. **ListCount** is always one greater than the largest value for the **ListIndex** property, because index numbers begin with 0 and the count of items begins with 1. If no item is selected, **ListCount** is 0 and **ListIndex** is –1.



ListIndex Property

Identifies the currently selected item in a **ListBox** or **ComboBox**.

Syntax

object.ListIndex [= Variant]

The **ListIndex** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Variant Optional. The currently selected item in the control.

Remarks

The **ListIndex** property contains an index of the selected row in a list. Values of **ListIndex** range from -1 to one less than the total number of rows in a list (that is, **ListCount** -1). When no rows are selected, **ListIndex** returns -1. When the user selects a row in a **ListBox** or **ComboBox**, the system sets the **ListIndex** value. The **ListIndex** value of the first row in a list is 0, the value of the second row is 1, and so on.

Note If you use the **MultiSelect** property to create a **ListBox** that allows multiple selections, the **Selected** property of the **ListBox** (rather than the **ListIndex** property) identifies the selected rows. The **Selected** property is an **array** with the same number of values as the number of rows in the **ListBox**. For each row in the list box, **Selected** is **True** if the row is selected and **False** if it is not. In a **ListBox** that allows multiple selections, **ListIndex** returns the index of the row that has <u>focus</u>, regardless of whether that row is currently selected.

The **ListIndex** value is also available by setting the **BoundColumn** property to 0 for a combo box or list box. If **BoundColumn** is 0, the underlying <u>data source</u> to which the combo box or list box is <u>bound</u> contains the same list index value as **ListIndex**.

ListRows Property

Specifies the maximum number of rows to display in the list.

Syntax

object.ListRows [= Long]

The **ListRows** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. An integer indicating the maximum number of rows. The default value is 8.

Remarks

If the number of items in the list exceeds the value of the **ListRows** property, a scroll bar appears at the right edge of the list-box portion of the combo box.

ListStyle Property

Specifies the visual appearance of the list in a **ListBox** or **ComboBox**.

Syntax

object.ListStyle [=ListStyle]

The **ListStyle** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.ListStyleOptional. The visual style of the list.

Settings

The settings for *ListStyle* are:

Value

Description

- 0 Looks like a regular list box, with the background of items highlighted. Shows option buttons, or check boxes for a multi-select list (default).
- 1 When the user selects an item from the group, the option button associated with that item is selected and the option buttons for the other items in the group are deselected.

Remarks

The **ListStyle** property lets you change the visual presentation of a **ListBox** or **ComboBox**. By specifying a setting other than 0, you can present the contents of either control as a group of individual items, with each item including a visual cue to indicate whether it is selected.

If the control supports a single selection (the **MultiSelect** property is set to 0), the user can press one button in the group. If the control supports multi-select, the user can press two or more buttons in the group.

ListWidth Property

Specifies the width of the list in a **<u>ComboBox</u>**.

Syntax

object.ListWidth [= Variant]

The **ListWidth** property syntax has these parts:

Part

Description

object Required. A valid object.

Optional. The width of the list. A value of zero makes the list as wide as *Variant* the **ComboBox**. The default value is to make the list as wide as the text portion of the control.

Remarks

If you want to display a multicolumn list, enter a value that will make the list box wide enough to fit all the columns.

Tip When designing combo boxes, be sure to leave enough space to display your data and for a vertical scroll bar.



Locked Property

Specifies whether a control can be edited.

Syntax

object.Locked [= Boolean]

The **Locked** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control can be edited.

Settings

The settings for *Boolean* are:

Value Description

True You can't edit the value.False You can edit the value (default).

Remarks

When a control is locked and enabled, it can still initiate events and can still receive the <u>focus</u>.

The **Enabled** and **Locked** properties work together to achieve the following effects:

- If **Enabled** and **Locked** are both **True**, the control can receive focus and appears normally (not dimmed) in the form. The user can copy, but not edit, data in the control.
- If **Enabled** is **True** and **Locked** is **False**, the control can receive focus and appears normally in the form. The user can copy and edit data in the

control.

- If **Enabled** is **False** and **Locked** is **True**, the control cannot receive focus and is dimmed in the form. The user can neither copy nor edit data in the control.
- If **Enabled** and **Locked** are both **False**, the control cannot receive focus and is dimmed in the form. The user can neither copy nor edit data in the control.



MatchEntry Property

Returns or sets a value indicating how a **ListBox** or **ComboBox** searches its list as the user types.

Syntax

object.MatchEntry [=MatchEntry]

The **MatchEntry** property syntax has these parts:

Part Description

objectRequired. A valid object.MatchEntry Optional. The rule used to match entries in the list.

Settings

The settings for *MatchEntry* are:

Value

Description

Basic matching. The control searches for the next entry that starts with

- 0 the character entered. Repeatedly typing the same letter <u>cycles</u> through all entries beginning with that letter.
- 1 Extended matching. As each character is typed, the control searches for an entry matching all characters entered (default).
- 2 No matching.

Remarks

The **MatchEntry** property searches entries from the **<u>TextColumn</u>** property of a **ListBox** or **ComboBox**.

The control searches the column identified by **TextColumn** for an entry that matches the user's typed entry. Upon finding a match, the row containing the match is selected, the contents of the column are displayed, and the contents of

its **BoundColumn** property become the value of the control. If the match is unambiguous, finding the match initiates the **<u>Click</u>** event.

The control initiates the **Click** event as soon as the user types a sequence of characters that match exactly one entry in the list. As the user types, the entry is compared with the current row in the list and with the next row in the list. When the entry matches only the current row, the match is unambiguous.

In Microsoft Forms, this is true regardless of whether the list is sorted. This means the control finds the first occurrence that matches the entry, based on the order of items in the list. For example, entering either "abc" or "bc" will initiate the Click event for the following list:

abcde bcdef abcxyz bchij

Note that in either case, the matched entry is not unique; however, it is sufficiently different from the adjacent entry that the control interprets the match as unambiguous and initiates the **Click** event.

MatchFound Property

Indicates whether the text that a user has typed into a **<u>ComboBox</u>** matches any of the entries in the list.

Syntax

object.MatchFound

The **MatchFound** property syntax has these parts:

Part Description

object Required. A valid object.

Return Values

The **MatchFound** property return values are:

Value Description

True The contents of the <u>Value</u> property matches one of the records in the list.
 False The contents of Value does not match any of the records in the list (default).

Remarks

The **MatchFound** property is read-only. It is not applicable when the **MatchEntry** property is set to 2.



MatchRequired Property

Specifies whether a value entered in the text portion of a <u>ComboBox</u> must match an entry in the existing list portion of the control. The user can enter non-matching values, but may not leave the control until a matching value is entered.

Syntax

object.MatchRequired [= Boolean]

The **MatchRequired** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.

Boolean Optional. Whether the text entered must match an existing item in the list.

Settings

The settings for *Boolean* are:

Value

Description

True The text entered must match an existing list entry.False The text entered can be different from all existing list entries (default).

Remarks

If the **MatchRequired** property is **True**, the user cannot exit the **ComboBox** until the text entered matches an entry in the existing list. **MatchRequired** maintains the integrity of the list by requiring the user to select an existing entry.

Note Not all <u>containers</u> enforce this property.

Max, Min Properties

Specify the maximum and minimum acceptable values for the <u>Value</u> property of a <u>ScrollBar</u> or <u>SpinButton</u>.

Syntax

object.Max [= Long]

object.**Min** [= Long]

The **Max** and **Min** property syntaxes have these parts:

Part

Description

object Required. A valid object.

Long Optional. A numeric expression specifying the maximum or minimum **Value** property setting.

Remarks

Clicking a **SpinButton** or moving the scroll box in a **ScrollBar** changes the **Value** property of the control.

The value for the **Max** property corresponds to the lowest position of a vertical **ScrollBar** or the rightmost position of a horizontal **ScrollBar**. The value for the **Min** property corresponds to the highest position of a vertical **ScrollBar** or the leftmost position of a horizontal **ScrollBar**.

Any integer is an acceptable setting for this property. The recommended range of values is from –32,767 to +32,767. The default value is 1.

Note Min and **Max** refer to locations, not to relative values, on the **ScrollBar**. That is, the value of **Max** could be less than the value of **Min**. If this is the case, moving toward the **Max** (bottom) position means decreasing **Value**; moving toward the **Min** (top) position means increasing **Value**.

MaxLength Property

Specifies the maximum number of characters a user can enter in a **<u>TextBox</u>** or **<u>ComboBox</u>**.

Syntax

object.MaxLength [= Long]

The **MaxLength** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. An integer indicating the allowable number of characters.

Remarks

Setting the **MaxLength** property to 0 indicates there is no limit other than that created by memory constraints.

MouseIcon Property

Assigns a custom icon to an object.

Syntax

object.MouseIcon = LoadPicture(pathname)

The **MouseIcon** property syntax has these parts:

Part		Description	
obiect	Required. A valid object.		

pathname Required. A string expression specifying the path and filename of the file containing the custom icon.

Remarks

The **MouseIcon** property is valid when the **MousePointer** property is set to 99. The mouse icon of an object is the image that appears when the user moves the mouse across that object.

To assign an image for the mouse pointer, you can either assign a picture to the **MouseIcon** property or load a picture from a file using the **LoadPicture** function.

MousePointer Property

Specifies the type of pointer displayed when the user positions the mouse over a particular object.

Syntax

object.MousePointer [=MousePointer]

The **MousePointer** property syntax has these parts:

Part	Description
object	Required. A valid object.
MousePointe	er Optional. The shape you want for the mouse pointer.

Settings

The settings for *MousePointer* are:

Value

Description

- 0 Standard pointer. The image is determined by the object (default).
- 1 Arrow.
- 2 Cross-hair pointer.
- 3 I-beam.
- 6 Double arrow pointing northeast and southwest.
- 7 Double arrow pointing north and south.
- 8 Double arrow pointing northwest and southeast.
- 9 Double arrow pointing west and east.
- 10 Up arrow.
- 11 Hourglass.
- 12 "Not" symbol (circle with a diagonal line) on top of the object being dragged. Indicates an invalid drop target.
- 13 Arrow with an hourglass.
- 14 Arrow with a question mark.

- 15 Size all cursor (arrows pointing north, south, east, and west).
- 99 Uses the icon specified by the **MouseIcon** property.

Remarks

Use the **MousePointer** property when you want to indicate changes in functionality as the mouse pointer passes over controls on a form. For example, the hourglass setting (11) is useful to indicate that the user must wait for a process or operation to finish.

Some icons vary depending on system settings, such as the icons associated with desktop themes.



MultiLine Property

Specifies whether a control can accept and display multiple lines of text.

Syntax

object.MultiLine [= Boolean]

The **MultiLine** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control supports more than one line of text.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueThe text is displayed across multiple lines (default).FalseThe text is not displayed across multiple lines.

Remarks

A multiline <u>TextBox</u> allows absolute line breaks and adjusts its quantity of lines to accommodate the amount of text it holds. If needed, a multiline control can have vertical scroll bars.

A single-line **TextBox** doesn't allow absolute line breaks and doesn't use vertical scroll bars.

For controls that support the **MultiLine** property as well as the **WordWrap** property, **WordWrap** is ignored when **MultiLine** is **False**.

Single-line controls ignore the value of the **WordWrap** property.

Note If you change **MultiLine** to **False** in a multiline **TextBox**, all the characters in the **TextBox** will be combined into one line, including non-printing characters (such as carriage returns and new-lines).

The **EnterKeyBehavior** and **MultiLine** properties are closely related. The **EnterKeyBehavior** values of **True** and **False** only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing ENTER always moves the <u>focus</u> to the next control in the <u>tab order</u> regardless of the value of **EnterKeyBehavior**.

The effect of pressing CTRL+ENTER also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+ENTER creates a new line regardless of the value of **EnterKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+ENTER has no effect.

The **TabKeyBehavior** and **MultiLine** properties are closely related. The values described above only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing TAB always moves the <u>focus</u> to the next control in the <u>tab order</u> regardless of the value of **TabKeyBehavior**.

The effect of pressing CTRL+TAB also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+TAB creates a new line regardless of the value of **TabKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+TAB has no effect.

MultiRow Property

Specifies whether the control has more than one row of tabs.

Syntax

object.**MultiRow** [= Boolean]

The **MultiRow** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control has more than one row of tabs.

Settings

The settings for *Boolean* are:

ValueDescriptionTrueAllows more than one row of tabs.FalseRestricts tabs to a single row (default).

Remarks

The width and number of tabs determines the number of rows. Changing the control's size also changes the number of rows. This allows the developer to resize the control and ensure that tabs wrap to fit the control. If the **MultiRow** property is **False**, then truncation occurs if the width of the tabs exceeds the width of the control.

If **MultiRow** is **False** and tabs are truncated, there will be a small scroll bar on the **TabStrip** to allow scrolling to the other tabs or pages.



MultiSelect Property

Indicates whether the object permits multiple selections.

Syntax

```
object.MultiSelect [=MultiSelect]
```

The **MultiSelect** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.MultiSelectOptional. The selection mode that the control uses.

Settings

The settings for *MultiSelect* are:

Value

Description

- 0 Only one item can be selected (default).
- 1 Pressing the SPACEBAR or clicking selects or deselects an item in the list.

Pressing SHIFT and clicking the mouse, or pressing SHIFT and one of the arrow keys, extends the selection from the previously selected item to

2 the current item. Pressing CTRL and clicking the mouse selects or deselects an item.

Remarks

When the **MultiSelect** property is set to *Extended* or *Simple*, you must use the list box's <u>Selected</u> property to determine the selected items. Also, the <u>Value</u> property of the control is always **Null**.

The **ListIndex** property returns the index of the row with the keyboard <u>focus</u>. <P>



Name Property

Specifies the name of a control or an object, or the name of a font to associate with a **Font** object.

Syntax

For Font

Font.Name [= String]

For all other controls and objects *object*.**Name** [= *String*]

The **Name** property syntax has these parts:

PartDescriptionobject Required. A valid object.String Optional. The name you want to assign to the font or control.

Settings

Guidelines for assigning a string to **Name**, such as the maximum length of the name, vary from one application to another.

Remarks

For objects, the default value of **Name** consists of the object's <u>class</u> name followed by an integer. For example, the default name for the first <u>TextBox</u> you place on a form is TextBox1. The default name for the second **TextBox** is TextBox2.

You can set the **Name** property for a control from the control's property sheet or, for controls added at <u>run time</u>, by using program statements. If you add a control at <u>design time</u>, you cannot modify its **Name** property at run time.

Each control added to a form at design time must have a unique name.

For **Font** objects, **Name** identifies a particular typeface to use in the text portion of a control, object, or form. The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display or that isn't installed, Windows substitutes a similar font.

Object Property

Overrides a standard property or method when a new control has a property or method of the same name.

Syntax

object.Object[.property |.method]

The **Object** property syntax has these parts:

Part	Description
object	Required. The name of an object you have added to the Microsoft Forms Toolbox.
	Optional. A property that has the same name as a standard Microsoft Forms property.
	Optional. A method that has the same name as a standard Microsoft Forms method.

Remarks

Object is read-only.

If you add a new control to the Microsoft Forms Toolbox, it is possible that the added control will have a property or method with the same name as a standard Microsoft Forms property or method. The **Object** property lets you use the property or method from the added control, rather than the standard property or method.

Orientation Property

Specifies whether the **<u>SpinButton</u>** or **<u>ScrollBar</u>** is oriented vertically or horizontally.

Syntax

object.Orientation [=Orientation]

The **Orientation** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.OrientationOptional. Orientation of the control.

Settings

The settings for *Orientation* are:

Value

Description

- -1 Automatically determines the orientation based upon the dimensions of the control (default).
- 0 Control is rendered vertically.
- 1 Control is rendered horizontally.

Remarks

If you specify automatic orientation, the height and width of the control determine whether it appears horizontally or vertically. For example, if the control is wider than it is tall, it appears horizontally; if it is taller than it is wide, the control appears vertically.



Parent Property

Returns the name of the form, object, or <u>collection</u> that contains a specific control, object, or collection.

Syntax

object.Parent

The **Parent** property syntax has these parts:

Part Description

object Required. A valid object.

Remarks

Parent is read-only.

Use the **Parent** property to access the properties, methods, or controls of an object's parent.

This property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure in a module, and use **Parent** to access its parent form.



PasswordChar Property

Specifies whether <u>placeholder</u> characters are displayed instead of the characters actually entered in a <u>TextBox</u>.

Syntax

object.PasswordChar [= String]

The **PasswordChar** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. A string expression specifying the placeholder character.

Remarks

You can use the **PasswordChar** property to protect sensitive information, such as passwords or security codes. The value of **PasswordChar** is the character (usually an asterisk) that appears in a control instead of the actual characters that the user types. If you don't specify a character, the control displays the characters that the user types.



Picture Property

Specifies the bitmap to display on an object.

Syntax

object.Picture = LoadPicture(pathname)

The **Picture** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.pathnameRequired. The full path to a picture file.

Remarks

You must use the control's <u>property page</u> to assign a bitmap to the **Picture** property. You cannot use the **LoadPicture** function to assign a bitmap to **Picture**.

To remove a picture that is assigned to a control, click the value of the **Picture** property in the property page and then press DELETE. Pressing BACKSPACE will not remove the picture.

Note For controls with captions, use the **<u>PicturePosition</u>** property to specify where to display the picture on the object. Use the **<u>PictureSizeMode</u>** property to determine how the picture fills the object.

Transparent pictures sometimes have a hazy appearance. If you do not like this appearance, display the picture on a control that supports opaque images. **Image** and **MultiPage** support opaque images.



PictureAlignment Property

Specifies the location of a background picture.

Syntax

object.PictureAlignment [=PictureAlignment]

The **PictureAlignment** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.PictureAlignmentOptional. The position where the picture aligns with the
control.

Settings

The settings for *PictureAlignment* are:

Value Description

- 0 The top left corner.
- 1 The top right corner.
- 2 The center.
- 3 The bottom left corner.
- 4 The bottom right corner.

Remarks

The **PictureAlignment** property identifies which corner of the picture is the same as the corresponding corner of the control or <u>container</u> where the picture is used.

For example, setting **PictureAlignment** to 0 means that the top left corner of the picture coincides with the top left corner of the control or container. Setting **PictureAlignment** to 2 positions the picture in the middle, relative to the height

as well as the width of the control or container.

If you tile an image on a control or container, the setting of **PictureAlignment** affects the tiling pattern. For example, if **PictureAlignment** is set to 0, the first copy of the image is laid in the upper left corner of the control or container and additional copies are tiled from left to right across each row. If **PictureAlignment** is 2, the first copy of the image is laid at the center of the control or container, additional copies are laid to the left and right to complete the row, and additional rows are added to fill the control or container.

Note Setting the **<u>PictureSizeMode</u>** property to 2 overrides **PictureAlignment**. When **PictureSizeMode** is set to 2, the picture fills the entire control or container.

PicturePosition Property

Specifies the location of the picture relative to its caption.

Syntax

object.PicturePosition [=PicturePosition]

The **PicturePosition** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.PicturePosition Optional. How the picture aligns with its container.

Settings

The settings for *PicturePosition* are:

Value

Description

- 0 The picture appears to the left of the caption. The caption is aligned with the top of the picture.
- 1 The picture appears to the left of the caption. The caption is centered relative to the picture.
- 2 The picture appears to the left of the caption. The caption is aligned with the bottom of the picture.
- 3 The picture appears to the right of the caption. The caption is aligned with the top of the picture.
- 4 The picture appears to the right of the caption. The caption is centered relative to the picture.
- 5 The picture appears to the right of the caption. The caption is aligned with the bottom of the picture.
- 6 The picture appears above the caption. The caption is aligned with the left edge of the picture.
- 7 The picture appears above the caption. The caption is centered below the picture (default).

- 8 The picture appears above the caption. The caption is aligned with the right edge of the picture.
- 9 The picture appears below the caption. The caption is aligned with the left edge of the picture.
- 10 The picture appears below the caption. The caption is centered above the picture.
- 11 The picture appears below the caption. The caption is aligned with the right edge of the picture.
- 12 The picture appears in the center of the control. The caption is centered horizontally and vertically on top of the picture.

Remarks

The picture and the caption, as a unit, are centered on the control. If no caption exists, the picture's location is relative to the center of the control.

This property is ignored if the **<u>Picture</u>** property does not specify a picture.



PictureSizeMode Property

Specifies how to display the background picture on a control, form, or page.

Syntax

object.PictureSizeMode [=PictureSizeMode]

The **PictureSizeMode** property syntax has these parts:

Part	Description
object	Required. A valid object.
PictureSizeMode	Optional. The action to take if the picture and the form or page that contains it are not the same size.
	that contains it are not the same size.

Settings

The settings for *PictureSizeMode* are:

Value

Description 0 Crops any part of the picture that is larger than the form or page (default).

- Stretches the picture to fill the form or page. This setting distorts the 1 picture in either the horizontal or vertical direction.
- Enlarges the picture, but does not distort the picture in either the 3 horizontal or vertical direction.

Remarks

The **PictureSizeModeClip** setting indicates you want to show the picture in its original size and scale. If the form or page is smaller than the picture, this setting only shows the part of the picture that fits within the form or page.

The 1 and 3 settings both enlarge the image, but 1 causes distortion. The 1 setting enlarges the image horizontally and vertically until the image reaches the corresponding edges of the <u>container</u> or control. The 3 setting enlarges the image until it reaches either the horizontal or vertical edges of the container or control. If the image reaches the horizontal edges first, any remaining distance to the vertical edges remains blank. If it reaches the vertical edges first, any remaining distance to the horizontal edges remains blank.

Note Setting the **PictureSizeMode** property to 2 overrides **PictureAlignment**. When **PictureSizeMode** is set to 2, the picture fills the entire control or container.

PictureTiling Property

Lets you tile a picture in a form or page.

Syntax

object.PictureTiling [= Boolean]

The **PictureTiling** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether a picture is repeated across a background.

Settings

The settings for *Boolean* are:

Value Description

True The picture is tiled across the background.False The picture is not tiled across the background (default).

Remarks

If a picture is smaller than the form or page that contains it, you can tile the picture on the form or page.

The tiling pattern depends on the current setting of the **PictureAlignment** and **PictureSizeMode** properties. For example, if **PictureAlignment** is set to 0, the tiling pattern starts at the upper left and repeats the picture across the form or page and down the height of the form or page. If **PictureSizeMode** is set to 0, the tiling pattern crops the last tile if it doesn't completely fit on the form or page.

ProportionalThumb Property

Specifies whether the size of the scroll box is proportional to the scrolling region or fixed.

Syntax

object.ProportionalThumb [= Boolean]

The **ProportionalThumb** property syntax has these parts:

Part Description

object Required. A valid object. *Boolean* Optional. Whether the scroll box is proportional or fixed.

Settings

The settings for *Boolean* are:

Value

Description

True The scroll box is proportional in size to the scrolling region (default).False The scroll box is a fixed size.

Remarks

The size of a proportional scroll box graphically represents the percentage of the object that is visible in the window. For example, if 75 percent of an object is visible, the scroll box covers three-fourths of the scrolling region in the scroll bar.

If the scroll box is a fixed size, the system determines its size based on the height and width of the scroll bar.

RowSource Property

Specifies the source providing a list for a **<u>ComboBox</u>** or **<u>ListBox</u>**.

Syntax

object.RowSource [= String]

The **RowSource** property syntax has these parts:

PartDescriptionobject Required. A valid object.String Optional. The source of the list for the ComboBox or ListBox.

Remarks

The **RowSource** property accepts worksheet ranges from Microsoft Excel.

ScrollBars Property

Specifies whether a control, form, or page has vertical scroll bars, horizontal scroll bars, or both.

Syntax

object.ScrollBars [=ScrollBars]

The **ScrollBars** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.ScrollBars Optional. Where scroll bars should be displayed.

Settings

The settings for *ScrollBars* are:

Value Description

- 0 Displays no scroll bars (default).
- 1 Displays a horizontal scroll bar.
- 2 Displays a vertical scroll bar.
- 3 Displays both a horizontal and a vertical scroll bar.

Remarks

If the **KeepScrollBarsVisible** property is **True**, any scroll bar on a form or page is always visible, regardless of whether the object's contents fit within the object's borders.

If visible, a scroll bar constrains its scroll box to the visible region of the scroll bar. It also modifies the scroll position as needed to keep the entire scroll bar visible. The range of a scroll bar changes when the value of the **ScrollBars** property changes, the scroll size changes, or the visible size changes.

If a scroll bar is not visible, then you can set its scroll position to any value. Negative values and values greater than the scroll size are both valid.

For a single-line control, you can display a horizontal scroll bar by using the **ScrollBars** and **AutoSize** properties. Scroll bars are hidden or displayed according to the following rules:

- 1. When **ScrollBars** is set to **ScrollBarsNone**, no scroll bar is displayed.
- 2. When **ScrollBars** is set to 1 or 3, the control displays a horizontal scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar underneath its edit region.
- 3. When **AutoSize** is **True**, the control enlarges itself to accommodate the addition of a scroll bar unless the control is at or near its maximum size.

For a multiline **TextBox**, you can display scroll bars by using the **ScrollBars**, **WordWrap**, and **AutoSize** properties. Scroll bars are hidden or displayed according to the following rules:

- 1. When **ScrollBars** is set to 0, no scroll bar is displayed.
- 2. When **ScrollBars** is set to 2 or 3, the control displays a vertical scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar at the right edge of its edit region.
- 3. When **WordWrap** is **True**, the multiline control will not display a horizontal scroll bar. Most multiline controls do not use a horizontal scroll bar.
- 4. A multiline control can display a horizontal scroll bar if the following conditions occur simultaneously:
 - The edit region contains a word that is longer than the edit region's width.
 - The control has enabled horizontal scroll bars.
 - The control has enough room to include the scroll bar under the edit region.
 - The **WordWrap** property is set to **False**.



ScrollHeight, ScrollWidth Properties

Specify the height, in <u>points</u>, of the total area that can be viewed by moving the scroll bars on the control, form, or page.

Syntax

object.ScrollHeight [= Single]

object.ScrollWidth [= Single]

The **ScrollHeight** and **ScrollWidth** property syntaxes have these parts:

PartDescriptionobject Required. A valid object.Single Optional. The height or width of the scrollable region.



ScrollLeft, ScrollTop Properties

Specify the distance, in <u>points</u>, of the left or top edge of the visible form from the left or top edge of the logical form, page, or control.

Syntax

object.ScrollLeft [= Single]

object.ScrollTop [= Single]

The **ScrollLeft** and **ScrollTop** property syntaxes have these parts:

PartDescriptionobject Required. A valid object.Single Optional. The distance from the edge of the form.

Remarks

The minimum value is zero; the maximum value is the difference between the value of the <u>ScrollWidth</u> property and the value of the <u>Width</u> property for the form or page.



Selected Property

Returns or sets the selection state of items in a **ListBox**.

Syntax

object.Selected(index) [= Boolean]

The **Selected** property syntax has these parts:

Part	Description
object	Required. A valid object.
index	Required. An integer with a range from 0 to one less than the number of items in the list.
Boolear	Optional. Whether an item is selected.

Settings

The settings for *Boolean* are:

Value Description

True The item is selected.False The item is not selected.

Remarks

The **Selected** property is useful when users can make multiple selections. You can use this property to determine the selected rows in a multi-select list box. You can also use this property to select or deselect rows in a list from code.

The default value of this property is based on the current selection state of the **ListBox**.

For single-selection list boxes, the <u>Value</u> or <u>ListIndex</u> properties are recommended for getting and setting the selection. In this case, **ListIndex**

returns the index of the selected item. However, in a multiple selection, **ListIndex** returns the index of the row contained within the <u>focus</u> rectangle, regardless of whether the row is actually selected.

When a list box control's **<u>MultiSelect</u>** property is set to *None*, only one row can have its **Selected** property set to **True**.

Entering a value that is out of range for the index does not generate an error message, but does not set a property for any item in the list.

SelectedItem Property

Returns the currently selected **Tab** or **Page** object.

Syntax

object.SelectedItem

The **SelectedItem** property syntax has these parts:

PartDescriptionobject Required. A valid TabStrip or MultiPage.

Remarks

The **SelectedItem** property is read-only. Use **SelectedItem** to programmatically control the currently selected **Tab** or **Page** object. For example, you can use **SelectedItem** to assign values to properties of a **Tab** or **Page** object.

SelectionMargin Property

Specifies whether the user can select a line of text by clicking in the region to the left of the text.

Syntax

object.SelectionMargin [= Boolean]

The **SelectionMargin** property syntax has these parts:

Description

object Required. A valid object. *Boolean* Optional. Whether clicking in the margin selects a line of text.

Settings

Part

The settings for *Boolean* are:

Value Description

True Clicking in margin causes selection of text (default).False Clicking in margin does not cause selection of text.

Remarks

When the **SelectionMargin** property is **True**, the selection margin occupies a thin strip along the left edge of a control's edit region. When set to **False**, the entire edit region can store text.

If the **SelectionMargin** property is set to **True** when a control is printed, the selection margin also prints.



SelLength Property

The number of characters selected in a <u>**TextBox**</u> or the text portion of a <u>**ComboBox**</u>.

Syntax

object.SelLength [= Long]

The **SelLength** property syntax has these parts:

Part

Description

object Required. A valid object.

Optional. A numeric expression specifying the number of characters selected. For **SelLength** and **SelStart**, the valid range of settings is 0 to

Long the total number of characters in the edit area of a **ComboBox** or **TextBox**.

Remarks

The **SelLength** property is always valid, even when the control does not have <u>focus</u>. Setting **SelLength** to a value less than zero creates an error. Attempting to set **SelLength** to a value greater than the number of characters available in a control results in a value equal to the number of characters in the control.

Note Changing the value of the **SelStart** property cancels any existing selection in the control, places an insertion point in the text, and sets **SelLength** to zero.

The default value, zero, means that no text is currently selected.



SelStart Property

Indicates the starting point of selected text, or the insertion point if no text is selected.

Syntax

object.SelStart [= Long]

The **SelStart** property syntax has these parts:

Part

Description

object Required. A valid object.

Optional. A numeric expression specifying the starting point of text selected. For <u>SelLength</u> and **SelStart**, the valid range of settings is 0 to

Long the total number of characters in the edit area of a <u>ComboBox</u> or <u>TextBox</u>. The default value is zero.

Remarks

The **SelStart** property is always valid, even when the control does not have <u>focus</u>. Setting **SelStart** to a value less than zero creates an error. Attempting to set **SelStart** to a value greater than the number of characters available in a control results in a value equal to the number of characters in the control.

Changing the value of **SelStart** cancels any existing selection in the control, places an insertion point in the text, and sets the **SelLength** property to zero.



SelText Property

Returns or sets the selected text of a control.

Syntax

object.SelText [= String]

The **SelText** property syntax has these parts:

PartDescriptionobject Required. A valid object.String Optional. A string expression containing the selected text.

Remarks

If no characters are selected in the edit region of the control, the **SelText** property returns a zero length string. This property is valid regardless of whether the control has the <u>focus</u>.

ShowDropButtonWhen Property

Specifies when to show the drop-down button for a **<u>ComboBox</u>**.

Syntax

object.ShowDropButtonWhen [=ShowDropButtonWhen]

The **ShowDropButtonWhen** property syntax has these parts:

Part	Description	
object	Required. A valid object.	
ShowDropButtonWher	Optional. The circumstances under which the drop-down button will be visible.	

Settings

The settings for *ShowDropButtonWhen* are:

Value

Description

- 0 Do not show the drop-down button under any circumstances.
- 1 Show the drop-down button when the control has the focus.
- 2 Always show the drop-down button.

For a **ComboBox**, the default value is 2.

SmallChange Property

Specifies the amount of movement that occurs when the user clicks either scroll arrow in a **ScrollBar** or **SpinButton**.

Syntax

object.SmallChange [= Long]

The **SmallChange** property syntax has these parts:

Part

Description

object Required. A valid object.

Long Optional. An integer that specifies the amount of change to the <u>Value</u> property.

Remarks

The **SmallChange** property does not have units.

Any integer is an acceptable setting for this property. The recommended range of values is from -32,767 to +32,767. The default value is 1.



SpecialEffect Property

Specifies the visual appearance of an object.

Syntax

```
object.SpecialEffect [=SpecialEffect]
```

The **SpecialEffect** property syntax has these parts:

Part	Description
object	Required. A valid object.
SpecialEffer	, Optional. The desired visual appearance of an object other than a

SpecialEffect CheckBox, OptionButton, or ToggleButton.

Settings

The settings for *SpecialEffect* are:

Value

Description

Object appears flat, distinguished from the surrounding form by a border,

- 0 a change of color, or both. Default for <u>Image</u> and <u>Label</u>, valid for all controls.
- 1 Object has a highlight on the top and left and a shadow on the bottom and right. Not valid for check boxes or option buttons.

Object has a shadow on the top and left and a highlight on the bottom and right. The control and its border appear to be carved into the form that

- ² contains them. Default for **CheckBox** and **OptionButton**, valid for all controls (default).
- Border appears to be carved around the edge of the control. Not valid for check boxes or option buttons.
- 6 Object has a ridge on the bottom and right and appears flat on the top and left. Not valid for check boxes or option buttons.

For a **Frame**, the default value is 2.

Note that only 0 and 2 are acceptable values for **CheckBox**, **OptionButton**, and **ToggleButton**. All values listed are acceptable for other controls.

Remarks

You can use either the **SpecialEffect** or the **BorderStyle** property to specify the edging for a control, but not both. If you specify a nonzero value for one of these properties, the system sets the value of the other property to zero. For example, if you set **SpecialEffect** to 1, the system sets **BorderStyle** to 0.

For a **Frame**, **BorderStyle** is ignored if **SpecialEffect** is 0.

SpecialEffect uses the <u>system colors</u> to define its borders.

Note Although the **SpecialEffect** property exists on the **ToggleButton**, the property is disabled. You cannot set or return a value for this property on the **ToggleButton**.

Style Property

For <u>**ComboBox**</u>, specifies how the user can choose or set the control's value. For <u>**MultiPage**</u> and <u>**TabStrip**</u>, identifies the style of the tabs on the control.

Syntax

object.Style [=Style]

The **Style** property syntax has these parts:

Part

Description

object Required. A valid object.

Style Optional. Specifies how a user sets the value of a **ComboBox**.

Settings

The settings for **ComboBox** are:

Value

Description

The **ComboBox** behaves as a drop-down combo box. The user can type a

- 0 value in the edit region or select a value from the drop-down list (default).
- 2 The **ComboBox** behaves as a list box. The user must choose a value from the list.

The settings for **MultiPage** and **TabStrip** are::

Value Description

- 0 Displays tabs on the tab bar (default).
- 1 Displays buttons on the tab bar.
- 2 Does not display the tab bar.



TabFixedHeight, TabFixedWidth Properties

Sets or returns the fixed height or width of the tabs in points.

Syntax

object.TabFixedHeight [= Single]

object.TabFixedWidth [= Single]

The **TabFixedHeight** and **TabFixedWidth** property syntaxes have these parts:

Part	Description
object Required. A valid object.	
Single Optional. The number of point <u>MultiPage</u> .	nts of the height or width of the tabs on a

Settings

If the value is 0, tab widths are automatically adjusted so that each tab is wide enough to accommodate its contents and each row of tabs spans the width of the control.

If the value is greater than 0, all tabs have an identical width as specified by this property.

Remarks

The minimum size is 4 points.



TabIndex Property

Specifies the position of a single object in the form's <u>tab order</u>.

Syntax

object.TabIndex [= Integer]

The **TabIndex** property syntax has these parts:

Part

Description

object Required. A valid object.

Optional. An integer from 0 to one less than the number of controls on the form that have a **TabIndex** property. Assigning a **TabIndex** value of *Integer* less than 0 generates an error. If you assign a **TabIndex** value greater than the largest index value, the system resets the value to the maximum allowable value.

Remarks

The index value of the first object in the tab order is zero.



TabKeyBehavior Property

Determines whether tabs are allowed in the edit region.

Syntax

object.TabKeyBehavior [= Boolean]

The **TabKeyBehavior** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. The effect of pressing TAB.

Settings

The settings for *Boolean* are:

Value

Description

True Pressing TAB inserts a tab character in the edit region.

False Pressing TAB moves the focus to the next object in the tab order (default).

Remarks

The **TabKeyBehavior** and **MultiLine** properties are closely related. The values described above only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing TAB always moves the <u>focus</u> to the next control in the <u>tab order</u> regardless of the value of **TabKeyBehavior**.

The effect of pressing CTRL+TAB also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+TAB creates a new line regardless of the value of **TabKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+TAB has no effect.

TabOrientation Property

Specifies the location of the tabs on a **MultiPage** or **TabStrip**.

Syntax

object.**TabOrientation** [=TabOrientation]

The **TabOrientation** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.TabOrientationOptional. Where the tabs will appear.

Settings

The settings for *TabOrientation* are:

Value

Description

- 0 The tabs appear at the top of the control (default).
- 1 The tabs appear at the bottom of the control.
- 2 The tabs appear at the left side of the control.
- 3 The tabs appear at the right side of the control.

Remarks

If you use TrueType fonts, the text rotates when the **TabOrientation** property is set to 2 or 3. If you use bitmapped fonts, the text does not rotate.



TabStop Property

Indicates whether an object can receive <u>focus</u> when the user tabs to it.

Syntax

object.TabStop [= Boolean]

The **TabStop** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the object is a tab stop.

Settings

The settings for *Boolean* are:

Value

Description

True Designates the object as a tab stop (default).

Bypasses the object when the user is tabbing, although the object stillFalse holds its place in the actual tab order, as determined by the <u>TabIndex</u> property.

Remarks

You can combine the settings of the **Enabled** and the **TabStop** properties to prevent the user from selecting a command button with TAB, while still allowing the user to click the button. Setting **TabStop** to **False** means that the command button won't appear in the <u>tab order</u>. However, if **Enabled** is **True**, then the user can still click the command button, as long as <u>TakeFocusOnClick</u> is set to **True**.

When the user tabs into an enabled **MultiPage** or **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **MultiPage** or

TabStrip is disabled, the first enabled page or tab of that control receives the focus. If all pages or tabs of a **MultiPage** or **TabStrip** are disabled, the control is disabled and cannot receive the focus.

Tag Property

Stores additional information about an object.

Syntax

object.Tag [= String]

The **Tag** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. A string expression identifying the object. The default is a zero-length string ("").

Remarks

Use the **Tag** property to assign an identification string to an object without affecting other property settings or attributes.

For example, you can use **Tag** to check the identity of a form or control that is passed as a variable to a procedure.



TakeFocusOnClick Property

Specifies whether a control takes the <u>focus</u> when clicked.

Syntax

object.TakeFocusOnClick [= Boolean]

The **TakeFocusOnClick** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Specifies whether a control takes the focus when clicked.

Settings

The settings for *Boolean* are:

Value Description

True The button takes the focus when clicked (default).False The button does not take the focus when clicked.

Remarks

The **TakeFocusOnClick** property defines only what happens when the user clicks a control. If the user tabs to the control, the control takes the focus regardless of the value of **TakeFocusOnClick**.

Use this property to complete actions that affect a control without requiring that control to give up focus. For example, assume your form includes a **TextBox** and a **CommandButton** that checks for correct spelling of text. You would like to be able to select text in the **TextBox**, then click the **CommandButton** and run the spelling checker without taking focus away from the **TextBox**. You can do this by setting the **TakeFocusOnClick** property of the **CommandButton** to **False**.

Text Property

Returns or sets the text in a <u>TextBox</u>. Changes the selected row in a <u>ComboBox</u> or <u>ListBox</u>.

Syntax

object.Text [= String]

The **Text** property syntax has these parts:

Part

Description

object Required. A valid object.

String Optional. A string expression specifying text. The default value is a zero-length string ("").

Remarks

For a **TextBox**, any value you assign to the **Text** property is also assigned to the **Value** property.

For a **ComboBox**, you can use **Text** to update the value of the control. If the value of **Text** matches an existing list entry, the value of the <u>ListIndex</u> property (the index of the current row) is set to the row that matches **Text**. If the value of **Text** does not match a row, **ListIndex** is set to -1.

For a **ListBox**, the value of **Text** must match an existing list entry. Specifying a value that does not match an existing list entry causes an error.

When the **Text** property of a **ComboBox** changes (such as when a user types an entry into the control), the new text is compared to the column of data specified by **TextColumn**.

You cannot use **Text** to change the value of an entry in a **ComboBox** or **ListBox**; use the **Column** or **List** property for this purpose.

The **ForeColor** property determines the color of the text.

TextAlign Property

Specifies how text is aligned in a control.

Syntax

object.TextAlign [=TextAlign]

The **TextAlign** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.TextAlign Optional. How text is aligned in the control.

Settings

The settings for *TextAlign* are:

Value

Description

- 1 Aligns the first character of displayed text with the left edge of the control's display or edit area (default).
- 2 Centers the text in the control's display or edit area.
- 3 Aligns the last character of displayed text with the right edge of the control's display or edit area.

Remarks

For a **ComboBox**, the **TextAlign** property only affects the edit region; this property has no effect on the alignment of text in the list. For stand-alone labels, **TextAlign** determines the alignment of the label's caption.

TextColumn Property

Identifies the column in a **<u>ComboBox</u>** or **<u>ListBox</u>** to display to the user.

Syntax

object.TextColumn [= Variant]

The **TextColumn** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Variant Optional. The column to be displayed.

Settings

Values for the **TextColumn** property range from –1 to the number of columns in the list. The **TextColumn** value for the first column is 1, the value of the second column is 2, and so on. Setting **TextColumn** to 0 displays the **ListIndex** values. Setting **TextColumn** to –1 displays the first column that has a **ColumnWidths** value greater than 0.

Remarks

When the user selects a row from a **ComboBox** or **ListBox**, the column referenced by **TextColumn** is stored in the <u>Text</u> property.For example, you could set up a multicolumn **ListBox** that contains the names of holidays in one column and dates for the holidays in a second column. To present the holiday names to users, specify the first column as the **TextColumn**. To store the dates of the holidays, specify the second column as the **BoundColumn**.

When the **Text** property of a **ComboBox** changes (such as when a user types an entry into the control), the new text is compared to the column of data specified by **TextColumn**.

TextLength Property

Returns the length, in characters, of text in the edit region of a <u>TextBox</u> or <u>ComboBox</u>.

Syntax

object.TextLength

The **TextLength** property syntax has these parts:

Part Description

object Required. A valid object.

Remarks

The **TextLength** property is read-only. For a multiline **TextBox**, **TextLength** includes LF (line feed) and CR (carriage return) characters.

TopIndex Property

Sets and returns the item that appears in the topmost position in the list.

Syntax

object.TopIndex [= Variant]

The **TopIndex** property syntax has these parts:

Part Description

object Required. A valid object.

Variant Optional. The number of the list item that is displayed in the topmost position. The default is 0, or the first item in the list.

Settings

Returns the value -1 if the list is empty or not displayed.



TripleState Property

Determines whether a user can specify, from the user interface, the <u>Null</u> state for a <u>CheckBox</u> or <u>ToggleButton</u>.

Syntax

object.TripleState [= Boolean]

The **TripleState** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control supports the Null state.

Settings

The settings for *Boolean* are:

Value Description

True The button clicks through three states.False The button only supports True and False (default).

Remarks

Although the **TripleState** property exists on the **OptionButton**, the property does not affect the action of the control. Regardless of the value of **TripleState**, you cannot set the control to **Null** through the user interface.

When the **TripleState** property is **True**, a user can choose from the values of **Null**, **True**, and **False**. The null value is displayed as a shaded button.

When **TripleState** is **False**, the user can choose either **True** or **False**.

A control set to **Null** does not initiate the <u>Click</u> event.

Regardless of the property setting, the **Null** value can always be assigned programmatically to an **OptionButton**, **CheckBox** or **ToggleButton**, causing that control to appear shaded.



Value Property

Specifies the state or content of a given control.

Syntax

object.Value [= Variant]

The **Value** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.

Variant Optional. The state or content of the control.

Settings

Control	Description
CheckBox	An integer value indicating whether the item is selected:
	Null Indicates the item is in a null state, neither selected nor <u>cleared</u> .
	-1 True. Indicates the item is selected.
	0 False. Indicates the item is cleared.
OptionButton	Same as CheckBox .
ToggleButton	Same as CheckBox .
<u>ScrollBar</u>	An integer between the values specified for the Max and Min properties.
<u>SpinButton</u>	Same as ScrollBar .
<u>ComboBox,</u>	The value in the BoundColumn of the currently selected
<u>ListBox</u>	rows.
CommandButton	Always False .
<u>MultiPage</u>	An integer indicating the currently active page.
	Zero (0) indicates the first page. The maximum value is one less than the number of pages.
<u>TextBox</u>	The text in the edit region.

Remarks

For a **CommandButton**, setting the **Value** property to **True** in a macro or procedure initiates the button's <u>Click</u> event.

For a **ComboBox**, changing the contents of **Value** does not change the value of **BoundColumn**. To add or delete entries in a **ComboBox**, you can use the **AddItem** or **RemoveItem** method.

For a **TextBox**, any value you assign to the **<u>Text</u>** property is also assigned to the **Value** property.

Value cannot be used with a multi-select ListBox.

VerticalScrollBarSide Property

Specifies whether a vertical scroll bar appears on the right or left side of a frame.

Syntax

object.VerticalScrollBarSide [= VerticalScrollBarSide]

The **VerticalScrollBarSide** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.VerticalScrollBarSideOptional. Where the scroll bar should appear.

Settings

ValueDescription0Puts the scroll bar on the right side (default).1Puts the scroll bar on the left side.



Visible Property

Specifies whether an object is visible or hidden.

Syntax

object.Visible [= Boolean]

The **Visible** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the object is visible or hidden.

Settings

The settings for *Boolean* are:

Value Description

True Object is visible (default).False Object is hidden.

Remarks

Use the **Visible** property to control access to information without displaying it. For example, you could use the value of a control on a hidden form as the criteria for a query.

All controls are visible at <u>design time</u>.

WordWrap Property

Indicates whether the contents of a control automatically wrap at the end of a line.

Syntax

object.WordWrap [= Boolean]

The **WordWrap** property syntax has these parts:

PartDescriptionobjectRequired. A valid object.Boolean Optional. Whether the control expands to fit the text.

Settings

The settings for *Boolean* are:

Value Description

True The text wraps (default).False The text does not wrap.

Remarks

For controls that support the **MultiLine** property as well as the **WordWrap** property, **WordWrap** is ignored when **MultiLine** is **False**.

Zoom Property

Specifies how much to change the size of a displayed object.

Syntax

object.Zoom [= Integer]

The **Zoom** property syntax has these parts:

Part Description

object Required. A valid object.

Integer The percentage to increase or decrease the displayed image.

Remarks

The value of the **Zoom** property specifies a percentage of image enlargement or reduction by which an image display should change. Values from 10 to 400 are valid. The value specified is a percentage of the object's original size; thus, a setting of 400 means you want to enlarge the image to four times its original size (or 400 percent), while a setting of 10 means you want to reduce the image to one-tenth of its original size (or 10 percent).

Accelerator, Caption Properties Example

This example changes the <u>Accelerator</u> and <u>Caption</u> properties of a <u>CommandButton</u> each time the user clicks the button by using the mouse or the accelerator key. The Click event contains the code to change the **Accelerator** and **Caption** properties.

To try this example, paste the code into the Script Editor of a form containing a **CommandButton** named CommandButton1. To run the code you need to open the form so the **Open** event will activate.

```
Dim CommandButton1
Sub Item_Open()
 Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
 CommandButton1.Accelerator = "C"
                                            'Set Accelerator key to
End Sub
Sub CommandButton1_Click ()
  If CommandButton1.Caption = "OK" Then
                                             'Check caption, then cha
    CommandButton1.Caption = "Clicked"
    CommandButton1.Accelerator = "C"
                                               'Set Accelerator key t
 Else
    CommandButton1.Caption = "OK"
    CommandButton1.Accelerator = "0"
                                              'Set Accelerator key t
  End If
End Sub
```

Accessing a Page Example

The following example accesses an individual page of a **MultiPage** in several ways:

- Using the <u>Pages</u> collection with a numeric index.
- Using the name of the individual page in the **MultiPage**.
- Using the **<u>SelectedItem</u>** property.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains a **MultiPage** named MultiPage1 and a **CommandButton** named CommandButton1.

```
Sub CommandButton1 Click
 Dim PageName
 Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").MultiP
  For i = 0 To MultiPage1.Count - 1
    'Use index (numeric or string)
    MsgBox "MultiPage1.Pages(i).Caption = " & MultiPage1.Pages(i).Ca
    MsgBox "MultiPage1.Pages.Item(i).Caption = " & MultiPage1.Pages.
    'Use Page object without referring to Pages collection
    If i = 0 Then
     MsgBox "MultiPage1.Page1.Caption = " & MultiPage1.Page1.Captio
    ElseIf i = 1 Then
      MsqBox "MultiPage1.Page2.Caption = " & MultiPage1.Page2.Captio
    End If
    'Use SelectedItem Property
    MultiPage1.Value = i
   MsgBox "MultiPage1.SelectedItem.Caption = " & MultiPage1.Selecte
 Next
End Sub
```

Accessing a Tab Example

The following example accesses an individual tab of a **<u>TabStrip</u>** in several ways:

- Using the <u>Tabs</u> collection with a numeric index.
- Using the name of the individual **Tab**.
- Using the **<u>SelectedItem</u>** property.

```
Sub Item_Open()
    Dim TabStrip1
    Dim TabName
     Set TabStrip1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
    For i = 0 To TabStrip1.Count - 1
        'Using index (numeric or string)
        MsgBox "TabStrip1.Tabs(i).Caption = " & TabStrip1.Tabs(i).Ca
        MsqBox "TabStrip1.Tabs.Item(i).Caption = " & TabStrip1.Tabs.
        'Use Tab object without referring to Tabs collection
        If i = 0 Then
            MsgBox "TabStrip1.Tab1. Caption = " & TabStrip1.Tab1.Cap
        ElseIf i = 1 Then
            MsgBox "TabStrip1.Tab2. Caption = " & TabStrip1.Tab2.Cap
        End If
        'Use SelectedItem Property
        TabStrip1.Value = i
        MsgBox " TabStrip1.SelectedItem.Caption = " & TabStrip1.Sele
    Next
End Sub
```

Accessing Controls Through the Controls Collection Example

The following example accesses individual controls from the **Controls** collection using a For Each...Next loop. When the user presses CommandButton1, the other controls are placed in a column along the left edge of the form using the **Move** method.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Dim CtrlHeight
Dim CtrlTop
Dim CtrlGap
Dim CommandButton1
Sub Item_Open()
  Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
  CtrlHeight = 20
  CtrlGap = 5
  CommandButton1.Caption = "Click to move controls"
  CommandButton1.AutoSize = True
  CommandButton1.Left = 120
  CommandButton1.Top = CtrlTop
End Sub
Sub CommandButton1 Click()
  Dim MyControl
  Set AllControls = Item.GetInspector.ModifiedFormPages("P.2").Contr
  CtrlTop = 5
  For i = 0 to AllControls.Count - 1
    Set MyControl = AllControls(i)
    If MyControl.Name = "CommandButton1" Then
    'Don't move or resize this control.
    Else
```

'Move method using unnamed arguments (left, top, width, height MyControl.Move 5, CtrlTop, ,CtrlHeight 'Calculate top coordinate for next control CtrlTop = CtrlTop + CtrlHeight + CtrlGap End If Next End Sub



Adding a Control Example

The following example uses the <u>Add</u> method to add a control to a form at <u>run</u> <u>time</u>.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains:

• A <u>CommandButton</u> named CommandButton1.

```
Dim Mycmd
Sub CommandButton1_Click()
Set Mycmd = Item.GetInspector.ModifiedFormPages("P.2").Controls.A
Mycmd.Left = 18
Mycmd.Top = 150
Mycmd.Width = 175
Mycmd.Width = 175
Mycmd.Height = 20
Mycmd.Caption = "This is fun." & Mycmd.Name
```

End Sub

Adding a Control to a MultiPage Example

The following example uses the <u>Add</u>, <u>Clear</u>, and <u>Remove</u> methods to add and remove a control to a <u>Page</u> of a <u>MultiPage</u> at run time.

- A **MultiPage** named MultiPage1.
- Three <u>CommandButton</u> controls named CommandButton1 through CommandButton3.

```
Dim MyTextBox
Dim MultiPage1
Sub Item_Open()
  Set MyPage = Item.GetInspector.ModifiedFormPages("P.2")
  Set MultiPage1 = MyPage.MultiPage1
  MyPage.CommandButton1.Caption = "Add control"
  MyPage.CommandButton2.Caption = "Clear controls"
  MyPage.CommandButton3.Caption = "Remove control"
End Sub
Sub CommandButton1 Click()
  Set MyTextBox = MultiPage1.Pages(0).Controls.Add("Forms.TextBox.1"
End Sub
Sub CommandButton2_Click()
  MultiPage1.Pages(0).Controls.Clear
End Sub
Sub CommandButton3 Click()
  If MultiPage1.Pages(0).Controls.Count > 0 Then
    MultiPage1.Pages(0).Controls.Remove "MyTextBox"
  End If
End Sub
```

Alignment Property Example

The following example demonstrates the <u>Alignment</u> property used with several <u>OptionButton</u> controls. In this example, the user can change the alignment by clicking a <u>ToggleButton</u>.

- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ToggleButton** named ToggleButton1.

```
Dim OptionButton1
Dim OptionButton2
Dim ToggleButton1
Sub Item_Open()
     Set OptionButton1 = Item.GetInspector.ModifiedFormPages.Item("F
     Set OptionButton2 = Item.GetInspector.ModifiedFormPages.Item("F
     Set ToggleButton1 = Item.GetInspector.ModifiedFormPages.Item("P
    OptionButton1.Alignment = 0 'fmAlignmentLeft
    OptionButton2.Alignment = 0 'fmAlignmentLeft
    OptionButton1.Caption = "Alignment with AutoSize"
    OptionButton2.Caption = "Choice 2"
    OptionButton1.AutoSize = True
    OptionButton2.AutoSize = True
    ToggleButton1.Caption = "Left Align"
    ToggleButton1.WordWrap = True
    ToggleButton1.Value = True
End Sub
Sub ToggleButton1 Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Left Align"
        OptionButton1.Alignment = 0 'fmAlignmentLeft
        OptionButton2.Alignment = 0 'fmAlignmentLeft
    Else
        ToggleButton1.Caption = "Right Align"
        OptionButton1.Alignment = 1 'fmAlignmentRight
```

```
OptionButton2.Alignment = 1 'fmAlignmentRight
End If
End Sub
```

AutoSize Property Example

The following example demonstrates the effects of the <u>AutoSize</u> property with a single-line <u>TextBox</u> and a multiline **TextBox**. The user can enter text into either of the **TextBox** controls and turn **AutoSize** on or off independently of the contents of the **TextBox**. This code sample also uses the <u>Text</u> property.

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **<u>ToggleButton</u>** named ToggleButton1.

```
Dim ToggleButton1
Dim TextBox1
Dim TextBox2
Sub Item Open()
 Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").Tog
  Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").TextBox1
  Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").TextBox2
  TextBox1.Text = "Single-line TextBox. Type your text here."
  TextBox2.MultiLine = True
  TextBox2.Text = "Multi-line TextBox. Type your text here. Use SHIF
  ToggleButton1.Value = True
  ToggleButton1.Caption = "AutoSize On"
  TextBox1.AutoSize = True
  TextBox2.AutoSize = True
End Sub
Sub ToggleButton1_Click()
  If ToggleButton1.Value = True Then
    ToggleButton1.Caption = "AutoSize On"
   TextBox1.AutoSize = True
    TextBox2.AutoSize = True
  Else
    ToggleButton1.Caption = "AutoSize Off"
    TextBox1.AutoSize = False
    TextBox2.AutoSize = False
```

End If End Sub

Bold, Italic, Size, StrikeThrough, Underline, Weight Properties Example

The following example demonstrates a **Font** object and the **Bold**, **Italic**, **Size**, **StrikeThrough**, **Underline**, and **Weight** properties related to fonts. You can manipulate font properties of an object directly or by using an alias, as this example also shows.

- A Label named Label1.
- Four **ToggleButton** controls named ToggleButton1 through ToggleButton4.
- A second **Label** and a **<u>TextBox</u>** named Label2 and TextBox1.

```
Dim MyFont
Dim ToggleButton1
Dim ToggleButton2
Dim ToggleButton3
Dim ToggleButton4
Dim Label1
Dim Label2
Dim TextBox1
Sub Item Open()
  Set MyPage = Item.GetInspector.ModifiedFormPages("P.2")
  Set ToggleButton1 = MyPage.ToggleButton1
  Set ToggleButton2 = MyPage.ToggleButton2
  Set ToggleButton3 = MyPage.ToggleButton3
  Set ToggleButton4 = MyPage.ToggleButton4
  Set Label1 = MyPage.Label1
  Set Label2 = MyPage.Label2
  Set TextBox1 = MyPage.TextBox1
  Set MyFont = Label1.Font
  ToggleButton1.Value = True
  ToggleButton1.Caption = "Bold On"
```

```
'Set size of Label1
  Label1.AutoSize = True
  Label1.AutoSize = False
  ToggleButton2.Value = False
  ToggleButton2.Caption = "Italic Off"
  ToggleButton3.Value = False
  ToggleButton3.Caption = "StrikeThrough Off"
  ToggleButton4.Value = False
  ToggleButton4.Caption = "Underline Off"
  Label2.Caption = "Font Weight"
  TextBox1.Text = Label1.Font.Weight
  TextBox1.Enabled = False
End Sub
Sub ToggleButton1 Click()
  If ToggleButton1.Value = True Then
    MyFont.Bold = True
                                  'Using MyFont alias to control fon
    ToggleButton1.Caption = "Bold On"
    MvFont.Size = 22
                                   'Increase the font size
 Else
    MyFont.Bold = False
    ToggleButton1.Caption = "Bold Off"
    MyFont.Size = 8
                                  'Return font size to initial size
  End If
  TextBox1.Text = CStr(MyFont.Weight) 'Bold and Weight are related
End Sub
Sub ToggleButton2_Click()
  If ToggleButton2.Value = True Then
    Label1.Font.Italic = True
                                          'Using Label1.Font directly
    ToggleButton2.Caption = "Italic On"
 Else
    Label1.Font.Italic = False
    ToggleButton2.Caption = "Italic Off"
  End If
End Sub
Sub ToggleButton3_Click()
  If ToggleButton3.Value = True Then
    Label1.Font.Strikethrough = True
                                                 'Using Label1.Font d
    ToggleButton3.Caption = "StrikeThrough On"
  Else
    Label1.Font.Strikethrough = False
    ToggleButton3.Caption = "StrikeThrough Off"
```

```
End If
End Sub
Sub ToggleButton4_Click()
If ToggleButton4.Value = True Then
MyFont.Underline = True 'Using MyFont alias for Label
ToggleButton4.Caption = "Underline On"
Else
Label1.Font.Underline = False
ToggleButton4.Caption = "Underline Off"
End If
End If
End Sub
```

Border, Color Enhancements Example

The following example demonstrates the **BorderStyle** and **SpecialEffect** properties, showing each border available through these properties. The example also demonstrates how to control color settings by using the **BackColor**, **BackStyle**, **BorderColor**, and **ForeColor** properties.

- Six **<u>TextBox</u>** controls named TextBox1 through TextBox6.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2.

```
Dim TextBox1
Dim TextBox2
Dim TextBox3
Dim TextBox4
Dim TextBox5
Dim TextBox6
Dim ToggleButton1
Dim ToggleButton2
Sub Item_Open()
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").TextBox1
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").TextBox2
Set TextBox3 = Item.GetInspector.ModifiedFormPages("P.2").TextBox3
Set TextBox4 = Item.GetInspector.ModifiedFormPages("P.2").TextBox4
Set TextBox5 = Item.GetInspector.ModifiedFormPages("P.2").TextBox5
Set TextBox6 = Item.GetInspector.ModifiedFormPages("P.2").TextBox6
Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").Toggl
Set ToggleButton2 = Item.GetInspector.ModifiedFormPages("P.2").Toggl
'Initialize each TextBox with a border style or special effect,
'and foreground and background colors
'TextBox1 initially uses a borderstyle
TextBox1.Text = "BorderStyle-Single"
TextBox1.BorderStyle = 1
TextBox1.BorderColor = RGB(255, 128, 128)
                                             'Color - Salmon
TextBox1.ForeColor = RGB(255, 255, 0)
                                             'Color - Yellow
```

```
TextBox1.BackColor = RGB(0, 128, 64) 'Color - Green #2
'TextBoxes 2 through 6 initially use special effects
TextBox2.Text = "Flat"
TextBox2.SpecialEffect = 0
TextBox2.ForeColor = RGB(64, 0, 0)
TextBox2.BackColor = RGB(0, 0, 255)
                                               'Color - Brown
                                               'Color - Blue
'Ensure the background style for TextBox2 is initially opaque.
TextBox2.BackStyle = 1
TextBox3.Text = "Etched"
TextBox3.SpecialEffect = 3
TextBox3.ForeColor = RGB(128, 0, 255) 'Color - Purple
TextBox3.BackColor = RGB(0, 255, 255) 'Color - Cyan
'Define BorderColor for later use (when borderstyle=fmBorderStyleSin
                                        'Color - Black
TextBox3.BorderColor = RGB(0, 0, 0)
TextBox4.Text = "Bump"
TextBox4.SpecialEffect = 6
                                            'Color - Magenta
'Color - Navy blue
TextBox4.ForeColor = RGB(255, 0, 255)
TextBox4.BackColor = RGB(0, 0, 100)
TextBox5.Text = "Raised"
TextBox5.SpecialEffect = 1
TextBox5.ForeColor = RGB(255, 0, 0) 'Color - Red
TextBox5.BackColor = RGB(128, 128, 128) 'Color - Gray
TextBox6.Text = "Sunken"
TextBox6.SpecialEffect = 2
TextBox6.ForeColor = RGB(0, 64, 0)
TextBox6.BackColor = RGB(0, 255, 0)
                                                'Color - Olive
                                               'Color - Green #1
ToggleButton1.Caption = "Swap styles"
ToggleButton2.Caption = "Transparent/Opaque background"
End Sub
Sub ToggleButton1_Click()
'Swap borders between TextBox1 and TextBox3
If ToggleButton1.Value = True Then
    'Change TextBox1 from BorderStyle to Etched
    TextBox1.Text = "Etched"
    TextBox1.SpecialEffect = 3
    'Change TextBox3 from Etched to BorderStyle
    TextBox3.Text = "BorderStyle-Single"
    TextBox3.BorderStyle = 1
Else
```

```
'Change TextBox1 back to BorderStyle
   TextBox1.Text = "BorderStyle-Single"
    TextBox1.BorderStyle = 1
    'Change TextBox3 back to Etched
   TextBox3.Text = "Etched"
   TextBox3.SpecialEffect = 3
End If
End Sub
Sub ToggleButton2_Click()
'Set background to Opaque or Transparent
If ToggleButton2.Value = True Then
    'Change TextBox2 to a transparent background
    TextBox2.BackStyle = 0
Else
    'Change TextBox2 back to opaque background
   TextBox2.BackStyle = 1
End If
End Sub
```

BoundColumn Property Example

The following example demonstrates how the **BoundColumn** property influences the value of a **ListBox**. The user can choose to set the value of the **ListBox** to the index value of the specified row, or to a specified column of data in the **ListBox**.

- A ListBox named ListBox1.
- A Label named Label1.
- Three **OptionButton** controls named OptionButton1, OptionButton2, and OptionButton3.

```
Dim Listbox1
Dim OptionButton1
Dim OptionButton2
Dim OptionButton3
Dim Label1
Sub Item_Open
 Set Listbox1 = Item.GetInspector.ModifiedFormPages("P.2").Listbox1
  Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").Opt
  Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").Opt
  Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").Opt
  Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Label1
  Listbox1.ColumnCount = 2
  Listbox1.AddItem "Item 1, Column 1"
 Listbox1.List(0, 1) = "Item 1, Column 2"
  Listbox1.AddItem "Item 2, Column 1"
 Listbox1.List(1, 1) = "Item 2, Column 2"
 Listbox1.Value = "Item 1, Column 1"
 OptionButton1.Caption = "List Index"
  OptionButton2.Caption = "Column 1"
  OptionButton3.Caption = "Column 2"
  OptionButton2.Value = True
End Sub
Sub OptionButton1_Click
```

```
Listbox1.BoundColumn = 0
Label1.Caption = Listbox1.Value
End Sub
Sub OptionButton2_Click
Listbox1.BoundColumn = 1
Label1.Caption = Listbox1.Value
End Sub
Sub OptionButton3_Click
Listbox1.BoundColumn = 2
Label1.Caption = Listbox1.Value
End Sub
Sub Listbox1_Click
Label1.Caption = Listbox1.Value
End Sub
```

ClientHeight, ClientLeft, ClientTop, ClientWidth Properties Example

The following example sets the dimensions of an **Image** to the size of a **TabStrip's** client area when the user clicks a **CommandButton**. This code sample uses the following properties: **Height**, **Left**, **Top**, **Width**, **ClientHeight**, **ClientLeft**, **ClientTop**, and **ClientWidth**.

- A **CommandButton** named CommandButton1.
- A **TabStrip** named TabStrip1.
- An **Image** named Image1.

```
Dim CommandButton1
Dim Image1
Dim TabStrip1
Sub Item Open
 Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
  Set Image1 = Item.GetInspector.ModifiedFormPages("P.2").Image1
 Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").TabStri
 CommandButton1.Caption = "Size Image to Tab Area"
 CommandButton1.WordWrap = True
 CommandButton1.AutoSize = True
End Sub
Sub CommandButton1 Click
  Image1.ZOrder (fmFront) 'Place Image in front of TabStrip
'Client Left and ClientTop are measured from the edge of the TabStri
'not from the edges of the form containing the TabStrip.
  Image1.Left = TabStrip1.Left + TabStrip1.ClientTop
  Image1.Top = TabStrip1.Top + TabStrip1.ClientTop
  Image1.Width = TabStrip1.ClientWidth
  Image1.Height = TabStrip1.ClientHeight
End Sub
```

Column Property Example

The following example loads a two-dimensional array with data and, in turn, loads two **ListBox** controls using the **Column** and **List** properties. Note that the **Column** property transposes the array elements during loading.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains two **ListBox** controls named ListBox1 and ListBox2.

```
Dim MyArray(6,3)
Sub Item Open()
    Dim i
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").ListBo
    Set ListBox2 = Item.GetInspector.ModifiedFormPages("P.2").ListBo
    ListBox1.ColumnCount = 3
ListBox2.ColumnCount = 6
                                     'The 1st list box contains 3 dat
                                     'The 2nd box contains 6 data col
    'Load integer values into first column of MyArray
    For i = 0 To 5
        MyArray(i, 0) = i
    Next
    'Load columns 2 and three of MyArray
    MyArray(0, 1) = "Zero"
   MyArray(1, 1) = "One"
    MyArray(2, 1) = "Two"
   MyArray(3, 1) = "Three"
    MyArray(4, 1) = "Four"
    MyArray(5, 1) = "Five"
    MyArray(0, 2) = "Zero"
    MyArray(1, 2) = "Un ou Une"
    MyArray(2, 2) = "Deux"
   MyArray(3, 2) = "Trois"
    MyArray(4, 2) = "Quatre"
    MyArray(5, 2) = "Cinq"
    'Load data into ListBox1 and ListBox2
    ListBox1.List() = MyArray
    ListBox2.Column() = MyArray
```

End Sub

ColumnWidths Property Example

The following example uses the <u>ColumnWidths</u> property to change the column widths of a multicolumn <u>ListBox</u>. The example uses three <u>TextBox</u> controls to specify the individual column widths and uses the Exit event to specify the units of measure of each **TextBox**.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **ListBox** named ListBox1.
- Three custom text fields named Text1, Text2, and Text3.
- Three **TextBox** controls named TextBox1, TextBox2, and TextBox3 that are bound to the custom text fields above.
- A <u>CommandButton</u> named CommandButton1.

Try entering the value 0 to hide a column.

```
Dim MyArray(2, 3)
Dim ListBox1
Dim TextBox1
Dim TextBox2
Dim TextBox3
Dim CommandButton1
Sub Item Open()
Dim i, j, Rows
Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").ListBox1
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").TextBox1
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").TextBox2
Set TextBox3 = Item.GetInspector.ModifiedFormPages("P.2").TextBox3
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Comm
ListBox1.ColumnCount = 3
Rows = 2
For j = 0 To ListBox1.ColumnCount - 1
    For i = 0 To Rows - 1
        MyArray(i, j) = "Row " & i & ", Column " & j
    Next
```

'Load MyArray into ListBox1 ListBox1.List() = MyArray'1-inch columns initially TextBox1.Text = "1 in" TextBox2.Text = "1 in" TextBox3.Text = "1 in" End Sub Sub CommandButton1_Click() 'ColumnWidths requires a value for each column separated by semi ListBox1.ColumnWidths = TextBox1.Text & ";" & TextBox2.Text & "; End Sub Sub Item_CustomPropertyChange(ByVal Name) msabox Name Select Case Name Case "Text1" 'ColumnWidths accepts points (no units), inches or centimeters; ma If Not (InStr(TextBox1.Text, "in") > 0 Or InStr(TextBox1.Text, " TextBox1.Text = TextBox1.Text & " in" End If Case "Text2" 'ColumnWidths accepts points (no units), inches or centimeters; If Not (InStr(TextBox2.Text, "in") > 0 Or InStr(TextBox2.Text, " TextBox2.Text = TextBox2.Text & " in" End If Case "Text3" 'ColumnWidths accepts points (no units), inches or centimeters; If Not (InStr(TextBox3.Text, "in") > 0 Or InStr(TextBox3.Text, " TextBox3.Text = TextBox3.Text & " in" End If End Select End Sub

Next

ControlTipText Property Example

The following example defines the <u>**ControlTipText**</u> property for three <u>**CommandButton**</u> controls and two <u>**Page**</u> objects in a <u>**MultiPage**</u>.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **MultiPage** named MultiPage1.
- Three **CommandButton** controls named CommandButton1 through CommandButton3.

Note For an individual **Page** of a **MultiPage**, **ControlTipText** becomes enabled when the **MultiPage** or a control on the current page of the **MultiPage** has the focus.

```
Sub Item_Open()
Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").MultiP
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
Set CommandButton2 = Item.GetInspector.ModifiedFormPages("P.2").Co
Set CommandButton3 = Item.GetInspector.ModifiedFormPages("P.2").Co
MultiPage1.Page1.ControlTipText = "Here in page 1"
MultiPage1.Page2.ControlTipText = "Now in page 2"
CommandButton1.ControlTipText = "And now here's"
CommandButton2.ControlTipText = "a tip from"
CommandButton3.ControlTipText = "your controls!"
End Sub
```



Count Property Example

The following example displays the <u>**Count</u>** property of the **Controls** <u>collection</u> for the form, and the **Count** property identifying the number of pages and tabs of each <u>**MultiPage**</u> and <u>**TabStrip**</u>.</u>

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. The form can contain any number of controls, with the following restrictions:

- Names of **MultiPage** controls must start with "MultiPage".
- Names of **TabStrip** controls must start with "TabStrip".

Note You can add pages to a **MultiPage** or add tabs to a **TabStrip** while in design mode. Double-click on the control, then right click in the tab area of the control and choose **New Page** from the shortcut menu.

```
Sub Item_Open
Dim Controls
Dim MyControl
Set Controls = Item.GetInspector.ModifiedFormPages.Item("P.2").C
MsgBox "Controls.Count = " & Controls.Count
For i = 0 to Controls.Count -1
Set MyControl = Item.GetInspector.ModifiedFormPages.Item("P.
If (MyControl.Name = "MultiPage1") Then
MsgBox MyControl.Name & ".Pages.Count = " & MyControl.Pa
ElseIf (MyControl.Name = "TabStrip1") Then
MsgBox MyControl.Name & ".Tabs.Count = " & MyControl.Tab
End If
Next
End Sub
```

Cut and Paste From a Page Example

The following example uses the <u>Add</u>, <u>Cut</u>, and <u>Paste</u> methods to cut and paste a control from a <u>Page</u> of a <u>MultiPage</u>. The control involved in the cut and paste operations is dynamically added to the form.

This example assumes the user will add, then cut, then paste the new control.

- Three <u>CommandButton</u> controls named CommandButton1 through CommandButton3.
- A **MultiPage** named MultiPage1.

```
Dim CommandButton1
Dim CommandButton2
Dim CommandButton3
Dim MultiPage1
Dim MyTextBox
Sub CommandButton1_Click()
    Set MyTextBox = MultiPage1.Pages(MultiPage1.Value).Controls.Add(
    CommandButton2.Enabled = True
    CommandButton1.Enabled = False
End Sub
Sub CommandButton2 Click()
    MultiPage1.Pages(MultiPage1.Value).Controls.Cut
    CommandButton3.Enabled = True
    CommandButton2.Enabled = False
End Sub
Sub CommandButton3_Click()
    Dim MyPage
    Set MyPage = MultiPage1.Pages.Item(MultiPage1.Value)
    MyPage.Paste
    CommandButton3.Enabled = False
End Sub
Sub Item_Open()
```

```
Set CommandButton1 = Item.GetInspector.ModifiedFormPages.Item("F
Set CommandButton2 = Item.GetInspector.ModifiedFormPages.Item("F
Set CommandButton3 = Item.GetInspector.ModifiedFormPages.Item("F
Set MultiPage1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
CommandButton1.Caption = "Add"
CommandButton2.Caption = "Cut"
CommandButton3.Caption = "Paste"
CommandButton1.Enabled = True
CommandButton1.Enabled = False
CommandButton3.Enabled = False
End Sub
```

Cut and Paste From a TextBox Example

The following example uses the <u>**Cut</u>** and <u>**Paste**</u> methods to cut text from one <u>**TextBox**</u> and paste it into another **TextBox**.</u>

- Two **TextBox** controls named TextBox1 and TextBox2.
- A <u>CommandButton</u> named CommandButton1.

```
Dim TextBox1
Dim TextBox2
Dim CommandButton1
Sub Item Open()
    Set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    Set TextBox2 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages.Item("F
    TextBox1.Text = "From TextBox1!"
    TextBox2.Text = "Hello "
    CommandButton1.Caption = "Cut and Paste"
    CommandButton1.AutoSize = True
End Sub
Sub CommandButton1 Click()
    TextBox2.SelStart = 0
    TextBox2.SelLength = TextBox2.TextLength
    TextBox2.Cut
    TextBox1.SetFocus
    TextBox1.SelStart = 0
    TextBox1.Paste
    TextBox2.SelStart = 0
End Sub
```

Cycle Property Example

The following example defines the <u>Cycle</u> property for a <u>Frame</u> and two <u>Page</u> objects in a <u>MultiPage</u>.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **Frame** named Frame1.
- A **MultiPage** named MultiPage1 that contains two objects named Page1 and Page2.
- Two <u>CommandButton</u> controls named CommandButton1 and CommandButton2.

In the form, the **Frame**, and each **Page** of the **MultiPage**, place a couple of controls, so you can see how **Cycle** affects the tab order of the **Frame** and **MultiPage**.

The user should tab through the controls to observe how **Cycle** affects the tab order. Pressing CommandButton1 extends the tab order to include controls in the **Frame** and **Page** objects. Pressing CommandButton2 restricts the tab order.

```
Dim Frame1
Dim MultiPage1
Sub Item_Open()
  Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Frame1
  Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").MultiP
  RestrictCycles
End Sub
Sub RestrictCycles()
  'Limit tab order for the Frame and Page objects
  Frame1.Cycle = 2
  MultiPage1.Page1.Cycle = 2
  MultiPage1.Page2.Cycle = 2
End Sub
Sub CommandButton1 Click()
  'Extend tab order subforms (the Frame and Page objects)
  Frame1.Cycle = 0
```

```
MultiPage1.Page1.Cycle = 0
MultiPage1.Page2.Cycle = 0
End Sub
Sub CommandButton2_Click()
RestrictCycles
End Sub
```

DragBehavior, EnterFieldBehavior Properties Example

The following example uses the **DragBehavior** and **EnterFieldBehavior** properties to demonstrate the different effects that you can provide when entering a control and when dragging information from one control to another.

The sample uses two **TextBox** controls. You can set **DragBehavior** and **EnterFieldBehavior** for each control and see the effects of dragging from one control to another.

- A **TextBox** named TextBox1.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2. These controls are associated with TextBox1.
- A **TextBox** named TextBox2.
- Two **ToggleButton** controls named ToggleButton3 and ToggleButton4. These controls are associated with TextBox2.

```
Dim TextBox1, TextBox2
Dim ToggleButton1, ToggleButton2, ToggleButton3, ToggleButton4
Sub Item_Open()
   set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
   set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton2 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton3 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton4 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton4 = Item.GetInspector.ModifiedFormPages("P.2").
   set ToggleButton4 = Item.GetInspector.ModifiedFormPages("P.2").
   TextBox1.Text = "Once upon a time in a land ...,"
   ToggleButton1.Value = True
   ToggleButton1.Caption = "Drag Enabled"
   ToggleButton1.WordWrap = True
   TextBox1.DragBehavior = 1 'fmDragBehaviorEnabled
   ToggleButton2.Value = True
```

```
ToggleButton2.Caption = "Recall Selection"
    ToggleButton2.WordWrap = True
    TextBox1.EnterFieldBehavior = 1 'fmEnterFieldBehaviorRecallSelec
    TextBox2.Text = "XXX, YYYY"
    ToggleButton3.Value = False
    ToggleButton3.Caption = "Drag Disabled"
    ToggleButton3.WordWrap = True
    TextBox2.DragBehavior = 0 'fmDragBehaviorDisabled
    ToggleButton4.Value = False
    ToggleButton4.Caption = "Select All"
    ToggleButton4.WordWrap = True
    TextBox2.EnterFieldBehavior = 0 'fmEnterFieldBehaviorSelectAll
End Sub
Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
       ToggleButton1.Caption = "Drag Enabled"
       TextBox1.DragBehavior = 1 'fmDragBehaviorEnabled
    Else
       ToggleButton1.Caption = "Drag Disabled"
       TextBox1.DragBehavior = 0 'fmDragBehaviorDisabled
    End If
End Sub
Sub ToggleButton2 Click()
    If ToggleButton2.Value = True Then
        ToggleButton2.Caption = "Recall Selection"
        TextBox1.EnterFieldBehavior = 1 'fmEnterFieldBehaviorRecallS
    Else
        ToggleButton2.Caption = "Select All"
        TextBox1.EnterFieldBehavior = 0 'fmEnterFieldBehaviorSelectA
    End If
End Sub
Sub ToggleButton3_Click()
    If ToggleButton3.Value = True Then
       ToggleButton3.Caption = "Drag Enabled"
       TextBox2.DragBehavior = 1 'fmDragBehaviorEnabled
    Else
       ToggleButton3.Caption = "Drag Disabled"
       TextBox2.DragBehavior = 0 'fmDragBehaviorDisabled
    End If
End Sub
Sub ToggleButton4_Click()
    If ToggleButton4.Value = True Then
        ToggleButton4.Caption = "Recall Selection"
        TextBox2.EnterFieldBehavior = 1 'fmEnterFieldBehaviorRecallS
```

```
Else
ToggleButton4.Caption = "Select All"
TextBox2.EnterFieldBehavior = 0 'fmEnterFieldBehaviorSelectAl
End If
End Sub
```

DropButtonStyle, ShowDropButtonWhen Properties Example

The following example demonstrates the different symbols that you can specify for a drop-down arrow in a **ComboBox** or **TextBox**. In this example, the user chooses a drop-down arrow style from a **ComboBox**. This example also uses the **Locked** property. To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A Label named Label1.
- A **TextBox** named TextBox1 placed beneath Label1.

```
Dim TextBox1
Dim ComboBox1
Dim Label1
Sub ComboBox1_Click()
    ComboBox1.DropButtonStyle = ComboBox1.Value
    TextBox1.DropButtonStyle = ComboBox1.Value
End Sub
Sub Item Open()
     Set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").
     Set ComboBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
     Set Label1 = Item.GetInspector.ModifiedFormPages.Item("P.2").Co
    ComboBox1.ColumnCount = 2
    ComboBox1.BoundColumn = 2
    ComboBox1.TextColumn = 1
    ComboBox1.AddItem "Blank Button"
    ComboBox1.List(0, 1) = 0
    ComboBox1.AddItem "Down Arrow"
    ComboBox1.List(1, 1) = 1
    ComboBox1.AddItem "Ellipsis"
    ComboBox1.List(2, 1) = 2
    ComboBox1.AddItem "Underscore"
    ComboBox1.List(3, 1) = 3
```

```
ComboBox1.Value = 0

TextBox1.Text = "TextBox1"

TextBox1.ShowDropButtonWhen = 2 'fmShowDropButtonWhenAlways

TextBox1.Locked = True

Label1.Caption = "TheDropButton also applies to a TextBox."

Label1.AutoSize = True

Label1.WordWrap = False

End Sub
```

DropDown Method Example

The following example uses the **DropDown** method to display the list in a **ComboBox**. The user can display the list of a **ComboBox** by clicking the **CommandButton**.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **CommandButton** named CommandButton1.

Dim ComboBox1

```
Sub CommandButton1_Click()
ComboBox1.DropDown
```

End Sub

```
Sub Item_Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
    ComboBox1.AddItem "Turkey"
    ComboBox1.AddItem "Chicken"
    ComboBox1.AddItem "Duck"
    ComboBox1.AddItem "Goose"
    ComboBox1.AddItem "Grouse"
End Sub
```

Enabled, Locked Properties Example

The following example demonstrates the **Enabled** and **Locked** properties and how they complement each other. This example exposes each property independently with a **CheckBox**, so you observe the settings individually and combined. This example also includes a second **TextBox** so you can copy and paste information between the **TextBox** controls and verify the activities supported by the settings of these properties.

Note You can copy the selection to the Clipboard using CTRL+C and paste using CTRL+V.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Two CheckBox controls named CheckBox1 and CheckBox2.
- A second **TextBox** named TextBox2.

```
Dim TextBox1
Dim TextBox2
Dim CheckBox1
Dim CheckBox2
Sub CheckBox1_Click()
    TextBox2.Text = "TextBox2"
    TextBox1.Enabled = CheckBox1.Value
End Sub
Sub CheckBox2_Click()
    TextBox2.Text = "TextBox2"
    TextBox1.Locked = CheckBox2.Value
End Sub
Sub Item_Open()
     Set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").
     Set TextBox2 = Item.GetInspector.ModifiedFormPages.Item("P.2").
     Set CheckBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
     Set CheckBox2 = Item.GetInspector.ModifiedFormPages.Item("P.2")
    TextBox1.Text = "TextBox1"
    TextBox1.Enabled = True
```

TextBox1.Locked = False

CheckBox1.Caption = "Enabled" CheckBox1.Value = True

CheckBox2.Caption = "Locked" CheckBox2.Value = False

TextBox2.Text = "TextBox2"
End Sub

EnterKeyBehavior Property Example

The following example uses the **EnterKeyBehavior** property to control the effect of ENTER in a **TextBox**. In this example, the user can specify either a single-line or multiline **TextBox**.

- A **TextBox** named TextBox1.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2.

```
Dim TextBox1
Dim ToggleButton1
Dim ToggleButton2
Sub Item_Open()
    set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    set ToggleButton1 = Item.GetInspector.ModifiedFormPages.Item("P.
    set ToggleButton2 = Item.GetInspector.ModifiedFormPages.Item("P.
    TextBox1.EnterKeyBehavior = True
    ToggleButton1.Caption = "EnterKeyBehavior is True"
    ToggleButton1.Width = 70
    ToggleButton1.Value = True
    TextBox1.MultiLine = True
   ToggleButton2.Caption = "MultiLine is True"
    ToggleButton2.Width = 70
    ToggleButton2.Value = True
   TextBox1.Height = 100
   TextBox1.WordWrap = True
    TextBox1.Text = "Type your text here. If EnterKeyBehavior is Tru
    " press Enter to start a new line. Otherwise, press SHIFT+ENTER.
End Sub
Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        TextBox1.EnterKeyBehavior = True
        ToggleButton1.Caption = "EnterKeyBehavior is True"
    Else
        TextBox1.EnterKeyBehavior = False
```

```
ToggleButton1.Caption = "EnterKeyBehavior is False"
End If
End Sub
Sub ToggleButton2_Click()
If ToggleButton2.Value = True Then
TextBox1.MultiLine = True
ToggleButton2.Caption = "MultiLine TextBox"
Else
TextBox1.MultiLine = False
ToggleButton2.Caption = "Single-line TextBox"
End If
End Sub
```

GroupName Property Example

The following example uses the <u>GroupName</u> property to create two groups of <u>OptionButton</u> controls on the same form.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains five **OptionButton** controls named OptionButton1 through OptionButton5.

```
Sub Item_Open()
    set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C
    set OptionButton4 = Item.GetInspector.ModifiedFormPages("P.2").C
    set OptionButton5 = Item.GetInspector.ModifiedFormPages("P.2").C
    OptionButton1.Caption = "Widgets"
    OptionButton2.Caption = "Widgets"
    OptionButton3.Caption = "Widgets"
    OptionButton1.GroupName = "Widgets"
    OptionButton2.GroupName = "Widgets"
    OptionButton3.GroupName = "Widgets"
    OptionButton4.Caption = "Gadgets-Group2"
   OptionButton5.Caption = "Gadgets-Group2"
    OptionButton4.GroupName = "Gadgets-Group2"
    OptionButton5.GroupName = "Gadgets-Group2"
End Sub
```

Index Property Example

The following example uses the **Index** property to change the order of the pages and tabs in a **MultiPage** and **TabStrip**. The user chooses CommandButton1 to move the third page and tab to the front of the **MultiPage** and **TabStrip**. The user chooses CommandButton2 to move the selected page and tab to the back of the **MultiPage** and **TabStrip**.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains:

- Two <u>CommandButton</u> controls named CommandButton1 and CommandButton2.
- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.

```
Dim MyPageOrTab
Dim MultiPage1
Dim TabStrip1
Sub CommandButton1_Click()
'Move third page and tab to front of control
    MultiPage1.page3.Index = 0
    TabStrip1.Tab3.Index = 0
End Sub
Sub CommandButton2_Click()
'Move selected page and tab to back of control
    Set MyPageOrObject = MultiPage1.SelectedItem
    MsgBox "MultiPage1.SelectedItem = " & MultiPage1.SelectedItem.Na
   MyPageOrObject.Index = 4
    Set MyPageOrObject = TabStrip1.SelectedItem
   MsgBox "TabStrip1.SelectedItem = " & TabStrip1.SelectedItem.Capt
    MyPageOrObject.Index = 4
End Sub
Sub Item_Open()
     Set MultiPage1 = Item.GetInspector.ModifiedFormPages.Item("P.2"
     Set TabStrip1 = Item.GetInspector.ModifiedFormPages.Item("P.2")
     Set CommandButton1 = Item.GetInspector.ModifiedFormPages.Item("
     Set CommandButton2 = Item.GetInspector.ModifiedFormPages.Item("
```

```
MultiPage1.Width = 200
MultiPage1.Pages.Add
MultiPage1.Pages.Add
TabStrip1.Width = 200
TabStrip1.Tabs.Add
TabStrip1.Tabs.Add
TabStrip1.Tabs.Add
CommandButton1.Caption = "Move third page/tab to front"
CommandButton1.Width = 120
CommandButton2.Caption = "Move selected item to back"
CommandButton2.Width = 120
End Sub
```

InsideHeight, InsideWidth Properties Example

The following example uses the **InsideHeight** and **InsideWidth** properties to resize a **CommandButton**. The user clicks the **CommandButton** to resize it.

Note InsideHeight and InsideWidth are read-only properties.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

• A **CommandButton** named CommandButton1.

```
Dim Resize
Sub Item_Open()
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
Resize = 0.75
CommandButton1.Caption = "Resize Button"
End Sub
Sub CommandButton1_Click()
Set Form = Item.GetInspector.ModifiedFormPages("P.2")
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
CommandButton1.Move 10, 10, Form.InsideWidth * Resize, Form.Insi
CommandButton1.Caption = "Button resized using InsideHeight and
End Sub
```

Item Method Example

The following example uses the <u>Item</u> method to access individual members of the **Controls** and <u>Pages</u> collections. The user chooses an option button for either the **Controls** collection or the <u>MultiPage</u>, and then clicks the <u>CommandButton</u>. The name of the appropriate control is returned in the <u>Label</u>.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A Label named Label1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **MultiPage** named MultiPage1.

Dim ControlsIndex

```
Sub CommandButton1_Click()
    Set Controls = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set OptionButton1 = Controls("OptionButton1")
    Set OptionButton2 = Controls("OptionButton2")
    Set Label1 = Controls("Label1")
    Set MultiPage1 = Controls("MultiPage1")
    If OptionButton1.Value = True Then
        'Process Controls collection for UserForm
        Set MyControl = Controls.Item(ControlsIndex)
        Label1.Caption = MyControl.Name
        'Prepare index for next control on Userform
        ControlsIndex = ControlsIndex + 1
        If ControlsIndex >= Controls.Count Then
            ControlsIndex = 0
        End If
    ElseIf OptionButton2.Value = True Then
        'Process Current Page of Pages collection
        Set MyControl = MultiPage1.Pages.Item(MultiPage1.Value)
        Label1.Caption = MyControl.Name
    End If
End Sub
```

```
Sub Item_Open()
ControlsIndex = 0
Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
OptionButton1.Caption = "Controls Collection"
OptionButton2.Caption = "Pages Collection"
OptionButton1.Value = True
CommandButton1.Caption = "Get Member Name"
```

End Sub

LargeChange, SmallChange Properties Example

The following example demonstrates the LargeChange and SmallChange properties when used with a stand-alone ScrollBar. The user can set the LargeChange and SmallChange values to any integer in the range of 0 to 100. This example also uses the MaxLength property to restrict the number of characters entered for the LargeChange and SmallChange values.

- A Label named Label1
- A **TextBox** named TextBox1 that is bound to the custom number field named ScrollBarSmallChange
- A Label named Label2
- A **TextBox** named TextBox2 that is bound to the custom number field named ScrollBarLargeChange.
- A **ScrollBar** named ScrollBar1 that is bound to the custom number field named ScrollBarValue.
- A Label named Label3.

```
Sub Item_Open()
```

```
Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set ScrollBar1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set Label2 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set Label3 = Item.GetInspector.ModifiedFormPages("P.2").Controls
ScrollBar1.Min = -1000
```

```
ScrollBar1.Max = 1000
```

```
Label1.Caption = "SmallChange 0 to 100"
ScrollBar1.SmallChange = 1
TextBox1.Text = ScrollBar1.SmallChange
TextBox1.MaxLength = 3
```

```
Label2.Caption = "LargeChange 0 to 100"
    ScrollBar1.LargeChange = 100
    TextBox2.Text = ScrollBar1.LargeChange
    TextBox2.MaxLength = 3
    ScrollBar1.Value = 0
    Label3.Caption = ScrollBar1.Value
End Sub
Sub Item CustomPropertyChange(byval pname)
    Set ScrollBar1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set Label3 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    If pname = "ScrollBarMin" Then
        If IsNumeric(TextBox1.Text) Then
            TempNum = CInt(TextBox1.Text)
            If TempNum >= 0 And TempNum <= 100 Then
                ScrollBar1.SmallChange = TempNum
            Else
                TextBox1.Text = ScrollBar1.SmallChange
            End If
        Else
            TextBox1.Text = ScrollBar1.SmallChange
        End If
    ElseIf pname = "ScrollBarMax" Then
        If IsNumeric(TextBox2.Text) Then
            TempNum = CInt(TextBox2.Text)
            If TempNum >= 0 And TempNum <= 100 Then
                ScrollBar1.LargeChange = TempNum
            Else
                TextBox2.Text = ScrollBar1.LargeChange
            End If
        Else
            TextBox2.Text = ScrollBar1.LargeChange
        End If
    ElseIf pname = "ScrollBarValue" Then
        Label3.Caption = ScrollBar1.Value
    End If
End Sub
```

LineCount, TextLength Properties Example

The following example counts the characters and the number of lines of text in a **<u>TextBox</u>** by using the <u>**LineCount**</u> and <u>**TextLength**</u> properties, and the <u>**SetFocus**</u> method. In this example, the user can type into a **TextBox**, and can retrieve current values of the **LineCount** and **TextLength** properties.

- A **TextBox** named TextBox1.
- A <u>CommandButton</u> named CommandButton1.
- Two Label controls named Label1 and Label2.

```
'Type SHIFT+ENTER to start a new line in the text box.
```

```
Dim CommandButton1
Dim TextBox1
Dim Label1
Dim Label2
Sub CommandButton1 Click()
    'Must first give TextBox1 the focus to get line count
    TextBox1.SetFocus
    Label1.Caption = "LineCount = " & TextBox1.LineCount
    Label2.Caption = "TextLength = " & TextBox1.TextLength
End Sub
Sub Item_Open()
     Set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").
     Set Label1 = Item.GetInspector.ModifiedFormPages.Item("P.2").Co
     Set Label2 = Item.GetInspector.ModifiedFormPages.Item("P.2").Co
     Set CommandButton1 = Item.GetInspector.ModifiedFormPages.Item("
    CommandButton1.WordWrap = True
    CommandButton1.AutoSize = True
    CommandButton1.Caption = "Get Counts"
    Label1.Caption = "LineCount = "
```

```
Label2.Caption = "TextLength = "
TextBox1.MultiLine = True
TextBox1.WordWrap = True
TextBox1.Text = "Enter your text here."
End Sub
```

List Property Example

The following example swaps columns of a multicolumn <u>ListBox</u>. The sample uses the <u>List</u> property in two ways:

- 1. To access and exchange individual values in the **ListBox**. In this usage, **List** has subscripts to designate the row and column of a specified value.
- 2. To initially load the **ListBox** with values from an array. In this usage, **List** has no subscripts.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains a **ListBox** named ListBox1 and a **CommandButton** named CommandButton1.

```
Dim Listbox1
Dim MyArray(6, 3)
Sub Item Open
  Dim i
  Set Listbox1 = Item.GetInspector.ModifiedFormPages("P.2").Listbox1
  Listbox1.ColumnCount = 3
  For i = 0 to 5
    MyArray(i, 0) = i
    MyArray(i, 1) = Rnd
    MyArray(i, 2) = Rnd
  Next
  Listbox1.List() = MyArray
End Sub
Sub CommandButton1 Click
  Dim i
  Dim Temp
  For i = 0 to 5
    Temp = Listbox1.List(i, 0)
    Listbox1.List(i, 0) = Listbox1.List(i, 2)
    Listbox1.List(i, 2) = Temp
  Next
End Sub
```

ListBox Control Example

The following example adds and deletes the contents of a **ListBox** using the **AddItem**, **RemoveItem**, and **SetFocus** methods, and the **ListIndex** and **ListCount** properties.

- A ListBox named ListBox1.
- Two <u>**CommandButton</u>** controls named CommandButton1 and CommandButton2.</u>

```
Dim EntryCount
Dim Listbox1
Sub Item Open()
 Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").ListBox1
 Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
 Set CommandButton2 = Item.GetInspector.ModifiedFormPages("P.2").Co
 EntryCount = 0
 CommandButton1.Caption = "Add Item"
 CommandButton2.Caption = "Remove Item"
End Sub
Sub CommandButton1 Click()
 EntryCount = EntryCount + 1
  ListBox1.AddItem (EntryCount & " - Selection")
End Sub
Sub CommandButton2_Click()
  ListBox1.SetFocus
  'Ensure ListBox contains list items
  If ListBox1.ListCount >= 1 Then
      'If no selection, choose last list item.
      If ListBox1.ListIndex = -1 Then
          ListBox1.ListIndex = ListBox1.ListCount - 1
     End If
      ListBox1.RemoveItem (ListBox1.ListIndex)
  End If
```

End Sub

ListRows Property Example

The following example uses a **SpinButton** to control the number of rows in the drop-down list of a **ComboBox**. The user changes the value of the **SpinButton**, then clicks on the drop-down arrow of the **ComboBox** to display the list.

- A **ComboBox** named ComboBox1.
- A **SpinButton** named SpinButton1 that is bound to a custom number field named SpinButtonValue.
- A Label named Label1.

```
Sub Item Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2").Con
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    For i = 1 To 20
        ComboBox1.AddItem "Choice " & (ComboBox1.ListCount + 1)
    Next
    SpinButton1.Min = 0
    SpinButton1.Max = 12
    SpinButton1.Value = ComboBox1.ListRows
    Label1.Caption = "ListRows = " & SpinButton1.Value
End Sub
Sub Item_CustomPropertyChange(byval pname)
    If pname = "SpinButtonValue" Then
        Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").C
        Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2")
        Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
            ComboBox1.ListRows = SpinButton1.Value
        Label1.Caption = "ListRows = " & SpinButton1.Value
    End If
End Sub
```

ListStyle, MultiSelect Properties Example

The following example uses the **ListStyle** and **MultiSelect** properties to control the appearance of a **ListBox**. The user chooses a value for **ListStyle** using the **ToggleButton** and chooses an **OptionButton** for one of the **MultiSelect** values. The appearance of the **ListBox** changes accordingly, as well as the selection behavior within the **ListBox**.

- A **ListBox** named ListBox1.
- A Label named Label1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **ToggleButton** named ToggleButton1.

```
Sub Item_Open()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    For i = 1 To 8
        ListBox1.AddItem "Choice" & (ListBox1.ListCount + 1)
    Next
    Label1.Caption = "MultiSelect Choices"
    Label1.AutoSize = True
    ListBox1.MultiSelect = 0
                                    '0=fmMultiSelectSingle
    OptionButton1.Caption = "Single entry"
    OptionButton1.Value = True
    OptionButton2.Caption = "Multiple entries"
    OptionButton3.Caption = "Extended entries"
```

```
ToggleButton1.Caption = "ListStyle - Plain"
   ToggleButton1.Value = True
   ToggleButton1.Width = 90
   ToggleButton1.Height = 30
End Sub
Sub OptionButton1 Click()
   Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
   ListBox1.MultiSelect = 0 '0=fmMultiSelectSingle
End Sub
Sub OptionButton2_Click()
   Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
   ListBox1.MultiSelect = 1 '1=fmMultiSelectMulti
End Sub
Sub OptionButton3_Click()
   Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
   ListBox1.MultiSelect = 2 '2=fmMultiSelectExtended
End Sub
Sub ToggleButton1_Click()
   Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
   Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    If ToggleButton1.Value = True Then
       ToggleButton1.Caption = "Plain ListStyle"
       ListBox1.ListStyle = 0 '0=fmListStylePlain
   Else
       ToggleButton1.Caption = "OptionButton or CheckBox"
       ListBox1.ListStyle = 1 '1=fmListStyleOption
   End If
End Sub
```

ListWidth Property Example

The following example uses a **SpinButton** to control the width of the drop-down list of a **ComboBox**. The user changes the value of the **SpinButton**, then clicks on the drop-down arrow of the **ComboBox** to display the list.

- A **ComboBox** named ComboBox1.
- A **SpinButton** named SpinButton1 that is bound to a custom number field named SpinButtonValue.
- A Label named Label1.

```
Sub Item Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2").Con
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    For i = 1 To 20
        ComboBox1.AddItem "Choice " & (ComboBox1.ListCount + 1)
    Next
    SpinButton1.Min = 0
    SpinButton1.Max = 130
    'convert listwidth value from '122 pt' to an integer
    intpos = instr(combobox1.listwidth," ")
    intwidth = left(combobox1.listwidth,intpos-1)
    SpinButton1.Value = intwidth
    SpinButton1.SmallChange = 5
    Label1.Caption = "ListWidth = " & SpinButton1.Value
End Sub
Sub Item CustomPropertyChange(byval pname)
    If pname = "SpinButtonValue" Then
        Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").C
        Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2")
        Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    ComboBox1.ListWidth = SpinButton1.Value
    Label1.Caption = "ListWidth = " & SpinButton1.Value
    End If
```

End Sub

MatchEntry Property Example

The following example uses the **MatchEntry** property to demonstrate character matching that is available for **ComboBox** and **ListBox**. In this example, the user can set the type of matching with the **OptionButton** controls and then type into the **ComboBox** to specify an item from its list.

- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **ComboBox** named ComboBox1.

```
Sub OptionButton1_Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    ComboBox1.MatchEntry = 2
                                   '2=fmMatchEntryNone
End Sub
Sub OptionButton2 Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    ComboBox1.MatchEntry = 0
                                   '0=fmMatchEntryFirstLetter
End Sub
Sub OptionButton3_Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
        ComboBox1.MatchEntry = 1 '1=fmMatchEntryComplete
End Sub
Sub Item_Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C
    For i = 1 To 9
       ComboBox1.AddItem "Choice " & i
    Next
    ComboBox1.AddItem "Chocoholic"
    OptionButton1.Caption = "No matching"
    OptionButton1.Value = True
```

```
OptionButton2.Caption = "Basic matching"
OptionButton3.Caption = "Extended matching"
End Sub
```

MatchFound, MatchRequired Properties Example

The following example uses the **MatchFound** and **MatchRequired** properties to demonstrate additional character matching for **ComboBox**. The matching verification occurs in the **Change** event.

In this example, the user specifies whether the text portion of a **ComboBox** must match one of the listed items in the **ComboBox**. The user can specify whether matching is required by using a **CheckBox** and then type into the **ComboBox** to specify an item from its list.

- A **ComboBox** named ComboBox1 that is bound to the Subject field.
- A **CheckBox** named CheckBox1.

```
Sub CheckBox1 Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    If CheckBox1.Value = True Then
        ComboBox1.MatchRequired = True
        MsgBox "To move the focus from the ComboBox, you must match
    Else
        ComboBox1.MatchRequired = False
        MsgBox " To move the focus from the ComboBox, just tab to or
    End If
End Sub
Sub Item_PropertyChange(byval pname)
    if pname = "Subject" then
        Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").C
        If ComboBox1.MatchRequired = True Then
        'MSForms handles this case automatically
        Else
            If ComboBox1.MatchFound = True Then
                MsgBox "Match Found; matching optional."
```

```
Else
                MsgBox "Match not Found; matching optional."
            End If
        End If
    end if
End Sub
Sub Item_Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
   Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    For i = 1 To 9
            ComboBox1.AddItem "Choice " & i
    Next
    ComboBox1.AddItem "Chocoholic"
   CheckBox1.Caption = "MatchRequired"
    CheckBox1.Value = True
End Sub
```

Max, Min, MaxLength Properties Example

The following example demonstrates the **Max** and **Min** properties when used with a stand-alone **ScrollBar**. The user can set the **Max** and **Min** values to any integer in the range of –1000 to 1000. This example also uses the **MaxLength** property to restrict the number of characters entered for the **Max** and **Min** values.

- A Label named Label1
- A <u>**TextBox**</u> named TextBox1 that is bound to the custom number field named ScrollBarMin.
- A Label named Label2
- A **TextBox** named TextBox2 that is bound to the custom number field named ScrollBarMax.
- A **ScrollBar** named ScrollBar1 that is bound to the custom number field named ScrollBarValue.
- A Label named Label3.

```
Sub Item_Open()
```

```
Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set ScrollBar1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set Label2 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set Label3 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Label1.Caption = "Min -1000 to 1000"
```

```
Label1.Caption = "Min -1000 to 1000"
ScrollBar1.Min = -1000
TextBox1.Text = ScrollBar1.Min
TextBox1.MaxLength = 5
```

```
Label2.Caption = "Max -1000 to 1000"
ScrollBar1.Max = 1000
TextBox2.Text = ScrollBar1.Max
```

```
TextBox2.MaxLength = 5
    ScrollBar1.SmallChange = 1
    ScrollBar1.LargeChange = 100
    ScrollBar1.Value = 0
    Label3.Caption = ScrollBar1.Value
End Sub
Sub Item_CustomPropertyChange(byval pname)
    Set ScrollBar1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set Label3 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    If pname = "ScrollBarMin" Then
        If IsNumeric(TextBox1.Text) Then
                TempNum = CInt(TextBox1.Text)
            If TempNum >= -1000 And TempNum <= 1000 Then
                    ScrollBar1.Min = TempNum
                Else
                    TextBox1.Text = ScrollBar1.Min
            End If
            Else
                TextBox1.Text = ScrollBar1.Min
        End If
    ElseIf pname = "ScrollBarMax" Then
        If IsNumeric(TextBox2.Text) Then
                TempNum = CInt(TextBox2.Text)
                If TempNum >= -1000 And TempNum <= 1000 Then
                    ScrollBar1.Max = TempNum
            Else
                    TextBox2.Text = ScrollBar1.Max
                End If
        Else
                TextBox2.Text = ScrollBar1.Max
        End If
    ElseIf pname = "ScrollBarValue" Then
        Label3.Caption = ScrollBar1.Value
    End If
End Sub
```

MouseIcon, MousePointer Properties Example

The following example demonstrates how to specify a mouse pointer that is appropriate for a specific control or situation. You can assign one of several available mouse pointers using the **MousePointer** property.

This example works in the following ways:

• Choose a mouse pointer from the <u>ListBox</u> to change the mouse pointer associated with the ListBox.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

• A ListBox named ListBox1.

Dim ListBox1

```
Sub Item_Open()
    set ListBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    'Load ListBox with MousePointer choices
    ListBox1.Clear
    ListBox1.AddItem "Default"
    ListBox1.AddItem "Arrow"
    ListBox1.AddItem "Cross"
    ListBox1.AddItem "IBeam"
    ListBox1.AddItem "SizeNESW"
    ListBox1.AddItem "SizeNS"
    ListBox1.AddItem "SizeNWSE"
    ListBox1.AddItem "SizeWE"
    ListBox1.AddItem "UpArrow"
    ListBox1.AddItem "Hourglass"
    ListBox1.AddItem "NoDrop"
    ListBox1.AddItem "AppStarting"
    ListBox1.AddItem "Help"
    ListBox1.AddItem "SizeAll"
End Sub
Sub ListBox1_Click()
```

```
If IsNull(ListBox1.Value) = False Then
    Select Case ListBox1.Value
        Case "Default"
            pointer = 0
                                'Standard pointer.
            Case "Arrow"
            pointer = 1
                                'Arrow.
        Case "Cross"
            pointer = 2
                               'Cross-hair pointer.
        Case "IBeam"
                               'I-beam.
            pointer = 3
        Case "SizeNESW"
                                'Double arrow pointing northeast and
            pointer = 6
        Case "SizeNS"
            pointer = 7
                                'Double arrow pointing north and sout
         Case "SizeNWSE"
            pointer = 8
                                'Double arrow pointing northwest and
        Case "SizeWE"
            pointer = 9
                                'Double arrow pointing west and east.
        Case "UpArrow"
            pointer = 10
                             'Up arrow.
        Case "Hourglass"
            pointer = 11
                             'Hourglass.
        Case "NoDrop"
                             "Not" symbol (circle with a diagonal li
            pointer = 12
        Case "AppStarting"
            pointer = 13
                            'Arrow with an hourglass.
        Case "Help"
            pointer = 14
                            'Arrow with a question mark.
        Case "SizeAll"
                            'Size all cursor (arrows pointing north,
            pointer = 15
    End Select
    ListBox1.MousePointer = pointer
   End If
End Sub
```

Move Method Example for Controls Collection

The following example demonstrates moving all the controls on a form by using the **Move** method with the **Controls** collection. The user clicks on the **CommandButton** to move the controls.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Sub CommandButton1_Click()
  Set Controls = Item.GetInspector.ModifiedFormPages("P.2").Controls
  'Move each control on the form right 25 points and up 25 points.
  Controls.Move 25, -25
End Sub
```

MultiLine, WordWrap, ScrollBars Properties Example

The following example demonstrates the **MultiLine**, **WordWrap**, and **ScrollBars** properties on a **TextBox**.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Four **ToggleButton** controls named ToggleButton1 through ToggleButton4.

To see the entire text placed in the **TextBox**, set **MultiLine** and **WordWrap** to **True** by clicking the **ToggleButton** controls.

When **MultiLine** is **True**, you can enter new lines of text by pressing SHIFT+ENTER.

ScrollBars appears when you manually change the content of the TextBox.

```
Dim ToggleButton1
Dim ToggleButton2
Dim ToggleButton3
Dim ToggleButton4
Dim TextBox1
Sub Item Open
'Initialize TextBox properties and toggle buttons
  Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").Tog
  Set ToggleButton2 = Item.GetInspector.ModifiedFormPages("P.2").Tog
  Set ToggleButton3 = Item.GetInspector.ModifiedFormPages("P.2").Tog
  Set ToggleButton4 = Item.GetInspector.ModifiedFormPages("P.2").Tog
  Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").TextBox1
  TextBox1.Text = "Type your text here. Enter SHIFT+ENTER to move t
  TextBox1.AutoSize = False
  ToggleButton1.Caption = "AutoSize Off"
  ToggleButton1.Value = False
```

```
ToggleButton1.AutoSize = True
  TextBox1.WordWrap = False
  ToggleButton2.Caption = "WordWrap Off"
  ToggleButton2.Value = False
  ToggleButton2.AutoSize = True
  TextBox1.ScrollBars = 0
  ToggleButton3.Caption = "ScrollBars Off"
  ToggleButton3.Value = False
  ToggleButton3.AutoSize = True
  TextBox1.MultiLine = False
  ToggleButton4.Caption = "Single Line"
  ToggleButton4.Value = False
  ToggleButton4.AutoSize = True
End Sub
Sub ToggleButton1 Click
'Set AutoSize property and associated ToggleButton
  If ToggleButton1.Value = True Then
    TextBox1.AutoSize = True
    ToggleButton1.Caption = "AutoSize On"
  Else
    TextBox1.AutoSize = False
    ToggleButton1.Caption = "AutoSize Off"
 End if
End Sub
Sub ToggleButton2 Click
'Set WordWrap property and associated ToggleButton
  If ToggleButton2.Value = True Then
    TextBox1.WordWrap = True
    ToggleButton2.Caption = "WordWrap On"
 Else
    TextBox1.WordWrap = False
    ToggleButton2.Caption = "WordWrap Off"
  End if
End Sub
Sub ToggleButton3_Click
'Set ScrollBars property and associated ToggleButton
  If ToggleButton3.Value = True Then
    TextBox1.ScrollBars = 3
    ToggleButton3.Caption = "ScrollBars On"
 Else
```

```
TextBox1.ScrollBars = 0
ToggleButton3.Caption = "ScrollBars Off"
End if
End Sub
Sub ToggleButton4_Click
'Set MultiLine property and associated ToggleButton
If ToggleButton4.Value = True Then
TextBox1.MultiLine = True
ToggleButton4.Caption = "Multiple Lines"
Else
TextBox1.MultiLine = False
ToggleButton4.Caption = "Single Line"
End if
End Sub
```

MultiSelect, Selected Properties Example

The following example uses the **MultiSelect** and **Selected** properties to demonstrate how the user can select one or more items in a **ListBox**. The user specifies a selection method by choosing an option button and then selects an item(s) from the **ListBox**. The user can display the selected items in a second **ListBox** by clicking the **CommandButton**.

- Two **ListBox** controls named ListBox1 and ListBox2.
- A **CommandButton** named CommandButton1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.

```
Sub CommandButton1_Click()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set ListBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    ListBox2.Clear
    For i = 0 To 9
        If ListBox1.Selected(i) = True Then
            ListBox2.AddItem ListBox1.List(i)
        End If
    Next
End Sub
Sub OptionButton1_Click()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    ListBox1.MultiSelect = 0
                                    '0=fmMultiSelectSingle
End Sub
Sub OptionButton2 Click()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    ListBox1.MultiSelect = 1
                                   '1=fmMultiSelectMulti
End Sub
```

```
Sub OptionButton3 Click()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    ListBox1.MultiSelect = 2 '2=fmMultiSelectExtended
End Sub
Sub Item_Open()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    For i = 0 To 9
       ListBox1.AddItem "Choice " & (ListBox1.ListCount + 1)
    Next
    OptionButton1.Caption = "Single Selection"
    ListBox1.MultiSelect = 0
                                   '0=fmMultiSelectSingle
    OptionButton1.Value = True
    OptionButton2.Caption = "Multiple Selection"
    OptionButton3.Caption = "Extended Selection"
    CommandButton1.Caption = "Show selections"
    CommandButton1.AutoSize = True
End Sub
```



Name Property Example

The following example displays the **Name** property of each control on a form. This example uses the **Controls** <u>collection</u> to cycle through all the controls placed directly on the User form.

To use this example, copy this sample code to the Script Editor of a form. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Sub CommandButton1_Click()
   Set Controls = Item.GetInspector.ModifiedFormPages.Item("P.2").C
   For i = 0 to Controls.Count - 1
        MsgBox "MyControl.Name = " & Controls.Item(i).Name
   Next
End Sub
```

Object Property Example

Assume a new control has a **Top** property that is different from the standard **Top** property in Microsoft Forms. You can use either property, based on the syntax:

- control.Top uses the standard **Top** property.
- control.Object.Top uses the **Top** property from the added control.

Parent Property Example

The following example uses the **<u>Parent</u>** property to refer to the control or form that contains a specific control.

- Two Label controls named Label1 and Label2.
- A <u>CommandButton</u> named CommandButton1.
- One or more additional controls of your choice.

```
Dim MyControl
Dim MyParent
Dim ControlsIndex
Sub Item_Open()
 Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Co
 ControlsIndex = 0
 CommandButton1.Caption = "Get Control and Parent"
 CommandButton1.AutoSize = True
 CommandButton1.WordWrap = True
End Sub
Sub CommandButton1_Click()
  Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Label1
  Set Label2 = Item.GetInspector.ModifiedFormPages("P.2").Label2
  'Process Controls collection for UserForm
  Set MyControl = Item.GetInspector.ModifiedFormPages("P.2").Control
  Set MyParent = MyControl.Parent
  Label1.Caption = MyControl.Name
  Label2.Caption = MyParent.Name
  'Prepare index for next control on Userform
  ControlsIndex = ControlsIndex + 1
  If ControlsIndex >= Item.GetInspector.ModifiedFormPages("P.2").Con
    ControlsIndex = 0
 End If
End Sub
```

Picture, PicturePosition Properties Example

The following example uses a **ComboBox** to show the picture placement options for a control. Each time the user clicks a list choice, the picture and caption are updated on the **CommandButton**. This code sample also uses the **AddItem** method to populate the **ComboBox** choices.

- A Label named Label1.
- A **CommandButton** named CommandButton1 with the **Picture** property set to use an image on your computer.
- A **ComboBox** named ComboBox1.

```
Dim Label1
Dim CommandButton1
Dim ComboBox1
Sub Item Open()
Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Label1
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").Comm
Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").ComboBox1
    Label1.Left = 18
    Label1.Top = 12
    Label1.Height = 12
    Label1.Width = 190
    Label1.Caption = "Select picture placement relative to the capti
    'Add list entries to combo box. The value of each entry matches
    'corresponding ListIndex value in the combo box.
    ComboBox1.AddItem "Left Top"
                                         'ListIndex = 0
    ComboBox1.AddItem "Left Center"
                                         'ListIndex = 1
    ComboBox1.AddItem "Left Bottom"
                                         'ListIndex = 2
    ComboBox1.AddItem "Right Top"
                                         'ListIndex = 3
    ComboBox1.AddItem "Right Center"
                                         'ListIndex = 4
    ComboBox1.AddItem "Right Bottom"
                                         'ListIndex = 5
```

ComboBox1.AddItem "Above Left" 'ListIndex = 6ComboBox1.AddItem "Above Center" 'ListIndex = 7ComboBox1.AddItem "Above Right" 'ListIndex = 8 ComboBox1.AddItem "Below Left" 'ListIndex = 9 'ListIndex = 10 ComboBox1.AddItem "Below Center" ComboBox1.AddItem "Below Right" 'ListIndex = 11ComboBox1.AddItem "Centered" 'ListIndex = 12ComboBox1.Style = 2 'Use drop-down list ComboBox1.BoundColumn = 0'Combo box values are ListI ComboBox1.ListIndex = 0'Set combo box to first ent ComboBox1.Left = 18ComboBox1.Top = 36ComboBox1.Width = 90ComboBox1.ListWidth = 90'Initialize CommandButton1 CommandButton1.Left = 230CommandButton1.Top = 36CommandButton1.Height = 120CommandButton1.Width = 120 'Note: Be sure to refer to have set the CommandButton1 to a bitm 'Note: that is present on your system CommandButton1.PicturePosition = ComboBox1.Value End Sub Sub ComboBox1_Click() Select Case ComboBox1.Value Case 0 'Left Top CommandButton1.Caption = "Left Top" CommandButton1.PicturePosition = 0 Case 1 'Left Center CommandButton1.Caption = "Left Center" CommandButton1.PicturePosition = 1 Case 2 'Left Bottom CommandButton1.Caption = "Left Bottom" CommandButton1.PicturePosition = 2 Case 3 'Right Top CommandButton1.Caption = "Right Top" CommandButton1.PicturePosition = 3 Case 4 'Right Center CommandButton1.Caption = "Right Center" CommandButton1.PicturePosition = 4

Case 5 'Right Bottom CommandButton1.Caption = "Right Bottom" CommandButton1.PicturePosition = 5 Case 6 'Above Left CommandButton1.Caption = "Above Left" CommandButton1.PicturePosition = 6 Case 7 'Above Center CommandButton1.Caption = "Above Center" CommandButton1.PicturePosition = 7 Case 8 'Above Right CommandButton1.Caption = "Above Right" CommandButton1.PicturePosition = 8 Case 9 'Below Left CommandButton1.Caption = "Below Left" CommandButton1.PicturePosition = 9 Case 10 'Below Center CommandButton1.Caption = "Below Center" CommandButton1.PicturePosition = 10 Case 11 'Below Right CommandButton1.Caption = "Below Right" CommandButton1.PicturePosition = 11 Case 12 'Centered CommandButton1.Caption = "Centered" CommandButton1.PicturePosition = 12 End Select

End Sub

ScrollBars, KeepScrollBarsVisible Properties Example

The following example uses the **ScrollBars** and the **KeepScrollBarsVisible** properties to add scroll bars to a page of a **MultiPage** and to a **Frame**. The user chooses an option button that, in turn, specifies a value for **KeepScrollBarsVisible**.

- A **MultiPage** named MultiPage1.
- A **Frame** named Frame1.
- Four <u>OptionButton</u> controls named OptionButton1 through OptionButton4.

```
Sub Item Open()
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton4 = Item.GetInspector.ModifiedFormPages("P.2").C
   MultiPage1.Pages(0).ScrollBars = 3 '3=fmScrollBarsBoth
    MultiPage1.Pages(0).KeepScrollBarsVisible = 0
                                                     '0=fmScrollBars
    Frame1.ScrollBars = 3
                             '3=fmScrollBarsBoth
    Frame1.KeepScrollBarsVisible = 0 '0=fmScrollBarsNone
    OptionButton1.Caption = "No scroll bars"
    OptionButton1.Value = True
    OptionButton2.Caption = "Horizontal scroll bars"
    OptionButton3.Caption = "Vertical scroll bars"
    OptionButton4.Caption = "Both scroll bars"
End Sub
Sub OptionButton1 Click()
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
```

End	MultiPage1.Pages(0).KeepScrollBarsVisible = 0 '0=fmScrollBars Frame1.KeepScrollBarsVisible = 0 '0=fmScrollBarsNonefmScrollE Sub		
Sub	OptionButton2_Click() Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls		
End	MultiPage1.Pages(0).KeepScrollBarsVisible = 1 '1=fmScrollBars Frame1.KeepScrollBarsVisible = 1 '1=fmScrollBarsHorizontal Sub		
Sub	OptionButton3_Click() Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls		
End	<pre>MultiPage1.Pages(0).KeepScrollBarsVisible = 2 '2=fmScrollBars Frame1.KeepScrollBarsVisible = 2 '2=fmScrollBarsVertical Sub</pre>		
Sub	OptionButton4_Click() Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls		
End	MultiPage1.Pages(0).KeepScrollBarsVisible = 3 '3=fmScrollBars Frame1.KeepScrollBarsVisible = 3 '3=fmScrollBarsBoth Sub		

Style Property Example for ComboBox

The following example uses the **Style** property to change the effect of typing in the text area of a **ComboBox**. The user chooses a style by selecting an **OptionButton** control and then types into the **ComboBox** to select an item. When **Style** is *StyleDropDownList*, the user must choose an item from the drop-down list. When **Style** is *StyleDropDownCombo*, the user can type into the text area to specify an item in the drop-down list.

- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ComboBox** named ComboBox1.

```
Sub OptionButton1 Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    ComboBox1.Style = 0
                              '0=fmStyleDropDownCombo
End Sub
Sub OptionButton2_Click()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    ComboBox1.Style = 2 '2=fmStyleDropDownList
End Sub
Sub Item Open()
    Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    For i = 1 To 10
       ComboBox1.AddItem "Choice " & i
    Next
    OptionButton1.Caption = "Select like ComboBox"
    OptionButton1.Value = True
    ComboBox1.Style = 0
                               '0=fmStyleDropDownCombo
    OptionButton2.Caption = "Select like ListBox"
```

End Sub

Style Property Example for MultiPage and TabStrip

The following example uses the **Style** property to specify the appearance of the tabs in **MultiPage** and **TabStrip**. This example also demonstrates using a **Label**. The user chooses a style by selecting an **OptionButton**.

- A Label named Label1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.
- Any control inside the **TabStrip**.
- Any control in each page of the MultiPage.

```
Sub OptionButton1_Click()
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    MultiPage1.Style = 0 '0=fmTabStyleTabs
TabStrip1.Style = 0 '0=fmTabStyleTabs
End Sub
Sub OptionButton2 Click()
    'Note that the page borders are invisible
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    MultiPage1.Style = 1 '1=fmTabStyleButtons
TabStrip1.Style = 1 '1=fmTabStyleButtons
End Sub
Sub OptionButton3_Click()
    'Note that the page borders are invisible and
    'the page body begins where the tabs normally appear.
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    MultiPage1.Style = 2 '2=fmTabStyleNone
```

End	TabStrip1.Style = 2 '2= Sub	fmTabStyleNone	
Sub	<pre>Item_Open() Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls Set OptionButton1 = Item.GetInspector.ModifiedFormPages("P.2").C Set OptionButton2 = Item.GetInspector.ModifiedFormPages("P.2").C Set OptionButton3 = Item.GetInspector.ModifiedFormPages("P.2").C Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Cont</pre>		
	Label1.Caption = "Page/Tab Style" OptionButton1.Caption = "Tabs" OptionButton1.Value = True MultiPage1.Style = 0 '0=fmTabStyleTabs TabStrip1.Style = 0 '0=fmTabStyleTabs		
End	OptionButton2.Caption = "Butto OptionButton3.Caption = "No Ta Sub		

TabFixedHeight, TabFixedWidth Properties Example

The following example uses the **TabFixedHeight** and **TabFixedWidth** properties to set the size of the tabs used in **MultiPage** and **TabStrip**. The user clicks the **SpinButton** controls to adjust the height and width of the tabs within the **MultiPage** and **TabStrip**.

- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.
- A <u>Label</u> named Label1 for the width control.
- A **SpinButton** named SpinButton1 for the width control that is bound to a custom number field named SpinButtonWidth.
- A <u>TextBox</u> named TextBox1 for the width control.
- A Label named Label2 for the height control.
- A **SpinButton** named SpinButton2 for the height control that is bound to a custom number field named SpinButtonHeight.
- A **TextBox** named TextBox2 for the height control.

```
Sub UpdateTabWidth()
```

```
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2").Con
Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
```

```
TextBox1.Text = SpinButton1.Value
TabStrip1.TabFixedWidth = SpinButton1.Value
MultiPage1.TabFixedWidth = SpinButton1.Value
```

```
End Sub
```

```
Sub UpdateTabHeight()
```

```
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set SpinButton2 = Item.GetInspector.ModifiedFormPages("P.2").Con
Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
```

```
TextBox2.Text = SpinButton2.Value
    TabStrip1.TabFixedHeight = SpinButton2.Value
    MultiPage1.TabFixedHeight = SpinButton2.Value
End Sub
Sub Item_Open()
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set SpinButton1 = Item.GetInspector.ModifiedFormPages("P.2").Con
    Set SpinButton2 = Item.GetInspector.ModifiedFormPages("P.2").Con
    Set TabStrip1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set MultiPage1 = Item.GetInspector.ModifiedFormPages("P.2").Cont
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set Label2 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    MultiPage1.Style = 1
                                '1=fmTabStyleButtons
    Label1.Caption = "Tab Width"
    SpinButton1.Min = 0
    SpinButton1.Max = TabStrip1.Width / TabStrip1.Tabs.Count
    SpinButton1.Value = 0
    TextBox1.Locked = True
    UpdateTabWidth
    Label2.Caption = "Tab Height"
    SpinButton2.Min = 0
    SpinButton2.Max = TabStrip1.Height
    SpinButton2.Value = 0
    TextBox2.Locked = True
    UpdateTabHeight
End Sub
Sub Item_CustomPropertyChange(byval pname)
'msgbox pname
    If pname = "SpinButtonWidth" Then
        UpdateTabWidth
    ElseIf pname = "SpinButtonHeight" Then
        UpdateTabHeight
    End If
End Sub
```

TabIndex Property Example

The following example uses the **TabIndex** property to display and set the tab order for individual controls. The user can press TAB to reach the next control in the tab order and to display the **TabIndex** of that control. The user can also click on any control, except a **TextBox** or **ScrollBar**, to display its **TabIndex**. The user can change the **TabIndex** of a control by specifying a new index value in the **TextBox** and clicking CommandButton3. Changing the **TabIndex** for one control also updates the **TabIndex** for other controls in the **Frame**.

- A Label named Label1.
- A **TextBox** named TextBox1.
- A **Frame** named Frame1.
- A **TextBox** in the **Frame** named TextBox2.
- Two **CommandButton** controls in the **Frame** named CommandButton1 and CommandButton2.
- A ScrollBar in the Frame named ScrollBar1.
- A CommandButton (not in the Frame) named CommandButton3.

```
Sub MoveToFront()
Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Temp = Frame1.ActiveControl.TabIndex
For i = 0 To Temp - 1
Frame1.Controls.Item(i).TabIndex = i + 1
Next
Frame1.ActiveControl.TabIndex = 0
TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub
Sub CommandButton3_Click()
Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
If IsNumeric(TextBox1.Text) Then
```

```
Temp = CInt(TextBox1.Text)
        If Temp >= Frame1.Controls.Count Or Temp < 0 Then
            'Entry out of range; move control to front of tab order
            MoveToFront
        ElseIf Temp > Frame1.ActiveControl.TabIndex Then
            'Move entry down the list
            For i = Frame1.ActiveControl.TabIndex + 1 To Temp
                Frame1.Controls.Item(i).TabIndex = i - 1
            Next
            Frame1.ActiveControl.TabIndex = Temp
            TextBox1.Text = Frame1.ActiveControl.TabIndex
        Else
            'Move Entry up the list
            For i = Frame1.ActiveControl.TabIndex - 1 To Temp
                Frame1.Controls.Item(i).TabIndex = i + 1
            Next
            Frame1.ActiveControl.TabIndex = Temp
            TextBox1.Text = Frame1.ActiveControl.TabIndex
        End If
    Else
        'Text entry; move control to front of tab order
        MoveToFront
    End If
End Sub
Sub Item Open()
    Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set CommandButton3 = Item.GetInspector.ModifiedFormPages("P.2").
    Label1.Caption = "TabIndex"
    Frame1.Controls(0).SetFocus
    TextBox1.Text = Frame1.ActiveControl.TabIndex
    Frame1.Cycle = 2 '2=fmCycleCurrentForm
    CommandButton3.Caption = "Set TabIndex"
    CommandButton3.TakeFocusOnClick = False
End Sub
Sub CommandButton1_Click()
    Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub
```

```
Sub CommandButton2_Click()
Set Frame1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub
```

TabStop Property Example

The following example uses the **TabStop** property to control whether a user can press TAB to move the focus to a particular control. The user presses TAB to move the focus among the controls on the form, and then clicks the **ToggleButton** to change **TabStop** for CommandButton1. When **TabStop** is **False**, CommandButton1 will not receive the focus by using TAB.

- A <u>CommandButton</u> named CommandButton1.
- A **ToggleButton** named ToggleButton1.
- One or two other controls, such as an **<u>OptionButton</u>** or **<u>ListBox</u>**.

```
Sub CommandButton1 Click()
    MsgBox "Clicked CommandButton1."
End Sub
Sub ToggleButton1_Click()
    Dim CommandButton1
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    If ToggleButton1 = True Then
        CommandButton1.TabStop = True
        ToggleButton1.Caption = "TabStop On"
    Else
        CommandButton1.TabStop = False
        ToggleButton1.Caption = "TabStop Off"
    End If
End Sub
Sub Item_Open()
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    CommandButton1.Caption = "Show Message"
    ToggleButton1.Caption = "TabStop On"
    ToggleButton1.Value = True
```

ToggleButton1.Width = 90 End Sub

Tag Property Example

The following example uses the **Tag** property to store additional information about each control on the **UserForm**. The user clicks a control and then clicks the **CommandButton**. The contents of **Tag** for the appropriate control are returned in the **TextBox**.

- A **TextBox** named TextBox1.
- A **CommandButton** named CommandButton1.
- A <u>ScrollBar</u> named ScrollBar1.
- A <u>ComboBox</u> named ComboBox1.
- A <u>MultiPage</u> named MultiPage1.

```
Sub CommandButton1 Click()
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set MultiPage1= Item.GetInspector.ModifiedFormPages("P.2").Contr
    TextBox1.Text = Item.GetInspector.ModifiedFormPages("P.2").Activ
End Sub
Sub Item_Open()
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set CommandButton1= Item.GetInspector.ModifiedFormPages("P.2").C
   Set ComboBox1= Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set ScrollBar1= Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set MultiPage1= Item.GetInspector.ModifiedFormPages("P.2").Contr
    TextBox1.Locked = True
   TextBox1.Tag = "Display area for Tag properties."
    TextBox1.AutoSize = True
    CommandButton1.Caption = "Show Tag of Current Control."
    CommandButton1.AutoSize = True
    CommandButton1.WordWrap = True
    CommandButton1.TakeFocusOnClick = False
    CommandButton1.Tag = "Shows tag of control that has the focus."
    ComboBox1.Style = fmStyleDropDownList
```

ComboBox1.Tag = "ComboBox Style is that of a ListBox." ScrollBar1.Max = 100 ScrollBar1.Min = -273 ScrollBar1.Tag = "Max = " & ScrollBar1.Max & " , Min = " & Scrol MultiPage1.Pages.Add MultiPage1.Pages.Add MultiPage1.Tag = "This MultiPage has " & MultiPage1.Pages.Count End Sub

TakeFocusOnClick Property Example

The following example uses the **TakeFocusOnClick** property to control whether a **CommandButton** receives the focus when the user clicks on it. The user clicks a control other than CommandButton1 and then clicks CommandButton1. If **TakeFocusOnClick** is **True**, CommandButton1 receives the focus after it is clicked. The user can change the value of **TakeFocusOnClick** by clicking the **ToggleButton**.

- A **CommandButton** named CommandButton1.
- A **ToggleButton** named ToggleButton1.
- One or two other controls, such as an **OptionButton** or **ListBox**.

```
Sub CommandButton1_Click()
    MsqBox "Watch CommandButton1 to see if it takes the focus."
End Sub
Sub ToggleButton1 Click()
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    If ToggleButton1 = True Then
        CommandButton1.TakeFocusOnClick = True
        ToggleButton1.Caption = "TakeFocusOnClick On"
    Else
        CommandButton1.TakeFocusOnClick = False
        ToggleButton1.Caption = "TakeFocusOnClick Off"
    End If
End Sub
Sub Item_Open()
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    CommandButton1.Caption = "Show Message"
```

```
ToggleButton1.Caption = "TakeFocusOnClick On"
ToggleButton1.Value = True
ToggleButton1.Width = 90
End Sub
```

TextColumn Property Example

The following example uses the <u>TextColumn</u> property to identify the column of data in a <u>ListBox</u> that supplies data for its <u>Text</u> property. This example sets the third column of the ListBox as the text column. As you select an entry from the ListBox, the value from the **TextColumn** will be displayed in the <u>TextBox</u>.

This example also demonstrates how to load a multicolumn **ListBox** using the **<u>AddItem</u>** method and the <u>List</u> property.

To use this example, copy this sample code to the Script Editor of a form. To run the code you need to open the form so the **Open** event will activate. Make sure that the form contains:

- A ListBox named ListBox1.
- A **TextBox** named TextBox1.

```
Dim ListBox1
Dim TextBox1
Sub Item Open()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    Set TextBox1 = Item.GetInspector.ModifiedFormPages.Item("P.2").C
    ListBox1.ColumnCount = 3
    ListBox1.AddItem "Row 1, Col 1"
    ListBox1.List(0, 1) = "Row 1, Col 2"
    ListBox1.List(0, 2) = "Row 1, Col 3"
   ListBox1.AddItem "Row 2, Col 1"
   ListBox1.List(1, 1) = "Row 2, Col 2"
    ListBox1.List(1, 2) = "Row 2, Col 3"
    ListBox1.AddItem "Row 3, Col 1"
    ListBox1.List(2, 1) = "Row 3, Col 2"
    ListBox1.List(2, 2) = "Row 3, Col 3"
    ListBox1.TextColumn = 3
```

End Sub

```
Sub ListBox1_Click()
    TextBox1.Text = ListBox1.Text
End Sub
```

TopIndex Property Example

The following example identifies the top item displayed in a **ListBox** and the item that has the focus within the **ListBox**. This example uses the **TopIndex** property to identify the item displayed at the top of the **ListBox** and the **ListIndex** property to identify the item that has the focus. The user selects an item in the **ListBox**. The displayed values of **TopIndex** and **ListIndex** are updated when the user selects an item or when the user clicks the **CommandButton**.

- A Label named Label1.
- A **<u>TextBox</u>** named TextBox1.
- A Label named Label2.
- A **TextBox** named TextBox2.
- A **CommandButton** named CommandButton1.
- A ListBox named ListBox1 that is bound to the Subject field.

```
Sub CommandButton1_Click()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    ListBox1.TopIndex = ListBox1.ListIndex
    TextBox1.Text = ListBox1.TopIndex
    TextBox2.Text = ListBox1.ListIndex
End Sub
Sub Item_PropertyChange(byval pname)
    if pname = "Subject" then
        Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Co
        Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Co
        Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Co
        TextBox1.Text = ListBox1.TopIndex
        TextBox2.Text = ListBox1.ListIndex
    end if
End Sub
```

```
Sub Item Open()
    Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
   Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
    Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
   Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    Set Label1 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    Set Label2 = Item.GetInspector.ModifiedFormPages("P.2").Controls
    For i = 0 To 24
        ListBox1.AddItem "Choice " & (i + 1)
    Next
    ListBox1.Height = 66
    CommandButton1.Caption = "Move to top of list"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = False
    Label1.Caption = "Index of top item"
   TextBox1.Text = ListBox1.TopIndex
   Label2.Caption = "Index of current item"
    Label2.AutoSize = True
    Label2.WordWrap = False
    TextBox2.Text = ListBox1.ListIndex
End Sub
```



TripleState Property Example

The following example uses the **TripleState** property to allow **Null** as a legal value of a **CheckBox** and a **ToggleButton**. The user controls the value of **TripleState** through ToggleButton2. The user can set the value of a **CheckBox** or **ToggleButton** based on the value of **TripleState**. However, when a control is set to **Null**, no event is fired.

- A CheckBox named CheckBox1.
- A **ToggleButton** named ToggleButton1.
- A **ToggleButton** named ToggleButton2.

```
Sub Item Open()
    Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set ToggleButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    CheckBox1.Caption = "Value is True"
    CheckBox1.Value = True
    CheckBox1.TripleState = False
   ToggleButton1.Caption = "Value is True"
    ToggleButton1.Value = True
    ToggleButton1.TripleState = False
   ToggleButton2.Value = False
    ToggleButton2.Caption = "Triple State Off"
End Sub
Sub ToggleButton2_Click()
    Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set ToggleButton2 = Item.GetInspector.ModifiedFormPages("P.2").C
    If ToggleButton2.Value = True Then
        ToggleButton2.Caption = "Triple State On"
        CheckBox1.TripleState = True
        ToggleButton1.TripleState = True
```

```
Else
        ToggleButton2.Caption = "Triple State Off"
        CheckBox1.TripleState = False
        ToggleButton1.TripleState = False
    End If
End Sub
Sub CheckBox1_Click()
    Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    If IsNull(CheckBox1.Value) Then
        CheckBox1.Caption = "Value is Null"
    ElseIf CheckBox1.Value = False Then
        CheckBox1.Caption = "Value is False"
    ElseIf CheckBox1.Value = True Then
        CheckBox1.Caption = "Value is True"
    End If
End Sub
Sub ToggleButton1_Click()
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    If IsNull(ToggleButton1.Value) Then
        ToggleButton1.Caption = "Value is Null"
    ElseIf ToggleButton1.Value = False Then
        ToggleButton1.Caption = "Value is False"
    ElseIf ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Value is True"
    End If
End Sub
```

UndoAction, RedoAction Methods Example

The following example demonstrates how to undo or redo text editing within a text box or within the text area of a **ComboBox**. This sample checks whether an undo or redo operation can occur and then performs the appropriate action. The sample uses the **CanUndo** and **CanRedo** properties, and the **UndoAction** and **RedoAction** methods.

- A **<u>TextBox</u>** named TextBox1.
- A **ComboBox** named ComboBox1.
- Two <u>CommandButton</u> controls named CommandButton1 and CommandButton2.

```
Dim UserForm1
```

```
Sub CommandButton1 Click()
    If UserForm1.CanUndo = True Then
        UserForm1.UndoAction
        MsgBox "Undid IT"
    Else
        MsqBox "No undo performed."
    End If
End Sub
Sub CommandButton2_Click()
    If UserForm1.CanRedo = True Then
        UserForm1.RedoAction
        MsqBox "Redid IT"
    Else
        MsgBox "No redo performed."
    End If
End Sub
Sub Item_Open()
     Set UserForm1 = Item.GetInspector.ModifiedFormPages("P.2")
     Set TextBox1 = UserForm1.Controls("TextBox1")
```

```
Set ComboBox1 = UserForm1.Controls("ComboBox1")
Set CommandButton1 = UserForm1.Controls("CommandButton1")
Set CommandButton2 = UserForm1.Controls("CommandButton2")
TextBox1.Text = "Type your text here."
ComboBox1.ColumnCount = 3
ComboBox1.AddItem "Choice 1, column 1"
ComboBox1.List(0, 1) = "Choice 1, column 2"
ComboBox1.List(0, 2) = "Choice 1, column 3"
CommandButton1.Caption = "Undo"
CommandButton2.Caption = "Redo"
End Sub
```

Value Property Example

The following example demonstrates the values that the different types of controls can have by displaying the **Value** property of a selected control. The user chooses a control by pressing TAB or by clicking on the control. Depending on the type of control, the user can also specify a value for the control by typing in the text area of the control, by clicking one or more times on the control, or by selecting an item, page, or tab within the control. The user can display the value of the selected control by clicking the appropriately labeled <u>CommandButton</u>.

- A **CommandButton** named CommandButton1.
- A **<u>TextBox</u>** named TextBox1.
- A <u>CheckBox</u> named CheckBox1.
- A <u>ComboBox</u> named ComboBox1.
- A **CommandButton** named CommandButton2.
- A ListBox named ListBox1.
- A MultiPage named MultiPage1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A <u>ScrollBar</u> named ScrollBar1.
- A <u>SpinButton</u> named SpinButton1.
- A **TabStrip** named TabStrip1.
- A **TextBox** named TextBox2.
- A **ToggleButton** named ToggleButton1.

```
Sub CommandButton1_Click()
```

```
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set Form = Item.GetInspector.ModifiedFormPages("P.2")
TextBox1.Text = "Value of " & Form.ActiveControl.Name & " is "
End Sub
```

```
Sub Item_Open()
```

```
Set CommandButton1 = Item.GetInspector.ModifiedFormPages("P.2").
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set ComboBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
Set ListBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contro
```

```
Set CheckBox1 = Item.GetInspector.ModifiedFormPages("P.2").Contr
    Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").C
    Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").Contro
    CommandButton1.Caption = "Get value of current control"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = False
    CommandButton1.TabStop = False
   TextBox1.AutoSize = True
    For i = 0 To 10
        ComboBox1.AddItem "Choice " & (i + 1)
        ListBox1.AddItem "Selection " & (100 - i)
    Next
    CheckBox1.TripleState = True
    ToggleButton1.TripleState = True
   TextBox2.Text = "Enter text here."
End Sub
```

ZOrder Method Example

The following example sets the z-order of a **TextBox**, so the user can display the entire **TextBox** (by bringing it to the front of the z-order) or can place the **TextBox** behind other controls (by sending it to the back of the z-order).

- Three **TextBox** controls named TextBox1 through TextBox3.
- A **ToggleButton** named ToggleButton1.

```
Dim ToggleButton1
Dim TextBox1
Dim TextBox2
Dim TextBox3
Sub Item_Open()
Set TextBox1 = Item.GetInspector.ModifiedFormPages("P.2").TextBox1
Set TextBox2 = Item.GetInspector.ModifiedFormPages("P.2").TextBox2
Set TextBox3 = Item.GetInspector.ModifiedFormPages("P.2").TextBox3
Set ToggleButton1 = Item.GetInspector.ModifiedFormPages("P.2").Toggl
'Set up text boxes to show z-order in the form
TextBox1.Text = "TextBox 1"
TextBox2.Text = "TextBox 2"
TextBox3.Text = "TextBox 3"
TextBox1.Height = 40
TextBox2.Height = 40
TextBox3.Height = 40
TextBox1.Width = 60
TextBox2.Width = 60
TextBox3.Width = 60
TextBox1.Left = 10
TextBox1.Top = 10
TextBox2.Left = 25
                        'Overlap TextBox2 on TextBox1
TextBox2.Top = 25
```

TextBox3.Left = 40'Overlap TextBox3 on TextBox2, TextBox1 TextBox3.Top = 40ToggleButton1.Value = False ToggleButton1.Caption = "Bring TextBox2 to Front" ToggleButton1.Left = 10ToggleButton1.Top = 90ToggleButton1.Width = 50ToggleButton1.Height = 50End Sub Sub ToggleButton1_Click() If ToggleButton1.Value = True Then TextBox2.ZOrder (fmTop) 'Place TextBox2 on Top of z-order 'Update ToggleButton caption to identify next state ToggleButton1.Caption = "Send TextBox2 to back" Else TextBox2.ZOrder (1) 'Place TextBox2 on Bottom of z-'Update ToggleButton caption to identify next state ToggleButton1.Caption = "Bring TextBox2 to front" End If End Sub



Create an instant workgroup solution using public folders

By using custom views in a <u>public folder</u>, you can take an existing <u>item</u>, such as a contact, and turn it into a workgroup <u>form</u> from which any user of the public folder can get information or to which they can add data. With this method, you create an appointment, task, or contact in a public folder, create a custom view for the information in the folder, and then give permission to those who you want to use the folder.

Click a step below to begin:

Step 1: Create a public folder

- Step 2: Create an Outlook item in a public folder
- Step 3: Create and use a custom view in a public folder

Put a list of fields and values in the message body

To add a list of fields and values in the body of an item, define a variable to contain the string, and then use Outlook properties that refer to the field you want to include. For example, to include the To field in the message body, use the following. Chr (13) is the return character.

```
MessageString = "This letter is sent to " & Item.To & chr(13)
MessageString = MessageString & "second line goes here"
Item.Body = MessageString
```

Go to a line of code

1. Open the Script Editor.

How?

- 2. On the **Edit** menu, click **Go To**.
- 3. In the **Line Number** box, type in the number of the line of code to which you want to go.

Use message boxes to return values

One way to isolate errors is to use a message box to display the value of a variable or property at a particular point in the coce. This code example shows the selection length returned from the **SelLength** property in a message box.

MsgBox Item.GetInspector.ModifiedFormPages("Test").Textbox1.SelLengt



Step 1: Create a public folder

To create a <u>public folder</u>, you must have permission to create folders in an existing public folder. For information about how to obtain permission, see your administrator.

- 1. On the **File** menu, select **New**, and then click **Folder**. (CTRL+SHIFT+E)
- 2. In the **Name** box, enter a name for the folder.
- 3. In the **Folder contains** box, click the type of <u>item</u> that you want the folder to contain. A folder can only contain one type of item.
- 4. Click the **Select Folder** button, and then click the public folder in which you want your new public folder to appear.
- 5. If you do not want to add a <u>Shortcut</u> for the public folder to your <u>Navigation Pane</u>, click **No** in the **Add shortcut to Outlook Bar**? box.

Note You can copy a private folder to a public folder for quick creation of a public folder with existing items.

To go on to Step 2, c	lick ဳ
-----------------------	--------



Step 2: Create an Outlook item in a public folder

1. If the <u>public folder</u>, that you want to use does not exist, you can create it.

How?

- 2. In the Folder List or on the <u>Navigation Pane</u>, select the public folder in which you want to add the new <u>item</u>.
- 3. On the **File** menu, point to **New**, and then click the item that you want to create.

Tip To quickly let others add the public folder to their Public Folders folder, you can send a shortcut in a mail message.

To go on to Step 3, click 💌.



Step 3: Create and use a custom view in a public folder

With custom views, you can arrange information in a <u>public folder</u>, exactly how you want it. You can set which view is initially shown when a user opens the public folder, and you can remove all the standard views from a public folder and only show the custom views you create. By setting the permission for a public folder, you can determine who has access to the folder.