Imaging for Windows®
Automation Guide

Imaging for Windows®

Automation Guide

Disclaimer of Warranties and Limitation of Liabilities

Nothing contained herein modifies or alters in any way the standard terms and conditions of the purchase, lease, or license agreement by which the product was acquired, nor increases in any way the liability of the supplier of the software, its affiliates or suppliers (“the Supplier”). In no event shall the Supplier be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

Software Notice

All software must be licensed to customers in accordance with the terms and conditions of any approved and authorized license. No title or ownership of the software is transferred, and any use of the software beyond the terms of the aforesaid license, without written authorization of the publisher, is prohibited.

Restricted Rights Legend

The Licensed Product and accompanying documentation are Commercial Computer Software and documentation as defined under Federal Acquisition Regulations and agency supplements to them. Use, duplication or disclosure by the U.S. Government is subject to the restrictions of these licensing terms and conditions as prescribed in DFAR 227.7202-3(a) and DFAR 227.7202-4 or, as applicable, the Commercial Computer Software Restricted Rights clause at FAR 52.227-19. Manufacturer is Global 360, Inc., One Lincoln Centre, 5400 LBJ Freeway, Suite 300, Dallas, TX 75240, USA.

Contents

Imaging for Windows® Automation Guide

About This Guide

This guide describes the Automation component of the Global 360 Imaging for Windows® Developer Resources. Automation enables you to access Imaging objects programmatically using Visual Basic or another application that supports Automation.

In this Chapter

Purpose

The *Automation Guide* describes the Automation features of Global 360 Imaging for Windows®.

Prerequisites

To use this product, you should be familiar with the Microsoft® Windows® environment. If you are using a printer, scanner, or TWAIN-compliant device, you should also know how to connect and operate it.

If you plan to access documents residing on Global 360 Imaging Server (1.x) or an Execute360 server, you should be familiar with navigating document databases in those environments.

Related Information

For updated product information and general information about Imaging for Windows, visit our Web site at

www.global360.com

Support

Should you have questions regarding Imaging for Windows, or problems with your system after installation, consult your customer support representative.

For technical support, visit our Web site at

www.global360.com

1

Adding Imaging Using Automation

This chapter provides an overview of Imaging for Windows® and explains how to use Automation to image-enable your applications. It describes the object hierarchy of the Imaging application and explains how the Imaging application can function as an Automation server application or an Embedded server application. It also walks you through a sample project to help you get started.

In This Chapter

Overview

Imaging for Windows® features a rich Automation interface that provides programmatic access to the internal services of the Imaging application.

Automation is a powerful way to image-enable your application. It enables you to control the Imaging application programmatically from your application and to provide your users with the image display and manipulation functions that are contained within the Imaging application. You, in effect, make the Imaging application a fully functional, tested, and trusted *component* of your application.

Note: You cannot use Automation to control the Imaging Preview and Imaging Flow applications.

When the Imaging application is launched through Automation, it has a Single Document Interface (SDI). PDF files cannot be displayed.

Components are software modules that can be "plugged into" applications from other vendors. They provide end users with a specific set of additional functions and capabilities.

In addition to automating the Imaging application from your programs, you can also automate it from other Automation-capable programs, such as Microsoft® Word and Excel.

The Imaging application implements Automation as a full object model, similar to the Automation model of Microsoft Word and Microsoft Excel.

The object hierarchy starts with the Application object, continues with an ImageFile object and one or more Page objects, and then concludes with a Page Range object. Each object has its own set of properties and methods.

Note: Chapter 2 in this guide describes the properties and methods of each object.

The Automation demonstration project described later in this chapter shows you how to use Automation in Excel to:

- Invoke the Imaging application.
- Display an image.
- Select the view mode.
- Rotate the image.
- Obtain the number of pages in the image file.

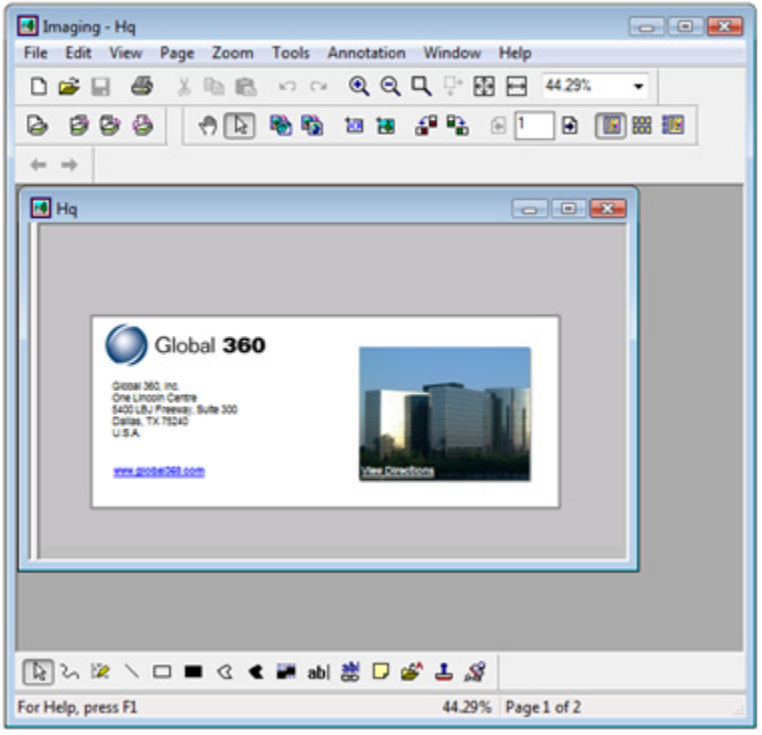
Imaging Components

Imaging for Windows lets your users access and control paper-based information directly on their computers. With it, users can view, manipulate, annotate, print, file, and share documents they used to manage as cumbersome paper files.

The following sections describe the components of Imaging for Windows.

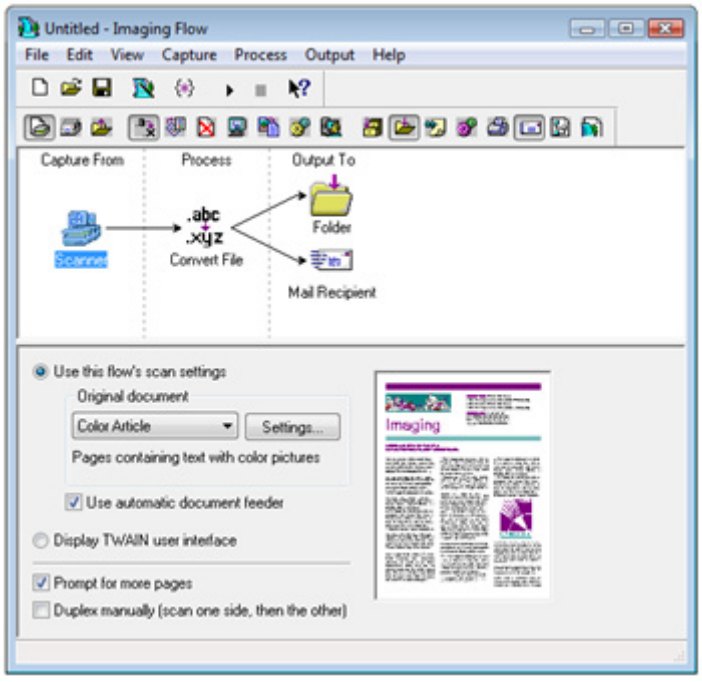
Imaging Application

The Imaging application is the main component of Imaging for Windows. It enables users to scan, view, annotate, manipulate, and store faxes, paper documents, and electronic images.



Imaging Flow

Imaging Flow enables users to automatically capture, process, and output image files. An intelligent and editable procedure — called a *flow* — defines and controls the work Imaging Flow performs.



Flow tools included within each flow perform specific functions. They can:

- Capture images from:
  - Scanners.
  - MAPI-compliant in-boxes.
  - Local and network folders.
- Process images by:
  - Converting them from one file type to another.
  - Applying compression.
  - Enhancing their appearance.
  - Permitting their review.
  - Converting them to text.
  - Deleting specified pages.
  - Entering information about an image document while the flow is processing.
  - Running a custom process.
- Output images by:
  - Posting them to Exchange folders.
  - Saving them to local or network folders.
  - Saving them to Execute360 Imaging or Imaging (1.x) servers.
  - Printing them.
  - Sending them to others via e-mail.
  - Running a custom process.

Imaging Preview

Imaging Preview is a light version of the Imaging application. It lets users view image files quickly and, if necessary, load them into the Imaging application for editing.

Invoking Imaging for Windows

Imaging for Windows includes several development tools and methods that let you add Imaging functions to your applications. The development tools and methods include:

- Command line invocation
- OLE
- Automation

Command Line Invocation

You can invoke the Imaging application using its command line. Command line invocation is the most simple but least powerful way to implement Imaging functions in your application.

Because the command line can accept a fully qualified image file name, you can use standard **Shell** functions within your application to invoke the Imaging application with an image on display.

Within your call to the **Shell** function, include the path and file name of the Imaging application along with the path and file name of the image file you want it to display.

For example, if you are developing under Imaging, you can use the following statement to invoke the Imaging application and display an image file:

```
Shell("c:\Program Files\Common Files\
Global 360\Imaging\Imaging.exe c:\Quote.tif", 1)
```

Employing the command line interface does not make the Imaging application a full-fledged component of your application. The command line interface does not give you the opportunity to manipulate the application or the image after it is displayed.

OLE

You can use standard OLE functions to embed and link image files in your application and other applications, such as Microsoft Word, Excel, Access, and SQL Server. OLE lets you add a subset of Imaging functions to your application. It is useful when you want to add Imaging functions with an absolute minimum of coding.

Using a container control such as that provided by Visual Basic, you can add image files as insertable objects within your application at design time. Image files can be embedded or linked. For example, you can use the Visual Basic OLE Container control to easily embed or link image files in your application.

As an alternative, you can use the container control to create a placeholder in your application for image files that will be added at run time. Set the appropriate properties or provide end users with drag-and-drop capability so they can select image files for display at run time.

OLE does not make the Imaging application a full-fledged component of your application. OLE does not give you the opportunity to manipulate the application or the image after it is displayed.

Users can edit embedded images within your application and linked images within the Imaging application.

Your application is the container, while the Imaging application is the server. Users can edit and open embedded or linked image files, as described in the following sections.

Embedded Image Files

When you embed an image file in your application, the application stores the image data within it.

When end users **edit** an embedded image file, it becomes "in-place activated," causing your application to display a subset of the Imaging application menus. The menus provide access to Imaging functions that let users edit the activated image file in-place, that is, within your application.

When end users **open** an embedded image file, the Imaging application appears with the embedded image displayed within it. Changes users make to the image in the Imaging application also appear on the linked image in your application. If desired, users can save a copy of the image to another file by clicking **SaveAs** on the **File** menu.

Linked Image Files

When you link an image in your application, the image data remains external to your application. Your application stores only a reference to the image file.

When end users **edit** or **open** a linked image file, the Imaging application appears with the image file displayed. This enables them to perform the full range of Imaging functions on the displayed image file.

In-place activation is not available because the linked image file may also be available to other containers (referential integrity).

As in the case of embedded image files, changes users make to the image in the Imaging application also appear on the linked image in your application.

When to Use Automation

Automation lets you add Imaging functions to your application by making the Imaging application a full-fledged component of your application.

Automation is useful when you want images to be displayed in a window that is separate from your application and when you want to control the Imaging application from your application.

Your application can control the state of the Imaging application as well as manipulate the displayed image but your application cannot respond to events that occur when users perform Imaging operations.

Depending on the degree of control you want to exert, automating the Imaging application from your application can be accomplished with a minimal or substantial amount of coding.

Example

Imaging Flow, a component of Imaging for Windows, demonstrates a good example of Automation.

The Review flow tool invokes the Imaging application to permit users to review image files as they are being processed by the current flow.

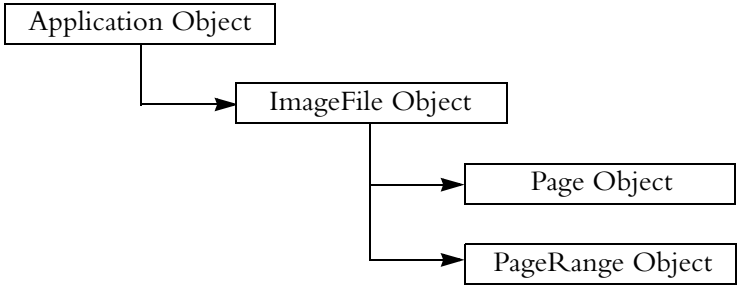
At flow design time, the author can set Review tool options that manipulate the Imaging application as well as the image it displays. These options include:

- Whether to view image pages, thumbnails, or both.
- The size and position of the Imaging application window.
- The zoom setting to apply to images.
- Whether to open image files as read only.
- Whether to scale black-and-white images to gray.

The Object Hierarchy

The object model of the Imaging application includes:

- One top-level object, called the Application object;
- One document object, called the ImageFile object; and
- Two objects that support the ImageFile object, called the Page object and the PageRange object.



The first time you start the Imaging application, it adds the Application object to the Windows® registry. Imaging Automation exposes only the Application object for creation. Other programmable objects can be created by referencing the Application object.

Each object in the hierarchy has its own set of properties and methods. Refer to Chapter 2 for a description of the properties and methods of each object.

Application Object

Use the Application object to create an instance of the Imaging application and to control it. The Application object controls every other object you create as well as the environment of the application, such as the application's size and position.

ImageFile Object

The ImageFile object represents an image document file. Use it to specify the name of an image file and to provide basic filing functions such as open, save, close, print, insert, update, and append. Use it also to provide image manipulation functions such as rotate, create contact sheet, and perform OCR.

Page Object

Each Page object represents an image document page. Use it to manipulate the individual pages of an image file and to provide functions such as delete, flip, print, rotate, scroll, and perform OCR.

PageRange Object

The PageRange object represents a range of consecutive pages within an ImageFile object — starting at the **StartPage** property and ending at the **EndPage** property. Use it to manipulate a range of pages and to provide page manipulation functions such as delete, print, and perform OCR.

Note: Automation is not aware of the actions performed by users within the Imaging application. The objects known to Automation remain in the state they were in when last affected programmatically.

In other words, if users change a displayed object, Automation does not update that object within its Application object. For example, if users change the active page, Automation does not update the **ActivePage** property.

However, properties and methods are available that let you determine whether a change has occurred. At your option, you can use them to update the corresponding objects known to Automation.

Imaging Application Modes

You can use the **AppState** property of the Application object to determine whether the Imaging application is running as an Automation server or an Embedded server.

The Imaging application can function as an Automation server application or as an Embedded server application.

The following sections describe each mode and include examples.

Automation Server Mode

In every version of Imaging for Windows, the Imaging application can function as a stand-alone Automation server application.

When automated in this mode, the Imaging application is directed to display and manipulate an image file that is external to your application; such as a file resident on a local or network drive. Your program uses the properties and methods of the Imaging Automation objects to control the Imaging application and to display and manipulate the image.

The demonstration project, described later in this chapter, is an excellent example of using the Imaging application as an Automation server application.

Embedded Server Mode

Imaging for Windows has several Imaging Automation properties and methods to manipulate an embedded image document object.

When automated in this mode, the Imaging application is directed to manipulate an image document object that has been embedded into your program using, for example, the OLE Container control of Visual Basic.

Depending on how you code your application, you can manipulate the embedded image document in-place or within the Imaging application window (refer to the next section for examples).

Note: The Automation interface allows the in-place activation of embedded objects only. It does not permit the in-place activation of linked objects.

Examples

This section contains examples that show how to automate the Imaging application as a stand-alone Automation server application and as an Embedded server application.

Note: The example that demonstrates automating the Imaging application as an Automation server application is more extensive because:

- The principles behind automating the Imaging application are similar no matter which mode is used.
- Use of the Imaging application as an Automation server application is more prevalent.

As an Automation Server Application

This example shows how to use Visual Basic to automate the Imaging application as an Automation server application. (Refer to the code snippet at the end of this section.)

Automating the Imaging application involves a series of programming steps that begin with the creation of Application and Image File objects and continue with the application control and image manipulation functions you want to perform.

After you create an object, you can access the properties and methods of the object using the object variable.

To create the Application and Image File Objects

1 Declare the object variables that will contain references to the Application and Image File objects.

2 Use the **Set** statement and the **CreateObject** function of Visual Basic to create and return a reference to the Application object.

3 Use the **Set** statement of Visual Basic and the **CreateImageViewerObject** method of the Application object to create and return a reference to the ImageFile object.

With the Application and ImageFile objects instantiated, you can now manipulate the Imaging application as well as any image the application displays.

To manipulate the Imaging Application

1 Set the **TopWindow** property of the Application object to True to have the Imaging application window remain on top of all other applications that may be running.

2 Invoke the **Open** method of the ImageFile object to open and display an image file. In your call to the **Open** method, pass the following parameters:

ImageFile — The path and file name of the image file to display

IncludeAnnotation (optional) — **True** or **False**: whether to display annotations that may be present in the image file

Page (optional) — The number of the image page to display

DisplayUIFlag (optional) — **True** or **False**: whether to display the **Open** dialog box, which lets end users select the file they want to display

Now that an image is open and on display, you can manipulate it. The following paragraphs provide some examples.

- Invoke the **RotateLeft** method of the Page object to rotate page 1 of the image file 90 degrees to the left. Keep in mind that there is one Page object for each image page in the file.
- Use the **Height** property of the Page object to assign the height of page 1 to the local variable `lngPageHeight`.
- Invoke the **Print** method of the PageRange object to print pages 1 and 2 on the default printer.
- Set the **ActivePage** property of the ImageFile object to 2 to display page 2 of the image file.

A PageRange object represents a range of consecutive pages within an ImageFile object.

To close the Image File and exit the Application

1. Invoke the **Close** method of the ImageFile object to close the image file.
2. Invoke the **Quit** method of the Application object to exit the application.
3. Set the object variables to **Nothing** to free system resources.

```
'Declare variables
   Dim objApp As Object
   Dim objImg As Object
   Dim vntPrtRange As Variant
   Dim lngPageHeight As Long

'Create the Application object (Standard VB call)
   Set objApp = CreateObject("Imaging.Application")

'Create the ImageFile object
   Set objImg = objApp.CreateImageViewerObject(1)

'Set the application's TopWindow property to TRUE (stay on top)
   objApp.TopWindow = True

'Call the ImageFile object Open Method to display page 1 of myimage.tif
   objImg.Open "c:\images\myimage.tif", True, 1, False

'Create and rotate one Page object
   objImg.Pages(1).RotateLeft

'Return the height of the image from the Page object
   lngPageHeight = objImg.Pages(1).Height

'Create a PageRange object and print pages 1 and 2
   vntPrtRange = objImg.Pages(1,2).Print

'Display page 2 of the image
   objImg.ActivePage = 2

'Close ImageFile object and quit the application
   objImg.Close
   objApp.Quit

'Release system resources
   Set objApp = Nothing
   Set objImg = Nothing
```

Methods Not Available in Automation Server Mode

You cannot use the following methods when the Imaging application is functioning as an Automation server application:

- **SaveCopyAs** method of the ImageFile object
- **Update** method of the ImageFile object

As an Embedded Server Application

The following sections demonstrate how to automate the Imaging application as an Embedded server application. The examples assume you are embedding an image document object into a Visual Basic application using the OLE Container control.

Example 1

In this example, the Imaging application displays the embedded image document in a separate window for editing.

```
Set objApp = CreateObject("Imaging.Application")
Set objImg = objApp.CreateImageViewerObject(1)
oleImg.CreateEmbed("", "Imaging.Document")
oleImg.DoVerb vbOLEOpen
objImg.InsertExistingPages "Test.tif", 1, 1, 1, False
```

Example 2

In this example, the Imaging application is in-place active and displays a subset of its menus within your application. The menus provide access to functions that let users edit the image document object "in-place" — that is, within your application.

```
Set objApp = CreateObject("Imaging.Application")
oleImg.CreateEmbed("", "Imaging.Document")
oleImg.DoVerb vbOLEShow
Set objImg = objApp.CreateImageViewerObject(1)
objImg.InsertExistingPages "Test.tif", 1, 1, 1, False
```

Example 3

In this example, the Imaging application displays the embedded image document in an instance of the Imaging application that is already running.

```
oleImg.CreateEmbed("", "Imaging.Document")
oleImg.DoVerb vbOLEOpen
Set objApp = CreateObject("Imaging.Application")
Set objImg = objApp.CreateImageViewerObject(1)
```

Properties and Methods Not Available in Embedded Server Mode

You cannot use the following properties and methods when the Imaging application is functioning as an Embedded server application:

- **Edit** property of the Application object
- **Height** and **Width** properties of the Application object
- **ImageView** property of the Application object (if the application is in-place active)
- **Left** property of the Application object
- **Top** property of the Application object
- **Close** method of the ImageFile object
- **FindOIServerDoc** method of the ImageFile object
- **New** method of the ImageFile object
- **Open** method of the ImageFile object
- **Quit** method of the Application object (if the application is in-place active)
- **SaveAs** method of the ImageFile object

Demonstration Project

This section demonstrates how to automate the Imaging application from Microsoft Excel.

While a wide-ranging discussion of every Imaging function is beyond the scope of this chapter, the information presented here is sufficient to get started.

The demonstration project was developed using Microsoft Visual Basic for Applications and Excel.

Even if you are not going to automate the Imaging application, you'll find the section in this chapter on View Modes useful.

To help you use Automation to image-enable your applications, a demonstration project — called Automation From Excel — shows you how to:

- Invoke the Imaging application and open an image.
- Obtain the page count.
- Rotate an image page.
- Set the desired view mode.
- Close the image and the application.

Note: Chapter 2 of this guide describes the properties and methods of each Imaging Automation object.

Before walking through the demonstration project, read the following section, which describes the view modes of the Imaging application.

View Modes

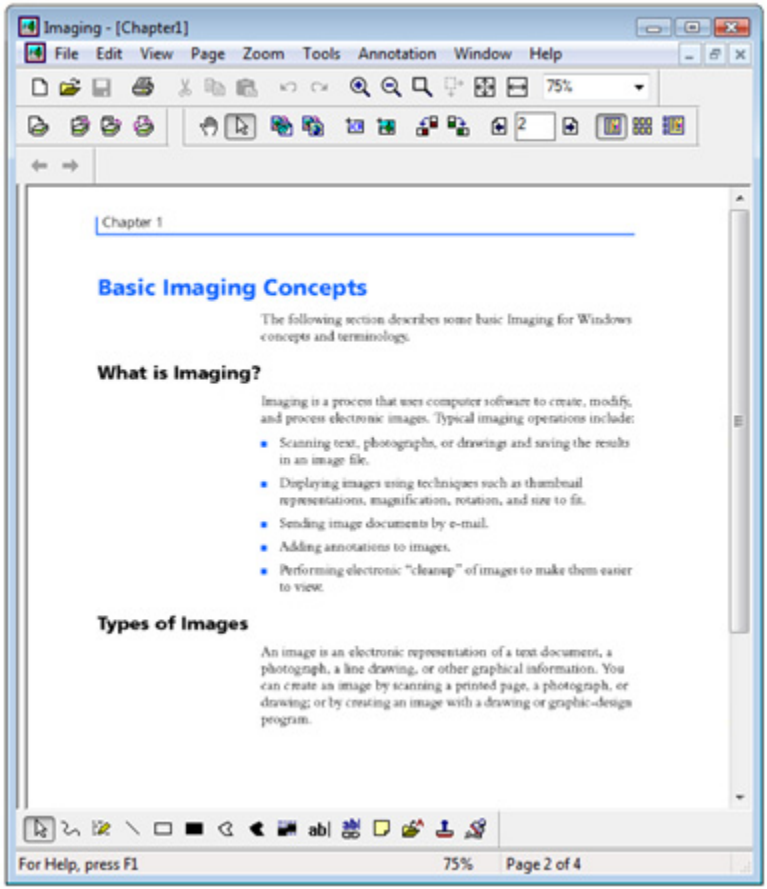
The Imaging application has three view modes that enable users to view and work with image files. Each view mode has its own set of advantages and capabilities.

The **ImageView** property of the Application object enables you to invoke — most likely in response to user input — any one of the three view modes. You should consider making view mode selection available to your users when automating the Imaging application.

The following sections describe the view modes.

One Page

The One Page view mode lets users display image files one page at a time. It lets users display image pages in the entire window while maintaining complete access to the menus, toolbars, and functions of the application.

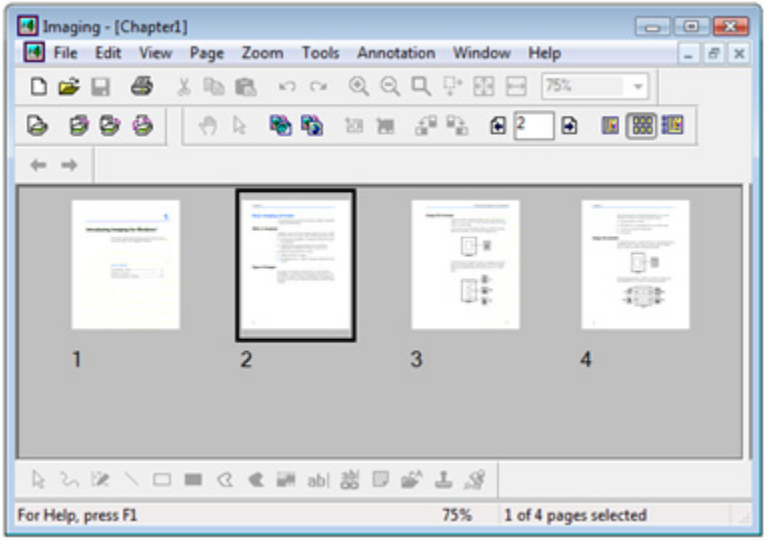


Thumbnail

The Thumbnail view mode lets users display image files as a series of thumbnail images — one for each image page. It lets users:

- View multiple image pages simultaneously.
- Rearrange pages using drag and drop.
- Delete pages.
- Drag and drop pages to and from other applications that support drag and drop functionality.

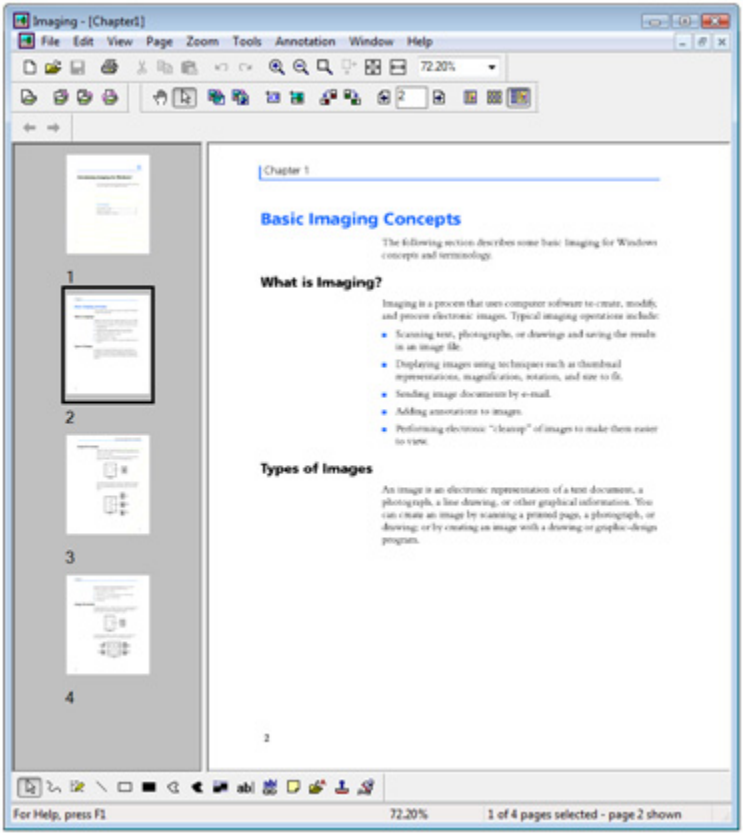
Keep in mind that some Imaging functions — such as annotation and zoom — are not available in this mode because they are not appropriate for use on such small images.



Page and Thumbnails

The Page and Thumbnails view mode is a combination of the first two view modes. It enables users to display image files one page at a time *and* as series of thumbnail images — one for each image page in the file.

This view mode lets users perform Imaging tasks that are available to both the One Page view mode and the Thumbnail view mode.



Example

Users of Excel may want to display and manipulate an image file referenced within a spreadsheet.

Scenario

In her role as a product manager for a major computer company, Eileen regularly uses Microsoft Excel to create product configurations of computers sold on contract to government agencies.

After she completes a configuration spreadsheet, she typically submits it to review via e-mail. In the past, several reviewers have requested that she also include a scanned copy of the contract.

At a recent employee meeting, Eileen asked whether her reviewers could display a scanned contract from Excel. Knowing that Imaging for Windows is on every desktop in the company, you told her that you could automate the Imaging application from Excel to give her reviewers quick access to a scanned contract, or any other image file for that matter.

All Eileen needs to do is:

1. Scan the contract using Imaging for Windows.
2. Import your code module into her Excel spreadsheet.
3. Enter the path and file name of the scanned contract in Cell A1 of the spreadsheet.
4. Send both the image file and the spreadsheet file to her reviewers.

The Automation From Excel Project

The file names for the Automation From Excel project are `ImagingAutomation.xls`, and `Facc.tif`.

The Automation From Excel project demonstrates:

- Invoking the Imaging application and opening an image from Excel.
- Obtaining the page count.
- Rotating an image page.
- Setting the desired view mode.
- Closing the image and the application.

The project consists of the following files:

ImagingAutomation.xls — A sample spreadsheet that contains the AutoFromExcel.bas code module.

Facc.tif — A sample TIFF image file that simulates the title page of a government contract.

The AutoFromExcel.bas code module contains the following macros:

f_InitializeApp() — Initializes the Imaging application.

s_DispImg() — Displays the image file.

s_GetPagecount() — Obtains the number of pages in the image file and displays it in a worksheet cell.

s_RotateImg() — Rotates the image 90 degrees to the left.

s_ViewSingle() — Places the Imaging application in the One Page view mode.

s_ViewThumbnails() — Places the Imaging application in the Thumbnail view mode.

s_ViewThumbAndSingle() — Places the Imaging application in the Page and Thumbnails view mode.

s_CloseImg() — Closes the image file and exits the Imaging application.

The AutoFromExcel.bas code module uses the following Automation methods to provide the Imaging functions:

Open method (ImageFile object) — Opens the image file in the Imaging application.

CreateImageViewerObject method (Application object) — Creates and returns an ImageFile object.

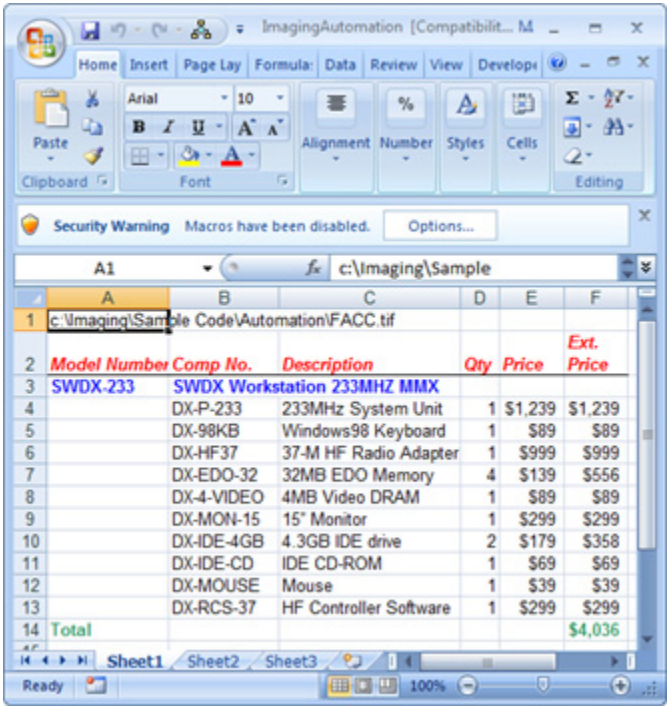
RotateLeft method (Page object) — Rotates the image 90 degrees counterclockwise.

Close method (ImageFile object) — Closes the ImageFile object.

Quit method (Application object) — Exits the application.

Opening the Spreadsheet File

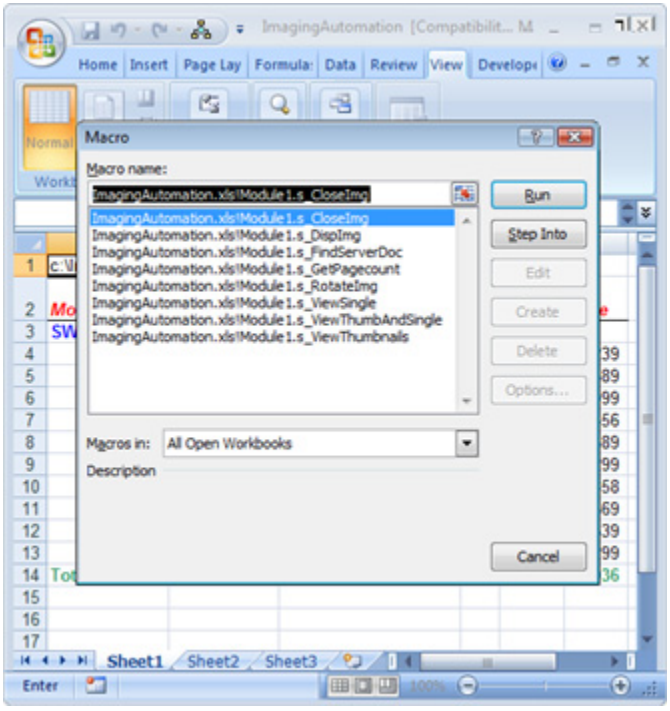
Start Excel and then open the `ImagingAutomation.xls` file. The sample spreadsheet appears.



Opening and Displaying the Image File

Give focus to Cell A1, which contains the path and file name of the sample TIFF image file.

On the **Tools** menu, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.



Click the **s_DispImg** macro and then click **Run**.

When the macro runs, code in the `General Declarations` area of the code module defines the object variables that contain references to the Application and Image File objects.

```
Dim objApp As Object
Dim objImg As Object
```

Then, the **s_DispImg()** subroutine executes its code.

The **s_DispImg()** subroutine obtains the path and file name of the image file to open from the active cell of the spreadsheet. Then it assigns the path and file name to the `strCurrentFile` local variable.

```
Sub s_DispImg()

    Dim strCurrentFile As String
    Dim strCurrentImageName As String

    'Get file name to display from spread sheet
    strCurrentFile = ActiveCell.Value
    .
    .
    .
    'If the Application object not created, create it.
    If objApp Is Nothing Then
        If f_InitializeApp() = False Then  'Continue if successful
            Exit Sub
        End If
    End If

    'Make the Imaging application on-top.
    objApp.TopWindow = True

    On Error Resume Next        'If no file is open.
    'Get the name of the open Image file.
    strCurrentImageName = objImg.Name

    On Error GoTo 0             'Reset error handler
    If strCurrentImageName <> "" Then
        'Always close existing image file before opening a new one.
        objImg.Close
    End If

    On Error GoTo OpenImageMethodError
    'Open the Image file in the ActiveCell
    objImg.Open strCurrentFile
    Exit Sub

OpenImageMethodError:
    sMsg = "Error => " & Str$(Err.Number) & " " & Err.Description
    MsgBox (sMsg)
    'Close the Imaging application
    s_CloseImg

End Sub
```

Next, the subroutine checks to see whether an instance of the Imaging application exists. If it does not, it invokes the **f_InitializeApp()** function.

The **f_InitializeApp()** function uses the **Set** statement and the **CreateObject** function of Visual Basic to create and return a reference to the Application object. Then it uses the **Set** statement of Visual Basic and the **CreateImageViewerObject** method of the Application object to create and return a reference to the ImageFile object.

```
'--------------------------------------------------------------------
'                     Initialize the Imaging application.
'       This function will be called when user attempts to open an
'       image file for the first time and the Imaging application
'       is not loaded.
'       If the Imaging application is found and the Application object
'       and the Image object are set, the function returns TRUE;
'       otherwise, the function returns FALSE.
'--------------------------------------------------------------------
Function f_InitializeApp() As Boolean

    Set objApp = Nothing
    Set objImg = Nothing
    'Create an Application Object
    Set objApp = CreateObject("Imaging.Application")
    'Create an ImageFile Object
    Set objImg = objApp.CreateImageViewerObject(1)
    f_InitializeApp = True

End Function
```

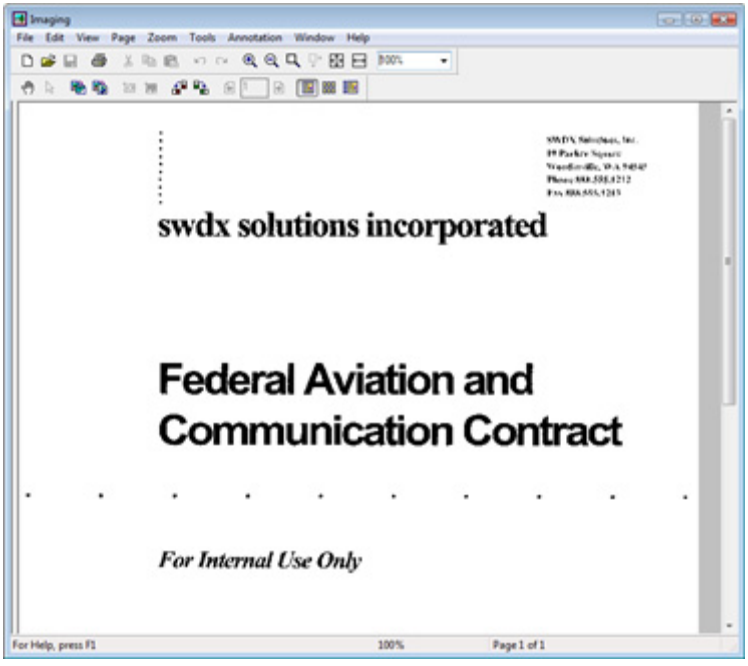
With the Application and ImageFile objects now fully instantiated, control returns to the **s_DispImg()** subroutine.

The **_DispImg()** subroutine sets the **TopWindow** property of the Application object to **True** to have the Imaging application window remain on top of all other applications that may be running.

Then it checks to see whether an image file is already displayed by examining the value of the **Name** property of the ImageFile object. If the **Name** property is not blank, the subroutine invokes

the **Close** method of the ImageFile object to close the displayed image file.

Next, the subroutine invokes the **Open** method of the ImageFile object, passing to it the path and file name of the image to display (from `strCurrentFile`). The **Open** method opens the image file in the Imaging application window.



Now that the image is open and on display, you can use some of the other macros to manipulate it and the Imaging application.

Obtaining the Page Count

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_GetPagecount** macro and then click **Run**. The **s_GetPagecount()** subroutine executes its code.

The subroutine obtains the page count from the **PageCount** property of the ImageFile object and assigns it to the `lngPageCount` local variable. Then it invokes the **Cells** function of Excel to display the page count (from `lngPageCount`)in the cell adjacent to the active cell on the spreadsheet.

```
Sub s_GetPagecount()

    Dim lngPageCount As Long

    If objImg Is Nothing Then
        MsgBox ("Please Open an Image file first")
        Exit Sub
    End If

    'Get the page count.
    lngPageCount = objImg.PageCount

    'Put the page count in the adjacent column.
    Cells(ActiveCell.Row, ActiveCell.Column + 1) = lngPageCount

End Sub
```

Rotating an Image Page

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_RotateImg** macro and then click **Run**. The **s_RotateImg()** subroutine executes its code.

The subroutine obtains the page number of the currently displayed image page from the **ActivePage** property of the ImageFile object, and assigns it to the `lngActivepage` local variable. Then it invokes the **RotateLeft** method of the Page object to rotate the displayed image page 90 degrees to the left.

```
Sub s_RotateImg()

    Dim lngActivepage As Long

    If objImg Is Nothing Then
        MsgBox ("Please open an image file first")
        Exit Sub
    End If

    lngActivepage = objImg.ActivePage
    objImg.Pages(lngActivePage).RotateLeft

End Sub
```

Setting the One Page View Mode

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_ViewSingle** macro and then click **Run**. The **s_ViewSingle()** subroutine executes its code.

The subroutine invokes the **ImageView** method of the Application object with a parameter value of 0, which places the Imaging application in the One Page view mode.

```
Sub s_ViewSingle()

  If objImg Is Nothing Then
        MsgBox ("Please Open an Image file first")
        Exit Sub
    End If

    'Place the Imaging application in One Page view mode.
    objApp.ImageView = 0

End Sub
```

Setting the Thumbnail View Mode

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_ViewThumbnails** macro and then click **Run**. The **s_ViewThumbnails()** subroutine executes its code.

The subroutine invokes the **ImageView** method of the Application object with a parameter value of 1, which places the Imaging application in the Thumbnail view mode.

```
Sub s_ViewThumbnails()

    If objImg Is Nothing Then
        MsgBox ("Please Open an Image file first")
        Exit Sub
    End If

    'Place the Imaging application in Thumbnail view mode.
    objApp.ImageView = 1

End Sub
```

Setting the Page and Thumbnails View Mode

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_ViewThumbAndSingle** macro and then click **Run**. The **s_ViewThumbAndSingle()** subroutine executes its code.

The subroutine invokes the **ImageView** method of the Application object with a parameter value of 2, which places the Imaging application in the Page and Thumbnails view mode.

```
Sub s_ViewThumbAndSingle()

    If objImg Is Nothing Then
        MsgBox ("Please Open an Image file first")
        Exit Sub
    End If

    'Place the Imaging application in Page and Thumbnails view mode.
    objApp.ImageView = 2

End Sub
```

Closing the Image File and the Imaging Application

On the **Tools** menu in Excel, point to **Macro** and then click **Macros**. The **Macro** dialog box appears.

Click the **s_CloseImg** macro and then click **Run**. The **s_CloseImg()** subroutine executes its code.

The subroutine invokes the **Close** method of the ImageFile object to close the currently displayed image file. Then it invokes the **Quit** method of the Application object to close the Imaging application. Finally, it sets the object variables to **Nothing** to free system resources.

```
Sub s_CloseImg()

    On Error Resume Next
    objImg.Close            'Close open image
    objApp.Quit             'Quit Automation application
    Set objImg = Nothing    'Destroy Image object
    Set objApp = Nothing    'Destroy Application object
    On Error GoTo 0         'Reset Error handler

End Sub
```

2

Automation Lexicon

This chapter describes the properties and methods of each Imaging for Windows® Automation object.

In this Chapter

Overview

Automation enables you to control the Imaging application programmatically from *within* your application. Using it, you can provide your end users with all of the capabilities of the Imaging application.

Each object has its own set of properties and methods. The remainder of this chapter describes each one.

Note: Refer to Chapter 1 of this guide for more information about using Automation to image-enable your applications.

Application Object

The Application object is a top-level object that controls every other object you create. The Application object also allows you to set the environment. For example, you can control the size and position of the Imaging application window and the visibility of scroll bars, the status bar, and the toolbar.

Application Object Properties

The following table lists the Application object properties. The properties that affect the displayed image (for example, **DisplayScaleAlgorithm**, **ImagePalette**, and **Zoom**) affect every image displayed in the Application object.

Application Object Properties

Property	Description
ActiveDocument	Returns the active ImageFile object.
AnnotationPaletteVisible	Sets or returns the visibility of the application's annotation palette.
Application	Returns the Application object.
AppState	Returns the state of the image viewer application.
DisplayScaleAlgorithm	Sets or returns the scaling algorithm used for displaying images.

Application Object Properties (cont.)

Property	Description
Edit	Sets or returns the application's ability to edit the displayed object.
FullName	Returns the file specification for the Application object.
Height	Sets or returns the distance between the top and bottom edge of the application window.
ImagePalette	Sets or returns the image palette used for image display.
ImageView	Sets or returns the present image view.
ImagingToolBarVisible	Sets or returns the visibility of the application's scan toolbar. Not available in all releases.
Left	Sets or returns the distance between the left edge of the physical screen and the main application window.
Name	Returns the name of the Application object.
Parent	Returns the Application object.
Path	Returns the path specification for this application's executable file.
ScannerIsAvailable	Sets or returns the state of the scanner.
ScanToolBarVisible	Sets or returns the visibility of the application's imaging toolbar.
ScrollBarsVisible	Sets or returns the visibility of the application's scroll bars.
StatusBarVisible	Sets or returns the visibility of the application's status bar.
ToolBarVisible	Sets or returns the visibility of the application's toolbar.
Top	Sets or returns the distance between the top edge of the physical screen and application's window.
TopWindow	Sets or returns the application's top window flag.
Visible	Returns the visibility of the application.
WebToolBarVisible	Sets or returns the visibility of the web toolbar.
Width	Sets or returns the distance between the left and right edges of the application's window.
Zoom	Sets or returns the zoom factor for image display.

ActiveDocument Property

Description Returns the active ImageFile object in the Application object. This is a read-only property.

Usage *ApplicationObject*.**ActiveDocument**

Data Type Object.

Example
```
'This example returns the ImageFile object in the application.
Dim Img as Object
Set Img = App.ActiveDocument
```

AnnotationPaletteVisible Property

Description Sets or returns the visibility of the annotation palette. This is a read/write property.

Usage *ApplicationObject*.**AnnotationPaletteVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **AnnotationPaletteVisible** property settings are:

Setting	Description
True	(Default) The annotation palette is visible.
False	The annotation palette is not visible.

Application Property

Description Returns the Application object. This is a read-only property.

Usage *ApplicationObject*.**Application**

Data Type Object.

Example
```
'This example returns the Application object.
Dim Parent As ObjectSet Parent = App.Application
```

AppState Property

Description Returns the state of the Application object. The state indicates whether the application is running as an embedded or automation server. This is a read-only property.

Usage *ApplicationObject*.**AppState**

Data Type Short.

Remarks The **AppState** property settings are:

Setting	Description
1	The application is running as an embedded server.
2	The application is running as an automation server.

DisplayScaleAlgorithm Property

Description Sets or returns the scaling algorithm used for displaying images. This is a read/write property.

Usage *ApplicationObject*.**DisplayScaleAlgorithm** [=*value*]

Data Type Short.

Remarks The DisplayScaleAlgorithm value can be specified before or after an image is displayed. The property settings are:

Setting	Description
0	(Default) Normal decimation.
1	Gray4 — 4-bit gray scale (16 shades of gray).
2	Gray8 — 8-bit gray scale (256 shades of gray).
3	Stamp — Represents the image as a thumbnail.
4	Optimize — Changes the display scale algorithm based on the image type of the displayed image. Black and white images are scaled to gray. Palettized 4- and 8-bit, RGB, and BGR images remain color.

Note: This property must be set prior to opening the ImageFile object. For this property to take effect after an image is open, you must reopen the image.

Edit Property

Description Sets or returns the Application object's ability to edit the displayed object. You should set the Edit property prior to opening each ImageFile object. This is a read/write property.

Usage *ApplicationObject*.**Edit** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **Edit** property settings are:

Setting	Description
True	(Default) Image editing is available.
False	The displayed object cannot be changed.

Note: You must set the **Edit** property prior to opening the ImageFile object. You can only set the Edit property once in the current session.

FullName Property

Description Returns the file specification for the Application object, including the path. This is a read-only property.

Usage *ApplicationObject*.**FullName**

Data Type String.

Height Property

Description Sets or returns the distance, in pixels, between the top and bottom edge of the Application object's window. This is a read/write property.

Usage *ApplicationObject*.**Height** [=*value*]

Data Type Long.

Remarks This property must be set prior to opening the ImageFile object. It only takes effect if the **Width**, **Top**, and **Left** properties are also set. If you set the **Height** property to less than the minimum allowable window size, the value is ignored. The minimum setting is usually 27.

The **Height** property only returns the value that you set programmatically prior to opening the window. It does not return changes made to the window after it has been opened.

ImagePalette Property

Description Sets or returns the image palette used to display an image. This is a read/write property.

> **Note:** The **ImagePalette** property must be set prior to opening the ImageFile object. For this property to take effect after an image is open, you must reopen the image.

Usage *ApplicationObject*.**ImagePalette** [=*value*]

Data Type Short.

Remarks The **ImagePalette** property settings are:

Setting	Description
0	(Default) Custom
1	Common
2	Gray8 — 8-bit grayscale (256 shades of gray)
3	RGB24 — 24-bit (millions of colors)
4	Black and white

ImageView Property

Description Sets or returns the present image view. This is a read/write property.

Usage *ApplicationObject*.**ImageView** [=*value*]

Data Type Short.

Remarks The **ImageView** property settings are:

Setting	Description
0	(Default) One page view
1	Thumbnails view
2	Page and Thumbnails view

The **ImageView** property and the **ImageFileObject.ActivePage** property have the following relationships:

View	Relationship
One Page	(Default) The active page is displayed.
Thumbnails	The active page appears in thumbnail view.
Page and Thumbnails	The active page is the page that is displayed.

See Also ImageFileObject.**ActivePage** property.

ImagingToolBarVisible Property

Description Sets or returns the visibility of this Application object's imaging toolbar. This is a read/write property.

Usage *ApplicationObject*.**ImagingToolBarVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **ImagingToolBarVisible** property settings are:

Setting	Description
True	(Default) The imaging toolbar is visible.
False	The imaging toolbar is not visible.

Left Property

Description Sets or returns the distance, in pixels, between the left edge of the physical screen and the Application object's window. This is a read/write property.

Usage *ApplicationObject*.**Left** [=*value*]

Data Type Long.

Remarks The **Left** property must be set prior to opening the ImageFile object. This property only takes effect if the **Height**, **Width**, and **Top** properties are also set.

The **Left** property only returns the value that you set programmatically prior to opening the window. It does not return changes made to the window after it has been opened.

Name Property

Description Returns the name of this Application object. This is a read-only property.

Usage *ApplicationObject*.**Name**

Data Type String.

Parent Property

Description Returns the parent of the Application object. This is a read-only property.

Usage *ApplicationObject*.**Parent**

Data Type Object.

Path Property

Description Returns the path specification for the Application object's executable file. This is a read-only property.

Usage *ApplicationObject*.**Path**

Data Type String.

ScannerIsAvailable Property

Description Sets or returns the availablity of the scanner. This is a read/write property.

Usage *ApplicationObject*.**ScannerIsAvailable** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **ScannerIsAvailable** property settings are:

Setting	Description
True	(Default) The scanner is available. If no scanner is attached to the system, this property setting is False.
False	The scanner is unavailable.

ScanToolBarVisible Property

Description Sets or returns the visibility of this Application object's scan toolbar. This is a read/write property.

Usage *ApplicationObject*.**ScanToolBarVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **ScanToolBarVisible** property settings are:

Setting	Description
True	The scan toolbar is visible.
False	(Default) The scan toolbar is not visible.

ScrollBarsVisible Property

Description Sets or returns the visibility of the Application object's scroll bars. This is a read/write property.

Usage *ApplicationObject*.**ScrollBarsVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **ScrollBarsVisible** property settings are:

Setting	Description
True	(Default) The scroll bars are visible.
False	The scroll bars are not visible.

Note: The **ScrollBarsVisible** property must be set prior to opening the ImageFile object. For this property to take effect after an image is open, you must reopen the image.

StatusBarVisible Property

Data Type Sets or returns the visibility of this Application object's status bar. This is a read/write property.

Usage *ApplicationObject*.**StatusBarVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **StatusBarVisible** property settings are:

Setting	Description
True	(Default) The status bar is visible.
False	The status bar is not visible.

ToolBarVisible Property

Data Type Sets or returns the visibility of this Application object's standard toolbar. Read/write property.

Usage *ApplicationObject*.**ToolBarVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **ToolBarVisible** property settings are:

Setting	Description
True	(Default) The toolbar is visible.
False	The toolbar is not visible.

Top Property

Description Sets or returns the distance, in pixels, between the top edge of the physical screen and main application window. This is a read/write property.

Usage *ApplicationObject*.**Top**

Data Type Long.

Remarks The **Top** property must be set prior to opening the ImageFile object. This property only takes effect if the **Height**, **Width**, and **Left** properties are also set.

The **Top** property only returns the value that you set programmatically prior to opening the window. It does not return changes made to the window after it has been opened.

TopWindow Property

Description Sets or returns this Application object's top window flag. This is a read/write property.

Usage *ApplicationObject*.**TopWindow** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **TopWindow** property settings are:

Setting	Description
True	The application is a stay-on-top window.
False	(Default) The application is not a stay-on-top window.

Example

```
'This example makes the application window a stay-on-top window.
App.TopWindow = True
```

Visible Property

Description Returns the visibility of the Application object. This is a read-only property.

Usage *ApplicationObject*.**Visible**

Data Type Integer (Boolean).

Remarks The **Visible** property settings are:

Setting	Description
True	The application is visible.
False	(Default) The application is not visible.

WebToolBarVisible Property

Description Sets or returns the visibility of this Application object's web toolbar. This is a read/write property.

Usage *ApplicationObject*.**WebToolBarVisible** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **WebToolBarVisible** property settings are:

Setting	Description
True	The web toolbar is visible.
False	(Default) The web toolbar is not visible.

Width Property

Description Sets or returns the distance, in pixels, between the left and right edges of the Application object's window. This is a read/write property.

Usage *ApplicationObject*.**Width** [=*value*]

Data Type Long.

Remarks The **Width** property must be set prior to opening the ImageFile object. This property only takes effect if the **Top**, **Left**, and **Height** properties are also set. If you set the **Width** property to less than the minimum allowable window size, the value is ignored. The minimum setting is usually 112.

The **Width** property only returns the value that you set programmatically prior to opening the window. It does not return changes made to the window after it has been opened.

Zoom Property

Data Type Sets or returns the zoom factor used for displaying images. This is a read/write property.

Usage *ApplicationObject*.**Zoom** [=*value*]

Data Type Float.

Remarks The zoom factor is a percent value.

Example

```
'This example sets the zoom factor to 100%.
App.Zoom = 100

'This example returns the current zoom factor.
x = App.Zoom
```

Application Object Methods

The following table lists the Application object methods.

Application Object Methods

Method	Description
CreateImageViewerObject	Creates an Imaging object of the specified class.
FitTo	Displays the image at the specified zoom option.
Help	Displays online Help.
Quit	Exits this application and closes all open objects.

CreateImageViewerObject Method

Description Creates and returns an ImageFile object. The ImageFile object is empty, with no image file associated with it. Use the object's **Open** or **New** method to associate a specific image file.

Usage *ApplicationObject*.**CreateImageViewerObject**([*ObjectClass*])

Data Type Object.

Remarks This method only supports the ImageFile object, for which the setting is 1.

Example

```
'This example creates an ImageFile object.
Dim Img as Object
Set Img = App.CreateImageViewerObject(1)
```

FitTo Method

Description Displays the current image at the specified zoom option. This method updates the Application object's **Zoom** property with the actual zoom factor.

This method affects each view as follows:

View	Display
One Page	The page is zoomed.
Thumbnails	No effect — The **Application** property is changed and affects other views when they are used.
Page & Thumbnails	The page is zoomed — No effect on thumbnails.

Usage *ApplicationObject*.**FitTo** (*ZoomOption*)

Data Type Short.

Remarks ZoomOption settings are:

Setting	Description
1	Best fit
2	Fit to width
3	Fit to height
4	Actual size

Help Method

Description Displays the Imaging online Help table of contents.

Usage *ApplicationObject*`.Help`

Quit Method

Description Closes all open objects and exits the application. The Application object is no longer active or available.

Usage *ApplicationObject*`.Quit`

ImageFile Object

An ImageFile object represents an image file. An ImageFile object can have

- One Page object, representing the currently displayed page of the ImageFile object.
- One or more PageRange objects, each representing different and possibly overlapping page ranges.

ImageFile Object Properties

The following table lists the ImageFile object properties.

ImageFile Object Properties

Property	Description
ActivePage	Sets or returns the ImageFile object's current page number.
Application	Returns the Application object.
FileType	Returns the ImageFile object's file type.
Name	Returns the name of the active image file.
OCRLaunchApplication	Launches an application with an output file after OCR[a] processing is complete.
OCROutputFile	Sets or returns the output file for OCR processing.
OCROutputType	Sets or returns the output file format for OCR processing.
PageCount	Returns the number of pages in the ImageFile object.
Parent	Returns the parent of the ImageFile object.

ImageFile Object Properties

Property	Description
Saved	Returns a flag indicating whether or not the file has ever been saved.

a. TextBridge® OCR technology by ScanSoft.

ActivePage Property

Description Sets or returns the ImageFile object's active page number. This is a read/write property.

Setting the **ActivePage** property to a page number causes that page to become active, which updates the display if the Application object is visible. Refer to the Application object's **ImageView** property for more information about the relationships between the active page and different views of the page.

Page selection and navigation by the end-user have no effect on the **ActivePage** property. The active page is always the active page according to automation.

Note: If you set the **ActivePage** property to a page number beyond those contained in the document, an error is returned.

Usage *ImageFileObject*.**ActivePage** [=*value*]

Data Type Long.

Remarks The number is the page number value.

See Also ApplicationObject.**ImageView** property.

Application Property

Description Returns the Application object. This is a read-only property.

Usage *ImageFileObject*.**Application**

Data Type Object.

Example

```
'This example returns the Application object.
Dim Parent As Object
Set Parent = Img.Application
```

FileType Property

Description Returns the file type of this ImageFile object. This is a read-only property.

Usage *ImageFileObject*.**FileType**

Data Type Short.

Remarks The **FileType** property settings are:

Setting	Description
0	Unknown
1	TIFF
2	Not supported
3	BMP
4	PCX
5	DCX
6	JPG-JFIF
7	XIF
8	GIF
9	WIFF

Name Property

Description Returns a string that contains the name of the active image file. This is a read-only property.

Usage *ImageFileObject*.**Name**

Data Type String.

OCRLaunchApplication Property

Description Launches the Application object with an output file after OCR processing is complete. This is a read/write property.

Usage *ImageFileObject*.**OCRLaunchApplication** = [{*True*|*False*}]

Data Type Integer (Boolean).

Remarks The **OCRLaunchApplication** property settings are:

Setting	Description
True	(Default) Launch the application.
False	Do not launch the application.

OCROutputFile Property

Description Sets or returns the output file name. If blank, the **SaveAs** dialog box is displayed. This is a read/write property.

Usage *ImageFileObject*.**OCROutputFile** = [*FileName*]

Data Type String.

OCROutputType Property

Description Sets or returns the output file type. This is a read/write property.

Usage *ImageFileObject*.**OCROutputType** = [*Type*]

Data Type Long.

Remarks The **OCROutputType** property results are:

Setting	Description
0	Word for Windows/RTF
1	WordPerfect
2	HTML
3	Text

PageCount Property

Description Returns the number of pages in this ImageFile object. This is a read-only property.

Usage *ImageFileObject*.**PageCount**

Data Type Long.

Parent Property

Description Returns the parent of the ImageFile object. This is a read-only property.

Usage *ImageFileObject*.**Parent**

Data Type Object.

Example

```
'This example returns the parent of the ImageFile object.
Dim App As Object
App = Img.Parent
```

Saved Property

Description Returns the saved state of the ImageFile object. Read-only property.

Usage *ImageFileObject*.**Saved**

Data Type Integer (Boolean).

Remarks The Saved property settings are:

Setting	Description
True	The ImageFile object has been saved and has not changed since it was last saved.
False	The imageFile object has never been saved and has changed since it was created; or, it has been saved but has changed since it was last saved.

Example

```
'This example returns the saved state of the file.
bIsSaved = Img.Saved
```

ImageFile Object Methods

The following table lists the ImageFile object methods.

ImageFile Object Methods

Method	Description
AppendExistingPages	Appends existing pages to the end of the ImageFile object.
Close	Closes the ImageFile object.
CreateContactSheet	Saves a contact sheet rendition of the ImageFile object.
FindOIServerDoc	Finds Global 360 Imaging 1.x documents and Execute360 Imaging documents. Not available when the application is running as an embedded server.
Help	Displays online Help.
InsertExistingPages	Inserts existing pages in the ImageFile object.
New	Creates a new blank ImageFile object. Not available when the application is running as an embedded server.
Ocr	OCRs opened Image File.
Open	Opens the ImageFile object. Not available when the application is running as an embedded server.
Pages	Returns a Page or PageRange object for the ImageFile object.
Print	Prints the ImageFile object.
RotateAll	Rotates all ImageFile object pages.
Save	Saves changes to the ImageFile object.
SaveAs	Saves the ImageFile object under another name.
SaveCopyAs	Saves a copy of the ImageFile object. The application must be running as an embedded server.

ImageFile Object Methods (cont.)

Method	Description
Update	Updates the ImageFile object embedded within the container application with the current data from the server application. The application must be running as an embedded server.

AppendExistingPages Method

Description Appends specified page(s) to the end of the current ImageFile object. If the page(s) being appended come from an image file of a type different than the active image file, the pages are converted before being appended. After appending page(s), all PageRange objects are invalid. You can optionally display a dialog box that allows the end-user to select a file from which to append page(s).

Usage

```
ImageFileObject.AppendExistingPages [ImageFile],[Page],
                                    [Count],[DisplayUIFlag]
```

Arguments The **AppendExistingPages** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	The image file from which pages will be appended (source image file).
Page	Long	The page from which to start appending pages (in the source image file).
Count	Long	The number of pages to append.
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to select an image file to append. **False** (Default) — Does not display a dialog box. If you specify **True** and the selected file is a multi-page file, the user is prompted to select the pages to append.

Example

```
'This example appends the first page from the file, BW.TIF.
Img.AppendExistingPages "c:\bw.tif", 1

'This example appends a file selected from a dialog box to the
'currently displayed image file. After the user selects a file
'to append, the application prompts the user to specify the
'starting page number and the number of pages to append from
'the selected file.
Img1.AppendExistingPages "", 0, 0, True

'This example appends pages to an Imaging Server 1.x file.
ImgFileObj.AppendExistingPages
      å "Image://nqa11\SYS:\tmp\3PAGES.tif", 1, 3
```

```
'This example appends pages to an Imaging Server 1.x document.
ImgFileObj.AppendExistingPages
      å "Image://PATRIOTS\CABINET\DRAWER\FOLDER\doc1", 3, 2

'This example appends pages to an Execute360 Imaging Server
'document.
ImgFileObj.AppendExistingPages "Imagex://sixpage",  1, 6
```

Close Method

Description Closes the ImageFile object. Closing an ImageFile object deletes it; all Page and PageRange objects associated with it are also deleted. The Application object no longer has an ImageFile object associated with it.

Usage *ImageFileObject*.**Close** [*SaveChangeFlag*]

Data Type Integer (Boolean).

Remarks The **Close** method SaveChangeFlag argument has the following settings:

Setting	Description
True	Changes are saved when the image file closes.
False	(Default) Changes are not saved when the image file closes.

CreateContactSheet Method

Description Saves a contact sheet rendition of the ImageFile object. This method is unavailable when the Application is running as an embedded server.

Usage *ImageFileObject*.**CreateContactSheet** (*ImageFile*, [*IncludeAnnotations*], [*OpenAfterSave*])

Data Type String.

Arguments The **CreateContactSheet** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	The image file object.
IncludeAnnotations	Integer	Option to include annotations on the image stamps.
OpenAfterSave	Integer	Option to open the contact sheet file after it has been created.

FindOIServerDoc Method

Description Finds 1.x documents or Execute360 Imaging documents. This method displays an Imaging server document **Find** dialog box, from which the user may search for 1.x documents or Execute360 Imaging documents. After the user selects a document and chooses the **Open** button, the **Find** dialog box is closed and returns the selected document name, with a path, to the user. A null string is returned if the user chooses **Cancel** in the **Find** dialog box. The user may use the returned document name string as input for the Image Object **Open** method.

Data Type String.

Usage *ImageFileObject*.**FindOIServerDoc**

Help Method

Description Displays the Imaging online Help table of contents.

Usage *ImageFileObject*.**Help**

InsertExistingPages Method

Description Inserts page(s) into the ImageFile object.

Page(s) to be inserted must come from an existing file. If the pages being inserted come from an image file of a type different than the active image file, the pages are converted before being inserted. After inserting page(s), all PageRange objects are invalid. You can optionally cause a dialog box to open for the end-user to select a file from which to insert page(s).

Usage *ImageFileObject*.**InsertExistingPages** (*ImageFile*, *ImagePage*, *Count*, *Page*, *DisplayUIFlag*)

Arguments The **InsertExistingPages** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	The image file from which page(s) are to be inserted (the source image file).
ImagePage	Long	The page before which the new page(s) are to be inserted.
Count	Long	The number of pages to insert.
Page	Long	The page in the source image file from which to start inserting pages.

Parameter	Data Type	Description
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to select a source image file. **False** (Default) — Does not display a dialog box. If you specify **True** and the selected file is a multi-page file, the user will be prompted to select the pages to append.

Example

```
'This example inserts pages 4 and 5 from the file BW.TIF
'before page 1.
Img.InsertExistingPages "c:\bw.tif", 1, 2, 4

'This example inserts page(s) into the current file at the
'current page. (A dialog box prompts the user for the image
'file to be selected for insertion. Another dialog box
'prompts for a page range.) Page, count, and pagenumber
'arguments are required but ignored when dialogflag is True.
Img.InsertExistingPages "", 1, 1, 2, True

'This example inserts pages in an Imaging Server 1.x file.
      å ImgFileObj.InsertExistingPages
         "Image://nqa11\SYS:\tmp\3PAGES.tif", 2, 3, 1

'This example inserts pages in an Imaging Server 1.x document.
ImgFileObj.InsertExistingPages
      å"Image://PATRIOTS\CABINET\DRAWER\FOLDER\doc1", 2, 3, 1

'This example inserts pages in an Execute360 Imaging Server
document.
ImgFileObj.InsertExistingPages "Imagex://sixpage",  1, 2, 5
```

New Method

Description Displays a dialog box that allows the end-user to create a new ImageFile object that contains one blank page.

> **Note:** This method is not available when application is running as an embedded server.

Creating a new ImageFile object causes the new object to become active. If the active ImageFile object is unsaved, the end-user is prompted to save it before the new object is created.

No image file is associated with the object until you save it. The file type of the new object is the same as the file type of the active object.

Usage *ImageFileObject*.**New** ([*DisplayUIFlag*])

Remarks The **New** method has the following parameter:

Parameter	Data Type	Description
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to create a new image file. **False** (Default) — Does not display a dialog box.

Example

```
'This example creates a new image object.
'Create the image object
Dim App, Img As Object
Set App = CreateObject("Imaging.Application")
Set Img = App.CreateImageViewerObject(1)
'Call the image object New Method
Img.New
```

Ocr Method

Description OCRs all image file pages.

Usage *ImageFileObject*.**Ocr**

Remarks The Image file must be open. The **Ocr** method uses the **OcrOutputFile** and **OcrOutputFileType** properties.

Example

```
'This example performs an OCR on an image object.
Dim App, Img As Object
Set App = CreateObject("Imaging.Application")
Set Img = App.CreateImageViewerObject(1)
Img.Open "d:\pcx.tif"
Img.Ocr
```

Open Method

Description Opens an image file in the parent application window. This associates an image file with the ImageFile object. If a file is currently open, it should be closed before a new file is opened. (See the **Close** Method).

> **Note:** This method is unavailable when the application is running as an embedded server.

The Imaging application has the focus after an **Open**. You can reset the focus programmatically after an **Open**, if desired.

Usage *ImageFileObject*.**Open**(*ImageFile*,[*IncludeAnnotation*],[*Page*], [*DisplayUIFlag*])

Remarks The **Open** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	Name string of the ImageFile object to open.
IncludeAnnotation	Flag	**True** (Default)— The image has annotations that are displayed. **False —** The image has annotations that are not displayed.
Page	Long	Page number in the image file to display. This parameter must be a constant, or use the **ActivePage** property to specify the page that you want displayed when you open the file.
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to select a file to open. **False** (Default) — Does not display a dialog box.

Example

```
'This example opens an image file named 5page.tif:
Img.Open "C:\images\5page.tif"

'This example opens the same file to page 4 with annotations
'displayed:
Img.Open "C:\images\5page.tif",TRUE,4

'This example opens a dialog box so the user can select a
'file to open:
Img.Open "",,,TRUE

'This example opens an Imaging Server 1.x file.
Img.Open "Image://nqa11\SYS:\tmp\3PAGES.tif", TRUE, 1

'This example opens an Imaging Server 1.x document.
Img.Open "Image://PATRIOTS\CABINET\DRAWER\FOLDER\doc1"

'This example opens an Execute360 Imaging document.
Img.Open"Imagex://sixpage"
```

See Also ApplicationObject.Edit.

Pages Method

Description Returns the Page or PageRange object for the ImageFile object.

Usage *ImageFileObject*.**Pages**(*StartPage*, *EndPage*)

Data Type Long.

Remarks If you specify one page number, this method returns a Page object. If you specify two page numbers, this method returns a PageRange object. To return a range of pages, specify the starting page number and ending page number. The first page number can be a variable, but the second page number must be a constant.

The **Pages** method uses these parameters:

Parameter	Data Type	Description
StartPage	Long	The starting page of the page range to be returned.
EndPage	Long	The ending page of the page range to be returned.

Example

```
'This example returns a Page object and a PageRange object.
Dim Page As Object
Dim PageRange As Object
Set Page = Img.Pages(1)
Set PageRange = Img.Pages(1,3)
```

Print Method

Description Prints the image file associated with the ImageFile object. You can optionally display a dialog box to allow the end-user to select the print options.

Usage *ImageFileObject*.**Print** ([*DisplayUIFlag*])

Remarks The **Print** method DisplayUIFlag argument has the following settings:

Setting	Description
True	Displays a dialog box that allows the end-user to select print file options.
False	(Default) No dialog box is displayed.

Example

```
'This example prints the specified image file.
x = Img.Print
```

RotateAll Method

Description Rotates all ImageFile object pages. Pages are rotated clockwise in 90 degree increments.

Usage *ImageFileObject*.**RotateAll**

Example

```
'This example rotates all pages of the currently displayed image.
Img.RotateAll
```

Save Method

Description Saves changes to the ImageFile object. If no image file is associated with the ImageFile object, the **SaveAs** method is executed instead of the **Save** method.

Usage *ImageFileObject*.**Save**

SaveAs Method

Description Saves the ImageFile object as another ImageFile object. Copies its image file and renames it.

This method allows you to specify the new object's image parameters. If specified, the file can be converted from one type to another. The current image file is closed without being saved and the Save As object becomes the active image file. You can optionally display a dialog box that allows the end-user to name the file for the first time or select a file to overwrite.

Usage *ImageFileObject*.**SaveAs** (*ImageFile*, [*FileType*], [*DisplayUIFlag*])

Data Type String.

Remarks The **SaveAs** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	The destination's ImageFile object name string.
FileType	Short	The file type that you want to save the image as. This number must be a constant. It must be present in the command if the dialog flag option is used, even though its value is ignored when the DisplayUIFlag is set to True.
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to enter or select a filename and options for saving the file. **False** (Default) — Does not display a dialog box.

The **SaveAs** method FileType argument settings are:

Setting	Description
1	TIFF
2	Not supported
3	BMP

Example

```
'This example saves a file in TIF format.
Img.SaveAs "picture1.tif", 1

'This example opens a Save As dialog box so that the end-user can
'name the file for the first time or overwrite an existing file:
Img.SaveAs "", 0, True
```

SaveCopyAs Method

Description Saves a copy of the ImageFile object as another ImageFile object. You may specify the FileType of the destination file. The FileType can be TIFF or BMP.

This method allows you to specify the new object's image parameters. If specified, the file can be converted from one type to another. The current image file remains the active image file. This method can only be used after launching the embedded server application in a separate window.

Usage *ImageFileObject*.**SaveCopyAs** (*ImageFile*, *FileType*, *DisplayUIFlag*)

Data Type String.

Remarks The **SaveCopyAs** method has the following parameters:

Parameter	Data Type	Description
ImageFile	String	The destination's ImageFile object name string.
FileType	Short	The image file type that you want to save the image as. This number must be a constant. It must be present in the command if the dialog flag option is used, even though its value is ignored when the DisplayUIFlag is set to True.
DisplayUIFlag	Flag	**True** — Displays a dialog box that allows the end-user to enter or select a filename and options for saving the file. **False** (Default) — Does not display a dialog box.

Update Method

Description Updates the ImageFile object embedded within the container application with the current data from the server application.

This method can only be used after launching the embedded server application in a separate window.

Usage *ImageFileObject*.**Update**

Page Object

A Page object represents a single page in an ImageFile object. Page objects can only be accessed by using the **Pages** method of the parent ImageFile object.

Page Object Properties

The following table lists the Page object properties.

Page Object Properties

Property	Description
Application	Returns the Application object.
CompressionInfo	Returns the page's compression information.
CompressionType	Returns the page's compression type.
Height	Returns the page's height.
ImageResolutionX	Sets or returns the page's horizontal resolution.
ImageResolutionY	Sets or the returns page's vertical resolution.
Name	Returns the page number of this page.
PageType	Returns the page's image type.
Parent	Returns the parent of the Page object.
ScrollPositionX	Sets or returns this page's horizontal scroll position.
ScrollPositionY	Sets or returns this page's vertical scroll position.
Width	Returns the page's width.

Application Property

Description Returns the Application object. This is a read-only property.

Usage *PageObject*.**Application**

Data Type Object.

Example

```
'This example returns the Application object.
Dim Img As ObjectDim Parent As ObjectSet Parent =
      å Img.Pages(1).Application
```

CompressionInfo Property

Description Returns this page's compression information. This is a read-only property.

Usage *PageObject*.**CompressionInfo**]

Data Type Long.

Remarks The **CompressionInfo** property settings are:

Setting	Description
0	No compression options set. Only applicable to uncompressed image files.
1	EOL (Include/expect End Of Line). Each line is terminated with an end-of-line bit. Not used for JPEG compression.
2	Packed Lines (Byte align new lines). Not used for JPEG compression.
4	Prefixed EOL (Include/expect prefixed End Of Line). Each strip of data is prefixed by a standard end-of-line bit sequence. Not used for JPEG compression.
8	Compressed LTR (Compressed bit order, left to right). The bit order for the compressed data is the most significant bit to the least significant bit. Not used for JPEG compression.
16	Expanded LTR (Expanded bit order, left to right). The bit order for the expanded data is the most significant bit to the least significant bit. Not used for JPEG compression.
32	Negate (Invert black and white on expansion). Indicates the setting of the Photometric Interpretation field of a TIFF file. Not used for JPEG compression.
64	Low Resolution/High Quality (JPEG compression only).
128	Low Resolution/Medium Quality (JPEG compression only).
256	Low Resolution/Low Quality (JPEG compression only).
512	Medium Resolution/High Quality (JPEG compression only).
1024	Medium Resolution/Medium Quality (JPEG compression only).
2048	Medium Resolution/Low Quality (JPEG compression only).
4098	High Resolution/High Quality (JPEG compression only).
8196	High Resolution/Medium Quality (JPEG compression only).
16392	High Resolution/Low Quality (JPEG compression only).

Remarks Image files that do not have a compression type of JPEG will have a value between 1 and 63. This value is a combination of the values of 1 to 32. For JPEG files, the value is from 64 to 16384, and is only one of these values.

Example

```
'This example returns the page's compression information.
x = Img.Pages(1).CompressionInfo
```

CompressionType Property

Description Returns this page's compression type. This is a read-only property.

Usage *PageObject*.**CompressionType**[=*value*]

Data Type Short.

Remarks The **CompressionType** property settings are:

Setting	Description
0	Unknown
1	No Compression
2	Group 3 1D FAX
3	Group 3 Modified Huffman
4	PackBits
5	Group 4 2D FAX
6	JPEG
7	Reserved
8	Group 3 2D FAX
9	LZW

Example

```
'This example returns this page's compression type.
x = Img.Pages(1).CompressionType
```

Height Property

Description Returns this page's height in pixels. This is a read-only property.

Usage *PageObject*.**Height**

Data Type Long.

Example

```
'This example returns this page's height in pixels.
x = Img.Pages(1).Height
```

ImageResolutionX Property

Description Sets or returns this page's horizontal resolution, in dots-per-inch. An error occurs when a value less than 20 or greater than 1200 dpi is specified. This is a read/write property.

Usage *PageObject*.**ImageResolutionX** [= *value*]

Data Type Long.

Example

```
'This example sets this page's horizontal resolution.
Img.Pages(1).ImageResolutionX = 200

'This example returns this page's horizontal resolution.
XRes = Img.Pages(1).ImageResolutionX
```

ImageResolutionY Property

Description Sets or returns this page's vertical resolution, in dots-per-inch. An error occurs when a value less than 20 or greater than 1200 dpi is specified. This is a read/write property.

Usage *PageObject*.**ImageResolutionY** [= *value*]

Data Type Long.

Example

```
'This example sets this page's vertical resolution.
Img.Pages(1).ImageResolutionY = 200

'This example returns this page's vertical resolution.
YRes = Img.Pages(1).ImageResolutionY
```

Name Property

Description Returns the page number of the page in the ImageFile object. This is a read-only property.

Usage *PageObject*.**Name**

Data Type Long.

Example

```
'This example returns the page number of the page in the
'ImageFile object.
x = Img.Pages(1).Name
```

PageType Property

Description Returns the page's image type. This is a read-only property.

Usage *PageObject*.**PageType**

Data Type Short.

Remarks The **PageType** property settings are:

Setting	Description
1	Black and White
2	Gray 4
3	Gray 8
4	Palettized 4
5	Palettized 8
6	RGB 24

Example

```
'This example returns the page's image type.
x = Img.Pages(1).PageType
```

Parent Property

Description Returns the parent of the Page object. This is a read-only property.

Usage *PageObject*.**Parent**

Data Type Object.

Example

```
'This example returns the parent of the Page object.
x = Img.Pages(1).Parent
```

ScrollPositionX Property

Description Sets or returns this page's horizontal scroll position, in pixels. This is a read/write property.

Usage *PageObject*.**ScrollPositionX** [=*value*]

Data Type Long.

Example

```
'This example sets this page's horizontal scroll position.
Img.Pages(1).ScrollPositionX = 200

'This example returns this page's horizontal scroll position.
xpos = Img.Pages(1).ScrollPositionX
```

ScrollPositionY Property

Description Sets or returns this page's vertical scroll position, in pixels. This is a read/write property.

Usage *PageObject*.**ScrollPositionY** [=*value*]

Data Type Long.

Example
```
'This example sets this page's vertical scroll position.
Img.Pages(1).ScrollPositionY = 200

'This example returns this page's vertical scroll position.
ypos = Img.Pages(1).ScrollPositionY
```

Width Property

Description Returns this page's width, in pixels. This is a read-only property.

Usage *PageObject*.**Width**

Data Type Long.

Example
```
'This example returns this page's width in pixels.
x = Img.Pages(1).Width
```

Page Object Methods

The following table lists the Page object methods.

Page Object Methods

Method	Description
Delete	Deletes the page.
Flip	Rotates the page 180 degrees.
Help	Displays online Help.
Ocr	OCRs Image Page.
Print	Prints the page.
RotateLeft	Rotates the page counterclockwise 90 degrees.
RotateRight	Rotates the page clockwise 90 degrees.
Scroll	Scrolls the page.

Delete Method

Description Deletes the specified page from the active object. After deleting a page, the next page is displayed (if one exists). Otherwise, the previous page is displayed.

Usage *PageObject*.**Delete**

Example
```
'This example deletes the specified page.
Img.Pages(1).Delete
```

Flip Method

Description Rotates the specified page 180 degrees. This change becomes permanent when the image file is saved.

Usage *PageObject*.**Flip**

Example
```
'This example flips the page.
Img.Pages(1).Flip
```

Help Method

Description Displays the Imaging online Help table of contents.

Usage *PageObject*.**Help**

Ocr Method

Description OCRs the image page.

Usage *PageObject*.**Ocr**

Print Method

Description Prints the page.

Usage *PageObject*.**Print**

Example
```
'This example prints the page.
x = Img.Pages(1).Print
```

RotateLeft Method

Description Rotates the page 90 degrees counterclockwise. This change becomes permanent when the image file is saved.

Usage *PageObject*.**RotateLeft**

Example
```
'This example rotates the page 90 degrees to the left.
Img.Pages(1).RotateLeft
```

RotateRight Method

Description Rotates the page 90 degrees clockwise. This change becomes permanent when the image file is saved.

Usage *PageObject*.**RotateRight**

Example
```
'This example rotates the page 90 degrees to the right.
Img.Pages(1).RotateRight
```

Scroll Method

Description Scrolls the page.

Usage *PageObject*.**Scroll** *Direction,ScrollAmount*

Remarks The **Scroll** method uses the following parameters:

Parameter	Data Type	Description
Direction	Integer	Direction in which to scroll the image: 0 — (Default) Scrolls down 1 — Scrolls up 2 — Scrolls right 3 — Scrolls Left
ScrollAmount	Long	Number of pixels to scroll the image

Example
```
'This example scrolls the page down 200 pixels.
Img.Pages(1).Scroll 0 200
```

PageRange Object

A PageRange object represents a range of consecutive pages in an ImageFile object. A page range is a set of pages starting at the **StartPage** property and ending at the **EndPage** property. PageRange objects can only be accessed by using the Pages method of the parent ImageFile object.

PageRange Object Properties

The following table lists the PageRange object properties.

PageRange Object Properties

Property	Description
Application	Returns the Application object.
Count	Returns the number of pages in this range.
EndPage	Returns or sets the page number of the last page in the range.
Parent	Returns the parent of the PageRange object.
StartPage	Returns or sets the page number of the first page in the range.

Application Property

Description Returns the Application object. This is a read-only property.

Usage *PageRangeObject*.**Application**

Description Object.

Count Property

Description Returns the number of pages in this range. This is a read-only property.

Usage *PageRangeObject*.**Count**

Data Type Long.

EndPage Property

Description Returns or sets the page number of the last page in the range. This is a read/write property.

Usage *PageRangeObject*.**EndPage** [=*value*]

Data Type Long.

Remarks This property setting is the number of the last page. The value of EndPage must be greater than or equal to the value of StartPage.

Parent Property

Description Returns the parent of the PageRange object. This is a read-only property.

Usage *PageRangeObject*.**Parent**

Data Type Object.

Example
```
'This example returns the parent of the PageRange object.
x = Img.Pages(1,7).Parent
```

StartPage Property

Description Returns or sets the page number of the first page in the range. This is a read/write property.

Usage *PageRangeObject*.**StartPage** [=*value*]

Data Type Long.

Remarks This property setting is the number of the first page. The value of StartPage must be less than or equal to the value of EndPage.

PageRange Object Methods

The following table lists the PageRange object methods.

PageRange Object Methods

Method	Description
Delete	Deletes the page range.
Ocr	OCRs the page range.
Print	Prints the page range.

The **Delete**, **Ocr**, and **Print** methods of the PageRange object use the following parameters:

Parameter	Data Type	Description
StartPage	Long	First page to be deleted.
NumPages	Long	Number of pages to be deleted, including the **StartPage**.

Delete Method

Description Removes pages from the ImageFile object. After deleting a PageRange object, all page ranges are invalid.

Usage *PageRangeObject*.**Delete**( )

Example
```
'This example deletes the pages 1 through 3.
Img.Pages(1,3).Delete
```

Ocr Method

Description OCRs the page range.

Usage *PageRangeObject*.**Ocr**( )

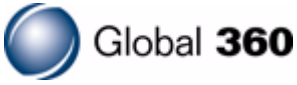
Example
```
'This example OCRs pages 2 through 6.
x = Img.Pages(2,6).Ocr
```

Print Method

Description Prints the page range.

Usage *PageRangeObject*.**Print**( )

Example
```
'This example prints pages 1 through 5.
x = Img.Pages(1,5).Print
```


Global 360