



Imaging for Windows[®] Developer's Guide

Imaging for Windows[®]

Developer's Guide

Copyright© 1998-2008, Global 360, Inc.
www.global360.com

715-C008A

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.

Microsoft and Windows are registered trademarks and Vista is a trademark of Microsoft Corporation in the USA and in other countries.

Other product names mentioned in this guide may be trademarks or registered trademarks of their respective companies.

Contents

Imaging for Windows[®] Developer's Guide

About This Guide

Purpose	xiv
Prerequisites	xiv
Documentation Conventions	xv
Related Information	xvi
Support	xvi

1 Introduction

Introducing Imaging for Windows	2
Sample Code	3

2 Adding Imaging Using ActiveX Controls

Introduction	6
ActiveX Demonstration Projects	7
ActiveX Sample Applications	8
Loading the Controls	9
Demonstration Projects	10
Unlocking the Controls with a Developer License Code File	10
Displaying an Image and Applying Fit-To Options	11
Fit-To Options Defined	11
Example	12
FitTo Options Project	13
Converting an Image	16

Image Conversion Defined	16
Example 17	
Example 19	
Example 21	
Example 22	
Example 23	
Convert Image Project	23
Copying An Image	28
Clipboard Functions Defined	28
Clipboard Copy and Cut	28
Clipboard Paste	29
Image Selection	29
Annotation Selection	30
Example 30	
Copy Image Project	31
Printing An Image	34
Image Printing Defined	34
Example 35	
Print Image Project	36
Scanning an Image Using a Template	42
Template Scanning Defined	42
Example 44	
Template Scan Project	46
Managing an Image File Using Thumbnails	55
Thumbnails Defined	55
Example 55	
Thumbnail Sorter Project	56
Subtracting the Value of X	70
Unloading a Multipage Image File	72
Multipage Image Files Defined	72
Page-Related Properties and Methods	74
Image Admin	74
Image Edit	75
Image Scan	75
Image Thumbnail	76
Example 77	
Unload Project	78

3 Developing Client/Server Applications

Imaging Server Concepts 84

- File Type Support 85
- Standard Dialog Boxes 85
- Image Files and Server Documents 86
- Interacting with Imaging 1.x Servers 86
 - Image File Volume 86
 - Document Volume 87
- Interacting with Execute360 Servers 87

Imaging 1.x Server Programming Considerations 88

- Logging On To the Server 89
- Setting Imaging 1.x Server Options 90
 - File Location for Document Pages (FileStgLoc1x Property) 94
 - Force Lower-Case File Names (ForceLowerCase1x Property) 95
 - Link Files On Reference (ForceFileLinking1x Property) 95
 - Delete Files With Pages (ForceFileDeletion1x Property) 97
- Browsing for Volumes or Image Files and Server Documents 98
 - Browsing for Volumes 98
 - Browsing for Files and Documents 100
- Querying Imaging 1.x Documents 102
- Finding Imaging 1.x Documents 103
 - Name and Location Tab 103
 - Date Modified Tab 106
 - Keywords Tab 107
 - Opening the Selected Document 107
- Saving 1.x Image Files and Documents 108

Execute360 Server Programming Considerations 109

- Logging On To the Server 109
- Querying Execute360 Documents 110
- Finding Execute360 Documents 111
 - Name and Location Tab 111
 - Field Tab 112
 - Opening the Selected Execute360 Document 113

Demonstration Project 113

- Zooming an Image 114
 - Zooming an Entire Image Page 114
 - Zooming a Portion of an Image Page 114
 - Example 115
- Annotations 115
 - Image Annotation Tool Button Control 118
 - Image Edit Control 119
 - Example 120
- The Image Server Project 121
 - Setting Server Options 123
 - Browsing for Imaging 1.x File and Document Volumes 124
 - Opening 1.x Files and Documents 127
 - Querying 1.x Document Manager Databases 129
 - Zooming an Image 149
 - Invoking the Standard Annotation Tool Palette 151

4 Image-Enabling a Web Page

Defining the Controls 154

- Class Identifiers for Imaging Controls 155

Demonstration Project 156

- Project Overview 157
- Starting Imgctrls 157
- Initialization 157
- Loading an Image 159
- Invoking the FitTo method 161
- Visiting Other Pages 162
- Rotating a Page 163
- Persistence 163

5 Compatibility with Microsoft .NET

Introduction 166

Package Contents 166

Software Development Tips 167

A Imaging ActiveX Sample Applications

Overview 172

Requirements 172

Sample Applications 172

Image Editor Samples 173

Sample Application 173

Image Editor 174

Function Specific Samples 174

Image Print 175

Image Properties 176

Image Scan 176

Image Thumbnails 177

Imaging Flow Samples 177

Flow Program 177

Flow Variables 178

B Imaging ActiveX Tips and Tricks

Overview 180

Miscellaneous Programming Tips 180

Image File Management Tips 184

Annotation Tips 188

Optical Character Recognition Tips 190

C Adding Imaging Using Automation

Overview 194

Imaging Components 195

Imaging Application 196

Imaging Flow 197

Imaging Preview 198

Invoking Imaging for Windows 199

Command Line Invocation 199

OLE 200

Embedded Image Files 201

Linked Image Files 201

When to Use Automation 202

The Object Hierarchy 203

Application Object 203

ImageFile Object 204

Page Object 204

PageRange Object 204

Imaging Application Modes 205

Automation Server Mode 205

Embedded Server Mode 205

Examples 206

As an Automation Server Application 206

As an Embedded Server Application 210

Demonstration Project 212

View Modes 212

One Page 213

Thumbnail 214

Page and Thumbnails 215

Example 216

The Automation From Excel Project 217

Opening the Spreadsheet File 218

Opening and Displaying the Image File 219

Obtaining the Page Count 222

Rotating an Image Page	223
Setting the One Page View Mode	224
Setting the Thumbnail View Mode	225
Setting the Page and Thumbnails View Mode	226
Closing the Image File and the Imaging Application	227

D Automation Lexicon

Overview 230

Application Object 230

Application Object Properties	230
ActiveDocument Property	232
AnnotationPaletteVisible Property	232
Application Property	232
AppState Property	233
DisplayScaleAlgorithm Property	233
Edit Property	234
FullName Property	234
Height Property	234
ImagePalette Property	235
ImageView Property	235
ImagingToolBarVisible Property	236
Left Property	236
Name Property	237
Parent Property	237
Path Property	237
ScannerIsAvailable Property	237
ScanToolBarVisible Property	237
ScrollBarsVisible Property	238
StatusBarVisible Property	238
ToolBarVisible Property	239
Top Property	239
TopWindow Property	239
Visible Property	240
WebToolBarVisible Property	240
Width Property	240
Zoom Property	241
Application Object Methods	241

- CreateImageViewerObject Method 241
- FitTo Method 242
- Help Method 242
- Quit Method 242

ImageFile Object 243

- ImageFile Object Properties 243
 - ActivePage Property 244
 - Application Property 244
 - FileType Property 245
 - Name Property 245
 - OCRLaunchApplication Property 245
 - OCROutputFile Property 246
 - OCROutputType Property 246
 - PageCount Property 246
 - Parent Property 246
 - Saved Property 247
- ImageFile Object Methods 247
 - AppendExistingPages Method 248
 - Close Method 249
 - CreateContactSheet Method 249
 - FindOIServerDoc Method 250
 - Help Method 250
 - InsertExistingPages Method 250
 - New Method 252
 - Ocr Method 252
 - Open Method 253
 - Pages Method 254
 - Print Method 255
 - RotateAll Method 255
 - Save Method 255
 - SaveAs Method 255
 - SaveCopyAs Method 256
 - Update Method 257

Page Object 258

- Page Object Properties 258
 - Application Property 258
 - CompressionInfo Property 259
 - CompressionType Property 260

Height Property	260
ImageResolutionX Property	261
ImageResolutionY Property	261
Name Property	261
PageType Property	262
Parent Property	262
ScrollPositionX Property	262
ScrollPositionY Property	263
Width Property	263
Page Object Methods	263
Delete Method	264
Flip Method	264
Help Method	264
Ocr Method	264
Print Method	264
RotateLeft Method	264
RotateRight Method	265
Scroll Method	265
PageRange Object	266
PageRange Object Properties	266
Application Property	266
Count Property	266
EndPage Property	266
Parent Property	267
StartPage Property	267
PageRange Object Methods	267
Delete Method	268
Ocr Method	268
Print Method	268

About This Guide

This guide explains imaging concepts and provides information about using the Imaging ActiveX controls to implement imaging features in your applications.

In this Chapter

Purpose	xiv
Prerequisites	xiv
Documentation Conventions	xv
Related Information	xvi
Support	xvi

Purpose

The *Developer's Guide* describes Global 360 Imaging for Windows® Developer Resources and provides software developers and IT professionals with the information they need to produce and support image-enabled applications.

This guide is a technical resource that supplements the online help included with Global 360 Imaging for Windows Developer Resources.

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 a Global 360 Imaging (1.x) server or Execute360 server, you should be familiar with navigating document databases in those environments.

Documentation Conventions

This guide uses the following conventions.

Conventions	Description
Image, Display, PasteCompleted	Words in bold with initial capitalization indicate names of properties, methods, and events.
<i>Object, arglist</i>	In the syntax section, words in lowercase italics indicate placeholders for information you must provide.
[<i>expressionlist</i>]	In the syntax section, items appearing inside square brackets are optional.
{ <i>True</i> <i>False</i> }	In the syntax section, braces and a vertical bar indicate a mandatory choice between two or more items.
Dim x As IFontDisp	This font is used for code examples.
➡	This character indicates that a line of code was too long to fit on one line in the Example window.

You should keep the code on one line in your program, or use the line continuation character provided by your programming environment. Refer to the documentation that came with your programming environment for more information on the line continuation character and its proper placement in your code.

Note that the ➡ character does not necessarily indicate the proper place for a line continuation character in your code.

Related Information

For instructions about how to use the Imaging ActiveX controls, access the online help system for the controls from your development environment.

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

Introduction

This chapter provides an introduction to Imaging for Windows[®]. It also explains how to use the sample code that is included on the release media.

In this Chapter

Introducing Imaging for Windows	2
Sample Code	3

Introducing Imaging for Windows

Imaging for Windows is a multi-faceted product that enables users to transform paper documents and faxes into electronic documents for viewing, annotating, editing, converting, printing, and sharing.

Imaging for Windows includes a rich set of development tools and methods that let you — the software developer — add Imaging functions to your applications.

To add Imaging functions, you must be using a development environment that supports OLE, Automation, and ActiveX controls such as:

- Microsoft Visual Basic
- Microsoft Visual C++

Microsoft Office (Visual Basic for Applications)

Note: If a newer version of Internet Explorer is installed after Imaging for Windows, or after an Imaging for Windows Hotfix, there may be a problem running the ActiveX controls in HTML or with Microsoft Office.

To resolve this problem, either merge the registry file, `activex compatibility - restore.reg` (included on the Release CD or in the Hotfix package), into the system registry by double-clicking the file or re-install the HotFix.

Sample Code

Sample code that was designed to help you add Imaging functions to your applications is included on the media on which your Imaging for Windows software was distributed.

Before you can use the sample code, you need to:

- Set up Imaging for Windows on your development system, if necessary.
- Set up Microsoft Excel and be familiar with Visual Basic for Applications (for the Automation demonstration project).
- Set up and be familiar with Visual Basic (for the ActiveX demonstration projects and the ActiveX sample applications).

Note: When you set up Visual Basic 6.0 or greater, you must perform a Typical installation. If you don't, the sample code may not function correctly.

Adding Imaging Using ActiveX Controls

This chapter demonstrates how to use the Imaging ActiveX controls to image-enable an application. It explains how to load the Imaging ActiveX controls into your development environment and walks you through some demonstration projects to help you get started.

In This Chapter

Introduction.....	6
Loading the Controls	9
Demonstration Projects.....	10

Introduction

Using ActiveX controls is a powerful way to include Imaging functions within your application. Imaging for Windows provides the following ActiveX controls:

Image Admin control — The Image Admin control manages administrative functions, such as: creating, opening, saving, and printing image files; appending, inserting, and deleting image pages; and entering Summary property information.

Image Annotation Tool Button control — The Image Annotation Tool Button control lets you create customized annotation toolbars for use within your application. The control links with the Image Edit control to provide annotation drawing and management functions to your end users.

Image Edit control — The Image Edit control manages all image display and annotation functions. Its huge array of properties, methods, and events provides Imaging functions, such as displaying, annotating, and editing images; rotating, flipping, and zooming images; applying compression to images; and copying, cutting, and pasting images to and from the Clipboard.

Image OCR control — The Image OCR control manages the recognition and recomposition of image files. It lets users convert images into editable text documents. Output formats include Microsoft Word/Rich Text Format (RTF), Corel® WordPerfect, Hypertext Markup Language (HTML), and text.

Image PDF control — The Image PDF control manages the display of PDF documents. It includes properties and methods that enable navigation between pages, zooming in or out to enhance readability, rotation of a single page or all pages, and printing of a page, a range of pages, or the entire document.

Image Scan control — The Image Scan control manages the scanning of documents using TWAIN-compliant image acquisition devices. TWAIN (Technology Without An Interesting Name) is an industry-standard interface between image-enabled applications and image acquisition devices.

Image Thumbnail control — The Image Thumbnail control displays and manages thumbnail renditions of individual image pages.

The Imaging ActiveX controls let you add Imaging functions to your application by making the functions an integral part of your application.

The controls are useful when you want to display images *within* a window in your application and when you want to manipulate all Imaging functions from *within* your application.

While the ActiveX controls add overhead to your application, they give you the power to determine the range of Imaging functions to be provided. And, unlike Automation, the ActiveX controls support extender as well as intrinsic events, which enable your application to respond to events that occur when users perform Imaging operations.

Depending on the Imaging functions you want to provide, image-enabling your application with ActiveX controls can require more coding when compared to Automation.

ActiveX Demonstration Projects

The ActiveX demonstration projects described later in this chapter are small Visual Basic applications that provide excellent examples of how to use the Imaging ActiveX controls to image-enable applications.

The demonstration projects show you how to:

- Display an image and apply fit-to options.
- Convert an image.
- Copy an image.
- Print an image.
- Scan images using a template.
- Reorganize an image file using thumbnails.

- Unload a multipage image file.
- Access and interact with Global 360 Imaging 1.x and Execute360 Imaging servers.

ActiveX Sample Applications

The ActiveX sample applications are relatively large Visual Basic projects that show how the Imaging ActiveX controls can be used to create comprehensive and useful image-enabled applications.

You should run each application and analyze its code to determine whether you can use the code directly in your application or as a guide to writing your own related code.

The code in each sample project is highly organized, heavily commented, and written using Hungarian notation. The sample applications show you how to:

- Create an application that is similar to the standard Imaging application.
- Develop an application that prepares separator pages for scanning several multipage documents in the Imaging Flow application.
- Perform template scanning.
- Use the Image Thumbnail control to create folder-based contact sheets.
- Print a selected portion of an image.
- Use General and Page properties to analyze image files in folders.

Appendix A describes each sample application in greater detail.

Loading the Controls








Before you can use the Imaging ActiveX controls, you must load them into your development environment: Microsoft Visual Basic, Visual C++, or Visual Basic for Applications (VBA).

Loading the controls consists of the following basic tasks:

- Selecting each Imaging ActiveX control from a list of registered ActiveX controls on your system.
- Inserting each Imaging ActiveX control icon into the controls toolbox of your development environment.

The following table lists the icons that represent each Imaging ActiveX control in the controls toolbox of your development environment.

Imaging ActiveX Toolbox Icons

Icon	Name
	Image Admin
	Image Annotation Tool Button
	Image Edit
	Image OCR
	Image PDF
	Image Scan
	Image Thumbnail

Demonstration Projects



All demonstration projects were developed using Microsoft Visual Basic.

To help you use the Imaging ActiveX controls, seven demonstration projects show you how to perform the following functions:

- Display an image and apply fit-to options.
- Convert an image.
- Copy an image.
- Print an image.
- Scan images using a template.
- Manage an image file using thumbnails.
- Unload a multipage image file..

Note: The ActiveX Controls online help system identifies the properties, methods, events, parameters, and constants that are available in Imaging for Windows®.

Unlocking the Controls with a Developer License Code File

The Imaging controls each have key features locked until a call is made to the **LicenseCheck()** method. For instance, the **ImgEdit** control does not display an image. The **LicenseCheck()** method is exposed on all controls but only one call is necessary to unlock all controls in the project.

The controls have a built-in 14 day evaluation period that begins when the **LicenseCheck()** call is first made. A call with **szLicenseFile** set to any simple string unlocks the controls and returns the remaining time for evaluation. Once the evaluation time has expired, the method returns 0 and the controls are not unlocked. A Developer License Code file is provided when the product is purchased.

Each demonstration project has code to unlock the controls during the evaluation period. It displays the number of remaining days for evaluation. If you want to run with a Developer License Code file, enter the fully qualified filename into Public Const LicensePath found in (one of) the global modules.

Displaying an Image and Applying Fit-To Options

The FitTo Options demonstration project shows how to display an image at various fit-to settings. Before walking through the demonstration project, read the following section, which explains the concept of fitting an image and describes the various fit-to options.

Fit-To Options Defined

Fit-to options govern the way an Image Edit control displays images.

The **FitTo** method of the Image Edit control lets you select — usually in response to user input — the scaling factor, or fit-to option, applied to images when they're displayed.

Most image application developers make fit-to functions available to their end users. These functions let users scale the image so it can be seen more clearly. This is particularly important when users *read* scanned documents or faxes.

You can provide the following fit-to options to your end users:



You can provide your users with even more control over a displayed image by using the **Zoom** property of the Image Edit control *in addition to* the **FitTo** method.

Best Fit — Scales the image so it appears in the entire Image Edit control. The full height or the full width of the image appears in the control, depending on which results in the least unused space.

Fit To Width — Scales the image so it matches the width of the Image Edit control.

Fit To Height — Scales the image so it matches the height of the Image Edit control.

The **ZoomMode** property determines the way a **Zoom** value affects image display. **PixelToPixel**, the default, is the most straightforward. An image zoomed to 100% has its pixels mapped directly to logical screen pixels. A 4-inch wide page at 300 DPI covers three times as much of the Desktop as a 4-inch wide page at 100 DPI.

InchToInch uses the ratio of image resolution to screen resolution to display a page in logical inches. A 4-inch wide page occupies the same portion of the Desktop whether it has a resolution of 100 DPI or 600 DPI.

When **ZoomMode** is set to **PixelToPixel**, calling **FitTo** with the **InchToInch** option displays the page as though the mode was **InchToInch** and the zoom was 100%. When **ZoomMode** is set to **InchToInch**, calling **FitTo** with the **PixelToPixel** option displays the page as though the mode was **PixelToPixel** and the zoom was 100%.

Example

Users of your application may want control over the display of image documents to make them easy to read.

Scenario

Assume Eileen receives several scanned business documents in her role as product manager for a major computer company. Because she receives these documents from others via e-mail, she has no control over how they are scanned, but she really needs to be able to read them.

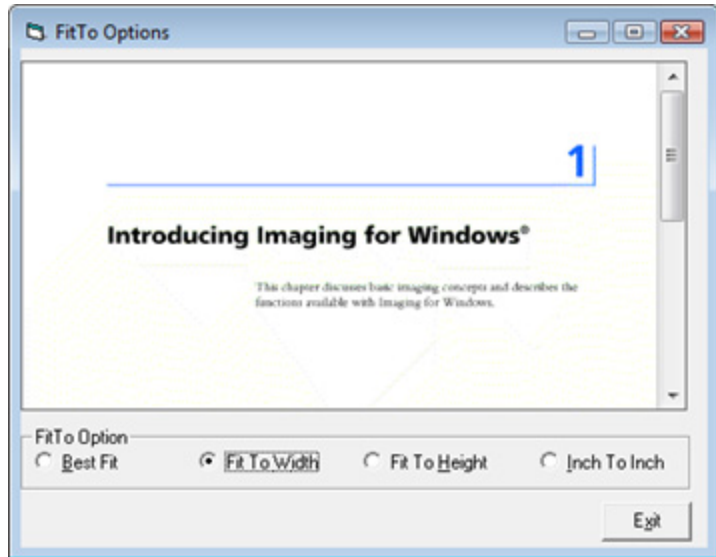
Because you included all of the Image Edit fit-to options in your application, Eileen can select the one that produces the best display quality—enabling her to view the image documents clearly and read them easily.

FitTo Options Project



The file name for the FitTo Options project is
FitTo.vbp.

The FitTo Options project demonstrates displaying an image at a variety of fit-to options



The project consists of one form and the following controls:

- One Image Admin control
- One Image Edit control
- Four Option Button controls in a control array
- One Frame control
- One Command Button control

It uses the following methods in the Image Edit control to provide the display and fit-to functions:

Display — Displays the image file specified in the **Image** property of the Image Edit control.

FitTo — Scales the image relative to the Image Edit control.

Displaying and Fitting an Image

Start the FitTo Options project. The `Form_Load()` event procedure displays an **Open** dialog box to let you select the TIFF image file you want to display.

After you select the image file, the `Form_Activate()` event procedure invokes the **FitTo** method of the Image Edit control with a parameter value of `BEST_FIT` (literal 0). This action sets the initial fit-to setting of the Image Edit control to the Best Fit option.

The procedure then sets the **Value** property of the corresponding **Option Button** control to `True`, to indicate that Best Fit is the initial fit-to mode.

Next, the `Form_Activate()` event procedure invokes the **Display** method of the Image Edit control to display the image file at the initial fit-to setting.

```
Private Sub Form_Activate()  
  
    'Initialize to a FitTo option of Best Fit  
    ImgEdit1.FitTo BEST_FIT  
    optFitTo(0).Value = True  
  
    'Display the selected image file  
    ImgEdit1.Display  
  
End Sub
```


To change the **FitTo** setting, click the desired option button in the **FitTo Setting** frame on the **FitTo Options** window. The `optFitTo_Click()` event procedure fires and executes the appropriate code in its `Select Case` statement.

Each `Case` expression corresponds to the `Index` value of the **FitTo** option buttons in the frame. Further, each `Case` expression invokes the **FitTo** method of the Image Edit control, passing to it the appropriate parameter values:

FitTo parameter — Determines the **FitTo** option applied:

- `Case 0` invokes `BEST_FIT` (literal 0).
- `Case 1` invokes `FIT_TO_WIDTH` (literal 1).
- `Case 2` invokes `FIT_TO_HEIGHT` (literal 2).
- `Case 3` invokes `INCH_TO_INCH` (literal 3).



You can control whether scrollbars appear in the Image Edit control by setting the **ScrollBars** property to the appropriate value.

Repaint parameter — Determines whether the image is refreshed immediately. All `Case` expressions invoke the **FitTo** method with a `Repaint` setting of `True`.

As you try the various FitTo options, notice the impact on the displayed image and how the scrollbars appear and disappear as needed.

```
Private Sub optFitTo_Click(Index As Integer)

    Select Case Index

        Case 0 'Best Fit
            ImgEdit1.FitTo BEST_FIT, True

        Case 1 'Fit To Width
            ImgEdit1.FitTo FIT_TO_WIDTH, True

        Case 2 'Fit To Height
            ImgEdit1.FitTo FIT_TO_HEIGHT, True

        Case 3 'Inch To Inch (Actual Size)
            ImgEdit1.FitTo INCH_TO_INCH, True

    End Select

End Sub
```

Converting an Image

This demonstration project shows how to add image conversion functions to your image-enabled applications. Before walking through the demonstration project, read the following sections, which explain the concept of image conversion.

Image Conversion Defined

Converting an image involves changing one or more of the following attributes:

- File type
- Color type
- Compression type
- Resolution
- Size

There are many reasons why your end users would want to convert an image. The following sections explain some of them.

File Type

The Imaging ActiveX controls provide three read/write file types. Your users will want to use the file type that best satisfies their requirements.

TIFF — Supports all color types and many compression types. Its image files can contain multiple pages and can store Summary property information and image annotation data separate from the actual image data.

BMP — Supports all color types. While its image files can contain only a single image page that cannot be compressed, they are readable by anyone with Windows on the computer.

JPG-JFIF — Supports 256 Shades of Gray and True Color. This image file contains a single image page that is JPEG compressed.

Example

Users of your application may want to change the file type to take advantage of the new file type's special features.

Scenario

Assume Krystina sends Tom two BMP image files of an automobile that was involved in an accident recently. Because the two BMP files are separate and quite large, Tom converts each one to the TIFF file format to take advantage of its special features. Once in the TIFF format, Tom can:

- Combine the two files into one multipage TIFF image file.
- Apply compression to save disk space.
- Annotate the images without making the annotations a permanent part of the image.
- Add Summary property information to the image file.

Color Type

The color type — also known as the page type or data type — specifies the number of colors images can have. Your users will want to use, or convert images to, the color type that best satisfies their color and storage requirements.

The factor that determines the color content of images is the number of data bits that compose each picture element (pixel). The formula for determining the color content of image documents is $2^{\text{number of bits}}$. The more color an image contains, the greater the number of data bits in each pixel.

Aesthetics aside, the most important consideration when selecting the color type is file size. The greater the number of data bits per pixel, the greater the memory and storage requirements.

The following color types are available:

Black and White — One bit makes up each pixel. Images can therefore have only two colors: black and white.

16 Shades of Gray — Four bits make up each pixel. Images can therefore have a maximum of 16 shades of gray.

256 Shades of Gray — Eight bits make up each pixel. Images can therefore have a maximum of 256 shades of gray.

16 Colors — Four bits make up each pixel. Images can therefore have a maximum of 16 colors.

256 Colors — Eight bits make up each pixel. Images can therefore have a maximum of 256 colors.

True Color — Twenty-four bits make up each pixel. Images can therefore have a maximum of 16,777,216 colors.

Example

Users of your application may want to change a color type to save disk space.

Scenario

Assume Tom scans a text-only insurance document in True Color and then sends the image to Krystina so she can view it. When Krystina receives the image, she notices that its file size is a little large for a text-based image. Realizing that color is not a requirement for this type of image, she converts its color type to Black and White. File size drops 27%, and the document is completely readable.

Compression Type

When saved to disk, images can require a large amount of storage space. Compression is a technique that reduces this large disk space requirement. The more compression applied when saving images, the lower the disk space requirement. Your users will want to use, or convert images to, the compression type that best satisfies their storage requirements.

The following compression types are available.

CCITT Group 3 (1d) Fax — Should be used to compress black-and-white TIFF images when users anticipate sending them as faxes over unreliable data links.

CCITT Group 3 (1d) Modified Huffman — Should be used to compress black-and-white TIFF images when users anticipate sending them as faxes over unreliable data links, and they require increased compression over that provided by Group 3 (1d) Fax.

CCITT Group 4 (2d) Fax — Should be used to compress black-and-white TIFF images when users anticipate saving them to disk or sending them as faxes over reliable data links, such as ISDN, X.25, or e-mail.

Note: With CCITT compression types, users can set the Reversed Bit Order compression option, which signifies that the compressed data codes begin at the left, most significant bit (MSB) of each byte and are ordered from MSB to the least significant bit (LSB).

Users should employ this option when they need to exchange images with someone whose image software cannot handle non-reversed image documents.



JPEG is a lossy compression type, which means that some data is altered and lost during compression. Usually, data alteration and loss are not significant. Lossy compression types often offer higher compression ratios than do lossless types, such as LZW.

JPEG — Can be used to compress TIFF images with a color type of 256 Shades of Gray or True Color. Should be used when users want to significantly reduce the storage requirement and they don't mind if the image is altered by the compression process.

When users select this compression type, they can also specify JPEG compression options, which comprise all combinations of high, medium, or low Resolution and high, medium, or low Quality. The higher the Resolution and Quality settings, the greater the image quality, but the greater the disk space requirement.

For example, an image compressed with the High Resolution/High Quality option has the highest image quality and the highest disk space requirement. Conversely, an image compressed with the Low Resolution/Low Quality option has the lowest image quality and the lowest disk space requirement.

LZW — Can be used to compress TIFF image documents of any color type, except black-and-white. Should be used when users do not want the image to be altered by the compression process.

PackBits — Can be used to compress black-and-white TIFF image documents for any purpose.

Uncompressed — No compression options are set.

Example

Users of your application may want to change the compression type to save disk space.

Scenario

In an earlier scenario, Krystina sent Tom two True Color bitmap (BMP) image files of an automobile that was involved in a recent accident. Tom converted each BMP file to the TIFF file format and saved both of the images in a single TIFF image file. The resulting uncompressed file was quite large at 21 megabytes (MB); so, Tom elected to compress the file using the JPEG compression type with the following compression options:

- JPEG Compression: Medium
- JPEG Resolution: Medium

The resulting image file was reduced to just under 2 MB.

Resolution

Resolution determines the display quality of an image. Typically expressed as horizontal and vertical dots per inch (dpi), resolution describes the density of the dots that make up the image. The higher the resolution — or dots per inch — the better the display quality.

An important consideration when setting the resolution is file size. The greater the dots per inch, the greater the memory and storage requirements. For example, the file size of an image with a resolution of 200 x 200 dpi is four times greater than the file size of the same image at 100 x 100 dpi.

Another important consideration when setting the resolution is how the images are to be used:

Displayed on the screen — For images that are displayed on the screen, resolution need not be any greater than the display resolution of the monitor, typically 75 x 75 dpi to 100 x 100 dpi.

Faxed — For images that are faxed, resolution should conform to the international fax standard of 200 x 200 dpi.

Converted to text or printed — For images that are printed, resolution should be set to 300 x 300 dpi.

Your users want to use, or convert images to, the resolution that best satisfies their aesthetic, storage, and usage requirements.

Example

Users of your application may want to change the resolution of an image to save disk space.

Scenario

Assume Tom received a complimentary letter from a customer that he wants to post on the company's intranet page for all to see. Tom knows that he can use your application to scan the letter and convert it to HTML using the OCR functions you have provided.

Because you stated in your documentation that the OCR engine processes images with optimum efficiency when their resolution is 300 x 300 dpi, Tom scans the letter at that resolution.

After performing OCR on the image and uploading its HTML file to the Web server, Tom realizes that he wants to save the image on his computer — just in case he needs it later. Knowing that an image with a resolution of 300 x 300 dpi takes more storage space than one with a lower resolution, Tom uses the conversion functions in your program to convert the image to 200 x 200 dpi just prior to saving it.

Size

The size settings determine the dimensions of an image. Your users may want to change the size of the image and/or the unit of measure employed to suit their purposes.

Example

Users of your application may want to change the size of an image to accommodate an annotation.

Scenario

Assume Krystina scans a claim form and then wants to add a rubber stamp annotation to the bottom of it. The problem is: there's no room at the bottom of the image to accommodate the annotation. To make room for the annotation, Krystina converts the size of the image from 8 1/2 x 11 inches to 8 1/2 x 12 inches, thereby making room for the annotation.

Convert Image Project



The file name for the Convert Image project is `Convert.vbp`.

The Convert Image project shows how to provide image file type and page property conversion functions to your users.

The project consists of one form and the following controls:

- One Image Admin control
- One Image Edit control
- Three Command Button controls in a control array

It uses the following Imaging methods to provide the image conversion functions:

ShowFileDialog (Image Admin) — To enter the path and file name and select the new file type of the converted image.

SaveAs (Image Edit) — To save the image file with the new file type.

ShowPageProperties (Image Edit) — To change the color type, compression type, resolution, and/or size of the image page.

Save (Image Edit) — To save the image file after changing its color type, compression type, resolution, and/or size.

Note: Users can change the color type, compression type, resolution, and size on a page-by-page basis only.

Changing the File Type

Start the Convert Image project. The application begins by displaying an **Open** dialog box, which lets you select the image file you want to convert. After you select the image file, the application displays it.

To change the file type of the displayed image, click the **File Type** button. The `cmdConvert_Click()` event procedure fires and executes the code in Case 0 of the Select Case statement.



Similar to the Microsoft Common Dialog box, you can use the **Filter** property of the Image Admin control to populate the **Files of Type** list box with the file types you desire.

The procedure invokes the **ShowFileDialog** method of the Image Admin control, passing to it the following parameter values:

`SaveDlg` (literal 1) — To display a **Save As** dialog box

`frmConvertImage.hwnd` — To assign the parent window handle to the **Save As** dialog box

The **Save As** dialog box lets you specify the new path and file name and the new file type you want for the image. It assigns the new path and file name to the **Image** property of the Image Admin control, and it assigns the list box index value of the selected file type to the **FilterIndex** property of the Image Admin control.

The procedure continues by assigning the content of the **Image** property of the Image Admin control to the **Image** property of the Image Edit control. And it assigns the new file type — provided by the value of the **FilterIndex** property of the Image Admin control — to the `intFileType` local variable.

Next, the procedure invokes the **SaveAs** method of the Image Edit control, passing to it the new path and file name provided by the **Image** property and the new file type provided by `intFileType`.

The **SaveAs** method saves the image file using the new file name and file type.

```

Private Sub cmdConvert_Click(Index As Integer)
    Dim intFileType As Integer
    Dim iResponse As Integer

    Select Case Index

        Case 0 'File Type button

            'Set the Filter property to include the file types that can be
            'written to disk.
            ImgAdmin1.Filter = "TIFF Image file (*.tif)|*. tif|
            Bitmap Image file (*.bmp)|*. bmp|"

            'Set the FilterIndex property to the file type of the displayed image
            'if it can be written; otherwise to TIFF.
            If ImgAdmin1.FileType = FileTypeBMP Then
                ImgAdmin1.FilterIndex = 2
            Else
                ImgAdmin1.FilterIndex = 1
            End If

            'Invoke ShowFileDialog method.
            On Error GoTo CancelPressed_EH
            ImgAdmin1.ShowFileDialog SaveDlg, frmConvertImage.hWnd

            'Set Image property of the Image Edit control to the filename
            'returned by theOpen dialog box.
            ImgEdit1.Image = ImgAdmin1.Image

            'Set the iFileType variable to the file type returned by the
            'Open dialog box.
            If ImgAdmin1.FilterIndex = 2 Then
                iFileType = iFileTypeBMP
            Else
                iFileType = FileTypeTIFF
            End If

            'Invoke the SaveAs method using the new file name and file type
            ImgEdit1.SaveAs ImgEdit1.Image, intFileType

            .
            .
            .
        End Select

    CancelPressed_EH:

End Sub

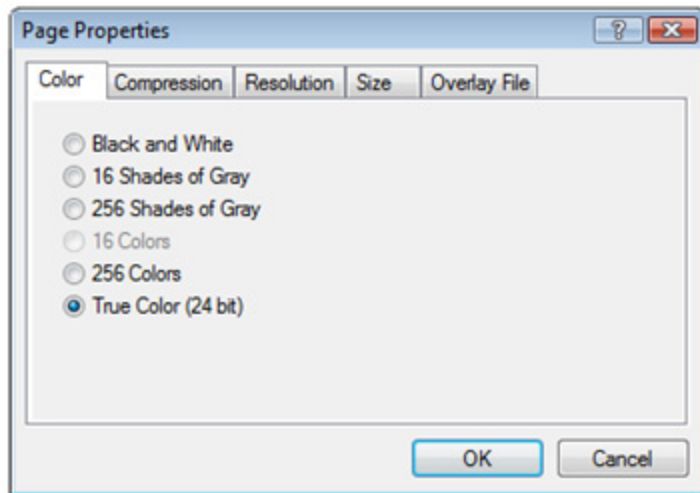
```

Changing the Color, Compression, Resolution, and Size

If necessary, start the Convert Image project and open an image file.

To change the color type, compression type, resolution, and/or size of the displayed image, click the **Page** button. The `cmdConvert_Click()` event procedure fires and executes the code in Case 1 of the Select Case statement.

The procedure invokes the **ShowPageProperties** method of the Image Edit control, which displays the **Page Properties** dialog box. As long as the `False` parameter is included in the call to the **ShowPageProperties** method, the dialog box can be used to specify a new color type, compression type, resolution, and/or size for the image page.



The **ShowPageProperties** method returns an integer that indicates whether the user has pressed the **OK** or **Cancel** button on the dialog box (the standard `vbOK` and `vbCancel` constants are available for use). The `cmdConvert_Click()` event procedure assigns this value to the `iResponse` local variable.

The procedure evaluates the value of the `iResponse` variable. If users click the **OK** button, it saves the altered image using the **Save** method of the Image Edit control. If users click the **Cancel** button, the event procedure exits without saving the image.

```
Private Sub cmdConvert_Click(Index As Integer)
    Dim intFileType As Integer
    Dim iResponse As Integer

    Select Case Index
    .
    .
    .
        Case 1 'Page button

            'Display the ShowPageProperties dialog box to let users convert the
            'image
            iResponse = ImgEdit1.ShowPageProperties(False)
            If iResponse = vbOK Then
                'User clicked OK on the dialog box so save the converted image
                ImgEdit1.Save
            ElseIf iResponse = vbCancel Then
                'User clicked Cancel on the dialog box so exit without saving
                Exit Sub
            End If
        .
        .
        .
    End Select

    CancelPressed_EH:

End Sub
```

Copying An Image

The Image Copy demonstration project shows how to add a Clipboard copy function to your image-enabled applications. Before walking through the demonstration project, read the following section, which explains the concept of using the Clipboard with image data.

Clipboard Functions Defined

You're probably familiar with using the Clipboard to copy, cut, and paste text data within your development environment or word processor. Using the Clipboard with image data is similar.

The Imaging ActiveX controls provide several properties, methods, and events that let you add Clipboard functions to your image-enabled applications. With them, your users can:

- Copy or cut image and/or annotation data to the Clipboard.
- Paste image or annotation data from the Clipboard onto an image displayed in the Image Edit control or into any application that supports the pasting of image data (for example, Microsoft Word, WordPad, Exchange, or Excel).

Depending on how you code your application, you can let your users copy or cut an entire image page, a selected portion of an image page, or selected annotations.

The following sections briefly describe the properties, methods, and events of the Image Edit control you'll find useful when adding Clipboard functions to your applications.

Clipboard Copy and Cut

ClipboardCopy method — Copies image or annotation data to the Clipboard.

ClipboardCut method — Copies image or annotation data to the Clipboard and then removes the data from the Image Edit control.

Clipboard Paste

IsClipboardDataAvailable method — Checks to see whether image or annotation data is present in the Clipboard. You can use this method to see whether data is available for pasting.

ClipboardPaste method — Pastes image or annotation data from the Clipboard onto an image in the Image Edit control.

CompletePaste method — Completes a Clipboard paste operation, making the pasted image or annotation data a part of the original image.

PasteCompleted event — Fires when the pasted image or annotation data is committed to a location on the target image.

PasteClip event — Fires when the pasted image or annotation data is too large to fit within the confines of the target image.

Image Selection

SelectionRectangle property — Sets whether a selection rectangle is drawn when an end user clicks the left mouse button and drags the mouse pointer over a displayed image. Can be used to select a portion of an image to copy or cut to the Clipboard.

DrawSelectionRect method — Draws a selection rectangle on an image programmatically.

SelectionRectDrawn event — Fires after a selection rectangle has been drawn by the end user or by the **DrawSelectionRect** method.

Note: A selection rectangle can be used to select a portion of an image with or without annotations; however, it cannot be used to select annotations alone.

Annotation Selection

AnnotationType property — When set to the Select Annotations annotation type, lets end users select one or more annotations for copying or cutting to the Clipboard (or for some other Imaging purpose).

Draw method — Draws an annotation. Can be used to select annotations programmatically by drawing a Select Annotations annotation type.

MarkSelect event — Fires after an end user or the program selects one or more annotations for copying or cutting to the Clipboard (or for some other Imaging purpose, such as **ZoomToSelection**, for example).

Note: The Select Annotations annotation type selects annotations exclusively. It does not select the underlying image data.

Example

Users of your application may want to copy an image page to the Clipboard so they can paste the image into a word processing document.

Scenario

Assume Susan is writing a follow-up letter to her insurance company about a reimbursement claim she has yet to receive. Before she sent the original receipt, she scanned it and saved it to disk. Now she wants to include a copy of the receipt in her follow-up letter.

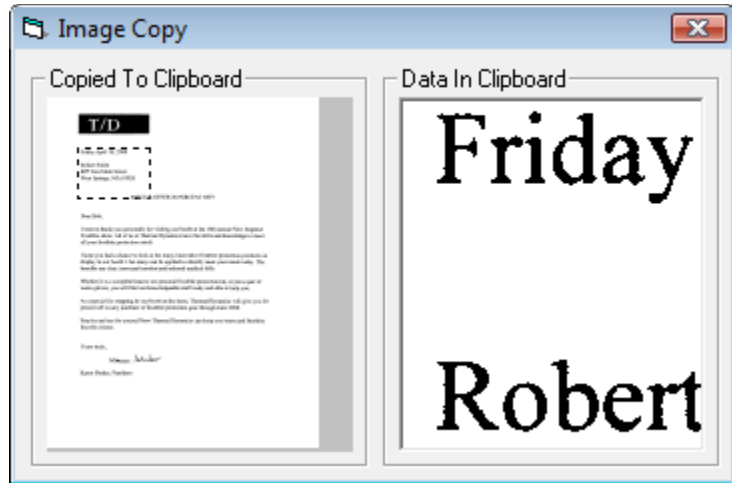
With the image of the receipt displayed in your application, she copies it to the Clipboard using the Clipboard functions you provided. Then, in Word, she pastes the image into her letter.

Copy Image Project



The file name for the Copy Image project is `Imgcopy.vbp`.

The Copy Image project demonstrates copying an entire image page to the Clipboard.



The project consists of one form and the following controls:

- One Image Admin control
- One Image Edit control
- One Picture Box control
- Two Frame controls

It uses the following methods in the Image Edit control to provide the image copy function:

Display — To display the image in the Image Edit control.

ClipboardCopy — To copy the image page to the Clipboard.

Copying the Image Page

Start the Image Copy project. The `Form_Load()` event procedure displays an **Open** dialog box to let you select the TIFF image file you want to copy.

After you select the image file, the procedure invokes the **Display** method to display the image in the Image Edit control (which is inside the **Copied To Clipboard** frame).

Next, the procedure invokes the **ClipboardCopy** method of the Image Edit control, passing to it the following parameters:

- The Left and Top coordinates of the image relative to the Image Edit control (16,26).
- The Width and Height of the image in pixels, as provided by the current values of the **ImageScaleWidth** “-100” and **ImageScaleHeight** “-150” properties of the Image Edit control.

Finally, the procedure obtains the current image content of the Clipboard using Visual Basic’s **GetData** method and displays it in the PictureBox control (which is inside the **Data In Clipboard** frame).

```
Private Sub Form_Load()  
    'Unlock the controls with the license file  
    'See VerifyLicense( ) in Globals  
    If Not VerifyLicense(ImgAdmin1.LicenseCheck(LicensePath)) Then End  
    .  
    .  
    .  
    'Check for valid TIFF file.  
    If ImgAdmin1.FileType <> 1 Then  
        GoTo File_EH  
    Else  
        'Use the FitTo method to make the displayed image fit into the  
        width  
        'of the Image Edit control.  
        ImgEdit1.FitTo FIT_TO_WIDTH  
  
        'Display the image.  
        ImgEdit1.Display  
        DoEvents  
  
        'Copy a portion the image onto the Clipboard.  
        ImgEdit1.ClipboardCopy 16, 26, ImgEdit1.ImageScaleWidth - 100, _  
            ImgEdit1.ImageScaleHeight - 150  
  
        'Get the image data from the the Clipboard and display it in  
        'the PictureBox control to show it was copied.  
        PicImage = Clipboard.GetData()  
    End If  
  
    Exit Sub  
  
File_EH:
```

Printing An Image

The Print Image demonstration project shows how to add image printing to your image-enabled applications. Before walking through the demonstration project, read the following section, which explains the concept of printing an image file.

Image Printing Defined

Printing an image file is very similar to printing a word processing document.

You can use the **ShowPrintDialog** method of the Image Admin control to present a **Print** dialog box to the end user. With it — and the **Print Options** dialog box that can be invoked from it — the end user can select:

- The printer to use.
- The pages to print.
- The number of copies to print, and whether to collate.
- The output format to use.
- The page orientation to use.
- Whether to print annotations.

The **PrintImage** method of the Image Edit control performs the actual print operation. Its parameters let you set the start page, end page, output format, annotation print preference, and printer to use.

Example

Users of your application may want to print an image file on a particular printer — and have *complete* control over the process of doing so.

Scenario

Assume Geoff is using your application to view an image of a technical drawing. As he views it, he annotates it with his comments.

After entering his last comment, Geoff realizes that he is supposed to meet Susan for lunch in just 15 minutes. He would like her to take a look at the technical drawing too — but he doesn't want her to see his annotated comments.

On the **File** menu of your application, Geoff clicks **Print**. Then, on the **Print** dialog box, he clicks the **Options** button.

On the **Print Options** dialog box, he unchecks the **Print displayed annotations and zones** check box and then clicks **OK** to return to the **Print** dialog box.

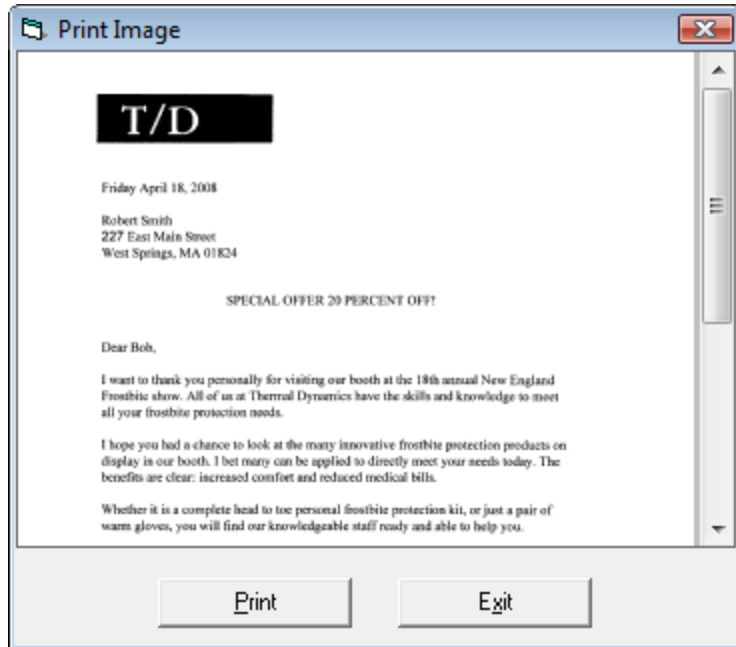
On the **Print** dialog box, Geoff selects the printer he wants to use and specifies the page range and number of copies to print. When he clicks **OK**, your application prints the drawing without Geoff's annotations.

Print Image Project



The file name for the Print Image project is `Print.vbp`.

The Print Image project demonstrates printing an image file.



The project consists of one form and the following controls:

- One Image Admin control
- One Image Edit control
- Two Command button controls in a control array

It uses the following Imaging methods to provide the print image function:

ShowPrintDialog (Image Admin) — To display a **Print** dialog box to the end user.

PrintImage (Image Edit) — To actually print the image.

Printing an Image File

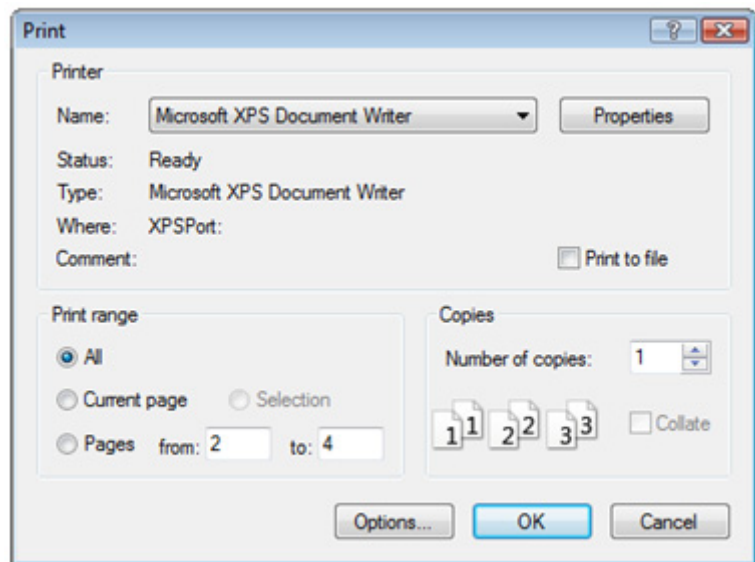
Start the Print Image project. The `Form_Load()` event procedure displays an **Open** dialog box to let you select the TIFF image file you want to print. After you select the image file, the procedure displays it in the Image Edit control.

To print the image, click the **Print** button. The `cmdPrint_Click()` event procedure fires and executes the code in Case 0 of the Select Case statement.

The procedure invokes the **ShowPrintDialog** method of the Image Admin control, passing to it the handle of the parent window.

Note: Even though passing the handle to the parent window is optional, for best results *always* include it when you invoke the **ShowPrintDialog** method.

The **ShowPrintDialog** method displays a **Print** dialog box, which lets you specify the print options mentioned earlier.





You can set the print-related properties to preferred values prior to invoking the **Print** dialog box. Doing so lets you preset dialog box fields to default settings

After you click the **OK** button on the **Print** dialog box, the Image Admin control sets several of its print-related properties to values that correspond to the selections made on the **Print** and **Print Options** dialog boxes. (Keep in mind that the Print Image project does not use all of these properties.)

Print-Related Properties Set By the Print and Print Options Dialog Boxes

Image Admin Property Set	Associated Field and Dialog Box	Value Property Contains
PrintAnnotations	Print displayed annotations and zones check box on the Print Options dialog box	True or False — Indicating whether to print annotations
PrintCollate	Collate check box on the Print dialog box	True or False — Indicating whether to collate image pages
PrintEndPage	Pages to text box on the Print dialog box	The ending page number in the range of pages to print
PrintNumCopies	Number of copies text box on the Print dialog box	The number of copies to print
PrintOrientation	Print orientation list box on the Print Options dialog box	The page orientation: 0 — Portrait 1 — Landscape 2 — Automatic

Print-Related Properties Set By the Print and Print Options Dialog Boxes (continued)

Image Admin Property Set	Associated Field and Dialog Box	Value Property Contains
PrintOutputFormat	Print format list box on the Print Options dialog box	The output format to use: 0 — Pixel to pixel 1 — Actual size 2 — Fit to page 3 — Best fit
PrintRangeOption	The following option buttons on the Print dialog box: <ul style="list-style-type: none"> ▪ All pages ▪ Current page ▪ Selection ▪ Pages 	Whether to print: 0 — All pages 1 — Range of pages 2 — Current page 3 — Selection
PrintStartPage	Pages from text box on the Print dialog box	The start page number in a range of pages to print
PrintToFile	Print to file check box on the Print dialog box	True or False — Indicating whether to print to a file

The procedure continues by evaluating the value of the **PrintRangeOption** property. It invokes the **PrintImage** method of the Image Edit control with the StartPage and EndPage parameter values that are appropriate for the **PrintRangeOption** value selected on the **Print** dialog box (as described in the following table).

StartPage and EndPage Parameter Values Passed

PrintRangeOption Constant (Literal)	StartPage Parameter	EndPage Parameter
PrintAll (0)	The value of the PrintStartPage property of Image Admin control	The value of the PrintEndPage property of Image Admin control
PrintRange (1)	The value of the PrintStartPage property of Image Admin control	The value of the PrintEndPage property of Image Admin control
PrintCurrent (2)	The value of the Page property of Image Edit control	The value of the Page property of Image Edit control

Each invocation of the **PrintImage** method also includes the OutputFormat and Annotation parameter values supplied by the **PrintOutputFormat** and **PrintAnnotations** properties of the Image Admin control.

Once invoked, the **PrintImage** method prints the image to the printer or file specified.

```

Private Sub cmdPrint_Click(Index As Integer)

    Select Case Index

        Case 0 'Print

            On Error GoTo Print_EH

            'Display the Print dialog box.
            ImgAdmin1.ShowPrintDialog frmPrintImage.hWnd

            'User pressed OK continue with print
            If ImgAdmin1.StatusCode = 0 Then

                'Check on which option the user selected then print
                'image using the Image Edit control.
                If ImgAdmin1.PrintRangeOption = PrintAll Then
                    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, _
                    ImgAdmin1.PrintEndPage, ImgAdmin1.PrintOutputFormat, _
                    ImgAdmin1.PrintAnnotations
                End If

                If ImgAdmin1.PrintRangeOption = PrintRange Then
                    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, _
                    ImgAdmin1.PrintEndPage, ImgAdmin1.PrintOutputFormat, _
                    ImgAdmin1.PrintAnnotations
                End If

                If ImgAdmin1.PrintRangeOption = PrintCurrent Then
                    ImgEdit1.PrintImage ImgEdit1.Page, ImgEdit1.Page, _
                    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
                End If

            End If

        Case 1 'Exit
            'End the program
            End

    End Select

    .
    .
    .
End Sub

```

Scanning an Image Using a Template

The Template Scanning demonstration project shows how to add template scanning to your image-enabled applications. Before walking through the demonstration project, read the following section, which explains the concept of template scanning.

Template Scanning Defined

When the Imaging software operates in the Template Scanning mode, it saves scanned images to files that are named and incremented automatically.

Each file name is based on a template, which consists of:

- A *path* to where the images are saved.
- A file name *prefix*, which is used to generate the file names.

The end user of the application usually provides the path and prefix.



The **ScanTo** property also has settings that permit scanning directly to a file (non-Template Scanning). When scanning to a file, consider using the **ShowScanNew** and the **ShowScanPage** methods of the Image Scan control to quickly add scanning functions to your applications.

Responding to input from your end user, you can place the Imaging software in Template Scanning mode by setting the **ScanTo** property of the Image Scan control to the appropriate value. You can then specify the template by setting the **Image** property of the Image Scan control to the path and prefix provided.

For example, if the user wants image files to be saved to the *c:\claims\automobile* path using file names prefixed with *auto*, enter the following string in the **Image** property of the Image Scan control:

```
imgScan1.Image = "c:\claims\automobile\auto"
```

Using this template, the Imaging software will save images to files named auto0001.tif, auto0002.tif, and so on in the *c:\claims\automobile* folder (TIFF file format assumed).

The file type as well as the setting of the **PageCount** and **MultiPage** properties of the Image Scan control determines the number of image files generated and the number of image pages saved per file.

When scanning images using the BMP file type, each image page is always saved to a separate file. For example:

```
c:\claims\automobile\auto0001.bmp
c:\claims\automobile\auto0002.bmp
c:\claims\automobile\auto0003.bmp
```

This occurs because the BMP file type does not support multiple image pages per file.

When scanning images using the TIFF file type — which supports multiple image pages per file — the setting of the **PageCount** and **MultiPage** properties of the Image Scan control determines the number of image files created and the number of pages in each image file. (Refer to the following table.)

PageCount and Multipage Property Influence

PageCount Setting	MultiPage Setting	Number of Image Files	Pages In Image File
0	True	1	All pages scanned
0	False	One file for each page scanned	1
X ^a	True	Total number of pages scanned divided by X	X
X	False	X	1

a. Any value greater than 0.

For example, if you set the **PageCount** property to 5 and the **MultiPage** property to True, and then you scan 20 pages, the Imaging software creates four image files with five pages in each one.

Example

Users of your application may want to use a scanner equipped with an automatic document feeder (ADF) to automatically scan multiple pages into one or more image files.

Scenario

Assume Gloria has five 10-page documents she wants to scan using her ADF-equipped scanner and your application.

She wants each 10-page document to be saved to its own TIFF image file in the `c:\employees` path, and she wants the file name of each document to be prefixed with the word *review*.

Because you provided a way for users to:

- Specify the desired path,
- Enter the file prefix,
- Select the file type,
- Set the **PageCount** property, and
- Set the **MultiPage** property,

Gloria was able to specify that:

- All documents are saved in the `c:\employees` folder.
- Each file name begins with the word *review*.
- Each document is scanned and saved as *TIFF*.
- Each document contains *10* pages.
- Each image file contains *multiple* image pages.

When Gloria begins scanning, your application:

- 1 Sets the **ScanTo** property of the Image Scan control to *3* (`DisplayAndUseFileTemplate`).

Note: A **ScanTo** property setting of *4* (`UseFileTemplateOnly`) also enables template scanning.

2 Creates the template by concatenating the path, a backslash, and the template prefix and by setting the **Image** property of the Image Admin control to the resulting string:

c:\employees\revi.

3 Sets the **FileType** property of the Image Admin control to *TIFF*

4 Sets the **PageCount** property of the Image Admin control to *10*.

5 Sets the **MultiPage** property of the Image Admin control to *True*.

6 Scans *50* pages and saves each set of *10* scanned pages to *5* individual TIFF image files — each one generated and incremented using the template specified:

c:\employees\revi0001.tif

c:\employees\revi0002.tif

c:\employees\revi0003.tif

c:\employees\revi0004.tif

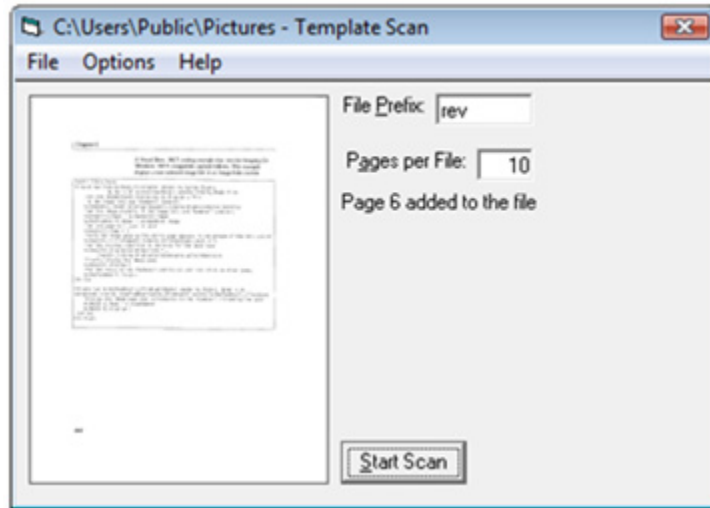
c:\employees\revi0005.tif

Template Scan Project



The file name for the Template Scan project is `Template.vbp`.

The Template Scan project demonstrates scanning to a template.



The project consists of the following forms and modules:

frmFileType — Enables the user to select the desired file type.

frmHelp — Presents a brief help message to the user.

frmMain — Lets the user enter the template prefix, specify the number of pages per file, commence scanning, and view the first page of the image file.

frmPaperSize — Enables the user to specify the desired paper size.

frmPath — Enables the user to specify the desired template path.

modMain — Contains global constant definitions and global variable declarations.

The scanning functions exist in the Main form (frmMain), which contains the following controls:

- One Image Scan control
- One Image Edit control
- Two text box and label controls
- One Command button control
- Three menus

The form uses the following methods of the Image Scan control to provide the scanning functions:

ShowSelectScanner — To select the scanner to use.

StartScan — To scan images.

Template Scanning

Start the Template Scan project. The Main form appears.

In the **File Prefix** text box, enter the prefix you want for the template. Then, in the **Pages per File** text box, enter the number of pages you want each image file to contain.

On the **File** menu:

- Click **File Type**. On the **File Type** form, click the file type you want and then click **OK**. Depending on the option button you clicked, the `cmdOK_Click()` event of the **File Type** form (frmFileType) sets the **FileType** property of the Image Scan control — contained on the **Main** form (frmMain) — to the file type specified.
- Click **Path**. On the **Folder** dialog box, specify the template path, which is where the image files are saved, and then click **OK**. The `cmdOK_Click()` event of the **Path** form (frmPath) sets the global variable, `gstrFolder`, to the path specified. (Later, the `cmdStartScan_Click()` event of the **Main** form uses the value of `gstrFolder` to set the **Image** property of the Image Scan control).

On the **Options** menu:

- Click **Paper Size**. On the **Paper Size** dialog box, click the paper size you expect to scan and then click **OK**. Depending on the option button you clicked, the `cmdOK_Click()` event of the **Paper Size** form (`frmPaperSize`) sets the global variable, `gsngAspect`, to the size specified. It then calls the `Form_Resize()` event of the **Main** form, which uses the value of `gsngAspect`, to resize the **Main** form and its controls (not shown).

```
Private Sub cmdOK_Click()

    'Set the File Type property of the Image Scan
    'control on frmMain according to File Type option
    'button on this form. In addition, set the Enabled
    'property of the label and text box controls accordingly.
    If optTiff.Value Then
        frmMain!ImgScan1.FileType = TIFF
        frmMain!lblPages.Enabled = True
        frmMain!txtPages.Enabled = True
    ElseIf optBMP.Value Then
        frmMain!ImgScan1.FileType = BMP_Bitmap
        frmMain!lblPages.Enabled = False
        'Set text box to 1 because BMP is a
        'single-page file format
        frmMain!txtPages.Text = 1
        frmMain!txtPages.Enabled = False
    End If

    Unload Me

End Sub
```

```
Private Sub cmdOK_Click()

    'Set the global path variable to the path specified
    gstrFolder = dirPath.Path

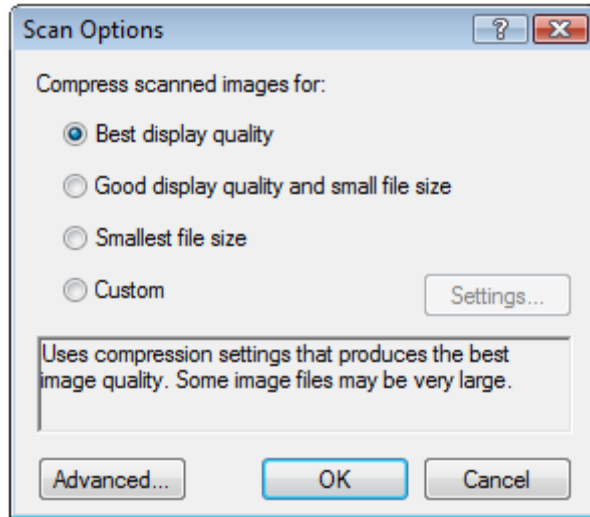
    'Set the Caption of the Main form to the path specified
    frmMain.Caption = gstrFolder + " - " + gstrCMainCaption

    'Unload the Path form
    Unload Me

End Sub
```

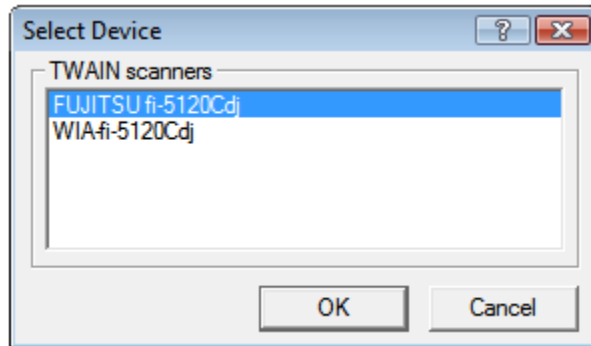
```
Private Sub cmdOK_Click()  
  
    'Set the global Aspect variable to either  
    'the size selected using an option button  
    'or the custom values entered  
    If optLetter Then  
        gsngAspect = 11 / 8.5  
    ElseIf optLegal Then  
        gsngAspect = 14 / 8.5  
    ElseIf optOther Then  
        If CSng(txtWidth.Text) < 1 Then  
            txtWidth.Text = 1  
            Beep  
        End If  
        If CSng(txtHeight.Text) < 1 Then  
            txtHeight.Text = 1  
            Beep  
        End If  
        msngTmpWidth = CSng(txtWidth.Text)  
        msngTmpHeight = CSng(txtHeight.Text)  
        gsngOtherWidth = CSng(txtWidth.Text)  
        gsngOtherHeight = CSng(txtHeight.Text)  
        gsngAspect = gsngOtherHeight / gsngOtherWidth  
    End If  
  
    'Hide this form, resize frmMain, and reset the  
    'Image Edit control to Best Fit  
    frmPaperSize.Hide  
    frmMain.Form_Resize  
  
End Sub
```

- Click **Scan Options** to apply compression. The `mnuCompressionOptions_Click()` event in the **Main** form invokes the **ShowScanPreferences** method of the Image Scan control. The method displays a **Scan Options** dialog box that lets you specify the compression you want.



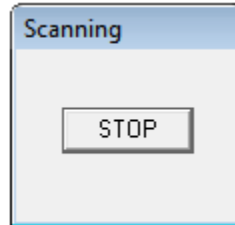
```
Private Sub mnuCompressionOptions_Click()  
    'Invoke the ShowScanPreferences method which  
    'displays the Scan Options dialog box  
    ImgScan1.ShowScanPreferences  
  
End Sub
```

- Click **Select Scanner** to select the scanner you want to use. The `mnuSelectScanner_Click()` event in the **Main** form invokes the **ShowSelectScanner** method of the Image Scan control. The method displays a **Select Device** dialog box that lets you select the scanner you want.



```
Private Sub mnuSelectScanner_Click()  
    'Invoke the ShowSelectScanner method which  
    'displays the Select Scanner dialog box  
    ImgScan1.ShowSelectScanner  
End Sub
```

- Click **Stop Button** to have the Imaging software display a **Stop** button while scanning.



The `mnuStopButton_Click()` event in the **Main** form sets the **StopScanBox** property to True or False (as appropriate) to either display, or not display, the **Stop** button. The **Stop** button enables you to abort a scanning operation in progress.

```
Private Sub mnuStopButton_Click()  
  
    'Set the StopScanBox property in accordance with  
    'the Checked status of the mnuStopButton menu  
    'selection  
    If mnuStopButton.Checked Then  
        mnuStopButton.Checked = False  
        ImgScan1.StopScanBox = False  
    Else  
        mnuStopButton.Checked = True  
        ImgScan1.StopScanBox = True  
    End If  
  
End Sub
```

To begin scanning, click the **Start Scan** button. The `cmdStartScan_Click()` event procedure in the **Main** form fires and executes its code.

The procedure sets several properties of the Image Scan control to enable template scanning:

DestImageControl — Set to the same value as the **ImageControl** property of the Image Edit control to permit image display while scanning.

Note: Setting the **DestImageControl** property to the value of the **ImageControl** property is essential whenever you want to display the image being scanned. It can be used for all types of scanning — not just template scanning.

ScanTo — Set to `DisplayAndUseFileTemplate` (literal 3) to select template scanning.

Image — Set to the template, which is a concatenated string containing:

- The path (as specified on the **Path** form and assigned to the `gstrFolder` global variable),
- A backslash (\), and
- The file name `prefix` (as specified in the **File Prefix** text box).

PageCount — Set to the value entered in the **Pages per File** text box to establish the number of pages per file.

MultiPage — Set to `True` to permit the scanning of multiple image pages.

Next, the procedure invokes the **StartScan** method of the Image Scan control, which scans multiple image pages:

- To the appropriate number of image files.
- Using auto-incremented path and file names that begin with the template specified.
- Containing the number of image pages specified.

```
Private Sub cmdStartScan_Click()  
    On Error GoTo Scan_EH  
    .  
    .  
    .  
    'Link the Image Scan and Image Edit controls to permit display  
    'while scanning  
    ImgScan1.DestImageControl = "ImgEdit1"  
  
    'Set the ScanTo property to enable template scanning  
    ImgScan1.ScanTo = DisplayAndUseFileTemplate  
  
    'Concatenate the path, backslash, and template prefix. Then assign the  
    'string to the Image property  
    ImgScan1.Image = gstrFolder + "\" + txtPrefix.Text  
  
    'Assign the Pages per File value to the PageCount property  
    ImgScan1.PageCount = txtPages  
  
    'Set the MultiPage property to enable multipage scanning  
    ImgScan1.MultiPage = True  
  
    'Commence scanning  
    ImgScan1.StartScan  
  
    Exit Sub  
  
Scan_EH:  
    'Display the error message  
    lblStatus.Caption = "ERROR - " + Err.Description  
    Beep  
  
End Sub
```


Managing an Image File Using Thumbnails

The Thumbnail Sorter demonstration project shows how to use the Image Thumbnail control to reorganize multipage image files. Before walking through the demonstration project, read the following section, which explains the basics of working with thumbnails.

Thumbnails Defined



The Image Thumbnail control has several properties that enable you to assign different fonts, colors, and styles to the captions, as well as to the control itself.

The Image Thumbnail control lets you view each page of an image file in miniature boxes called thumbnails. There is one thumbnail image for each page in the file.

Each thumbnail has a caption beneath it that indicates its page position within the image file and an annotation indicator if one or more annotation marks exist on the corresponding image page.

In addition to viewing image pages, the Image Thumbnail control — in conjunction with the Image Admin control — also lets you provide image file management functions to your end users. These functions enable them to:

- Select an image page for display, edit, manipulation, deletion, or some other Imaging function.
- Reorganize pages within the image file.
- Drag and drop image pages to and from other applications that support drag-and-drop.

Example

Users of your application may want to view image files as a series of thumbnail images. They may also want to manage image files using drag and drop.

Scenario

Assume Chris and his staff regularly review large fax files that contain mostly blueprint drawings.

Chris is concerned that in the middle of any of these files there might be a letter, or other piece of important information, that could go unnoticed when someone is scrolling through the file.

Using your application, Chris and his staff can quickly review each fax file by looking at its thumbnail images. When they find pages that have important information, they can drag and drop them into an another Image Thumbnail control, whose `ThumbDrop()` event contains code that routes the image pages to the appropriate personnel.

Thumbnail Sorter Project

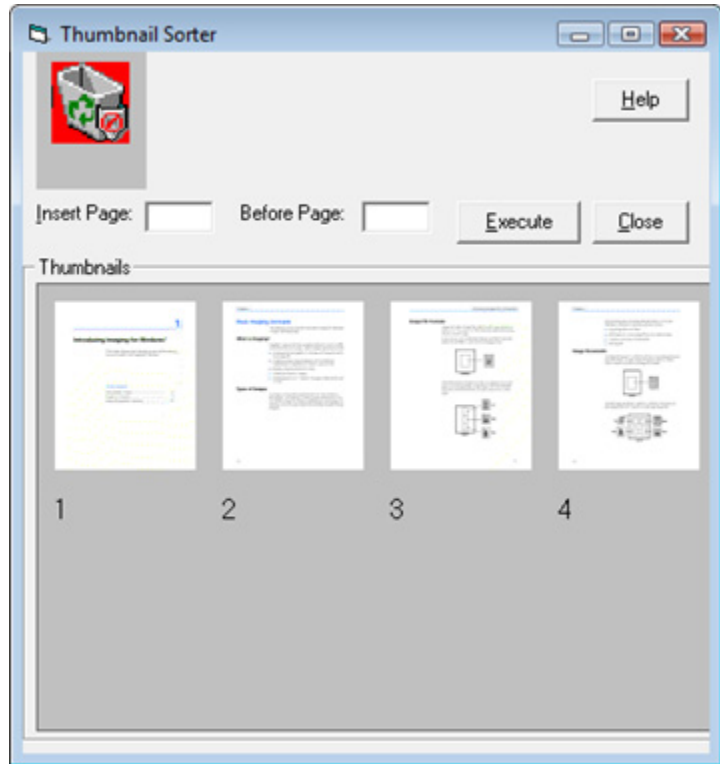


The file name for the Thumbnail Sorter project is `Thumbnail.vbp`.

The Thumbnail Sorter project demonstrates using thumbnails to manage an image file. Specifically, it enables you to:

- Reorganize images pages within the file.
- Drag and drop image pages from Explorer.

- Delete image pages from the file by dragging and dropping them into another Image Thumbnail control.



The project consists of one form and the following controls:

- One Image Admin control
- Two Image Thumbnail controls
- Three Command button controls
- Two Text Box box controls
- Two Label controls
- One Frame control

The project uses the following Imaging methods to provide the thumbnail file management functions:

DisplayThumbs (Image Thumbnail control) — To display the pages of an image file as a series of thumbnail images.

Insert (Image Admin control) — To insert one or more selected pages into the current image file.

InsertThumbs (Image Thumbnail control) — To refresh the Image Thumbnail control with the inserted pages.

DeletePages (Image Admin control) — To delete one or more selected pages from the current image file.

DeleteThumbs (Image Thumbnail control) — To refresh the Image Thumbnail control without the deleted pages.

Note: The Thumbnail Sorter project also uses the **Drag** (extender) method, which is provided by the frame that contains the Image Thumbnail control. When invoked, it begins a drag operation.

Sorting an Image File (Using Specified Page Numbers)

Start the Thumbnail Sorter project. The `Form_Load()` event procedure displays an **Open** dialog box to let you select the TIFF image file you want to work with. Be sure to select a multipage image file.

After you select the image file, the procedure:

- Sets the **Image** property of the Image Thumbnail control to the complete path and file name you selected (as supplied by the **Image** property of the Image Admin control).
- Invokes the **DisplayThumbs** method of the Image Thumbnail control to display each page of the file as a thumbnail in the Image Thumbnail control located inside the **Thumbnails** frame.



The **EnableDragDrop** property lets you set the desired drag-and-drop behavior.

- Sets the **AutoSelect** property of the Image Thumbnail control to **True** to have the control handle all thumbnail selections made using the mouse.
- Sets the **EnableDragDrop** property of the Image Thumbnail control to a literal value of 15, which is a bit-wise combination of the settings described in the following table.

Note: The **EnableDragDrop** property value determines the drag-and-drop behavior of the Thumbnail Sorter application, which is described in the sections entitled “Sorting an Image File (Using Drag and Drop)” and “Deleting Image Pages (Using Drag and Drop)” .

Combined EnableDragDrop Property Settings

Literal Value	Setting Description
1	Enable drag using left mouse button
2	Enable drag using right mouse button
4	Enable drop into
8	Enable dropping of image files
Result = 15	DropFilesDropDragLeftRight

- Gets the number of thumbnail images displayed from the **ThumbCount** property of the Image Thumbnail control.
- Sets up another Image Thumbnail control to serve as a trash bin by setting its **ThumbWidth** and **ThumbHeight** properties to the desired dimensions, and its **Image** property to a bitmap of a trash bin.

```
Private Sub Form_Load()  
    Dim strPathName As String  
    Dim strTemp As String  
    Dim strRightChar As String, strLeftChar As String  
    Dim intRPosition As Integer  
.  
.  
.  
    'Invoke the ShowFileDialog box  
    ImgAdmin1.ShowFileDialog OpenDlg  
  
End If  
  
'Display an error message if the image file is not TIFF  
If ImgAdmin1.FileType <> 1 Then  
    GoTo File_EH  
Else  
    'Set the Image property of the Thumbnail control.  
    ImgThumbnail.Image = ImgAdmin1.Image  
  
    'Get the path of the application  
    strPathName = App.Path  
  
    'Display a thumbnail for each image page in the file  
    ImgThumbnail.DisplayThumbs  
  
    'Set AutoSelect to true to enable drag and drop, and  
    'EnableDragDrop to DropFilesDropDragLeftRight (literal 15)  
    ImgThumbnail.AutoSelect = True  
    ImgThumbnail.EnableDragDrop = 15  
  
    'Get the thumbnail page count  
    mlngThumbCount = ImgThumbnail.ThumbCount  
  
    'Set up an Image Thumbnail control as a trash bin  
    'to provide a way to delete pages  
    ImgThumbnailTrash.ThumbWidth = 50  
    ImgThumbnailTrash.ThumbHeight = 50  
    ImgThumbnailTrash.Image = strPathName + "\trashbin.bmp"  
.  
.  
.  
End Sub
```

In the **Insert Page** text box, enter the number of the page you want to insert before another page in the file. Then in the **Before Page** text box, enter the number of the page you want the Insert page to appear before.

For example, to insert page 4 before page 2, enter 4 in the **Insert Page** text box and 2 in the **Before Page** text box.

Click the **Execute** button. The `cmdExecute_Click()` event procedure fires and executes its code.

The procedure performs the following actions:

- Gets the complete path and file name of the current image file from the **Image** property of the Image Thumbnail control and assigns it to the `strName` local variable.
- Gets the Insert page number from the **Insert Page** text box and assigns it to the `lngInsertPage` local variable.
- Gets the Insert Before page from the **Before Page** text box and assigns it to the `lngInsertBeforePage` local variable.
- Invokes the **Insert** method of the Image Admin control, passing to it the:
 - Path and file name of the current image file (from `strName`).
 - Insert Page number (from `lngInsertPage`).
 - Insert Before page number (from `lngInsertBeforePage`).
 - 1 (which specifies the number of pages to insert).

The **Insert** method inserts a *copy* of the Insert page before the Insert Before page in the current image file.

Note: You must call the **Insert** method of the Image Admin control before calling the **InsertThumbs** method of the Image Thumbnail control.

- Invokes the **InsertThumbs** method of the Image Thumbnail control, passing to it the:
 - Insert Before page number (from `lngInsertBeforePage`).
 - 1 (which specifies the number of pages to insert).

The **InsertThumbs** method refreshes the control. Were you to set a breakpoint after invoking this method, you'd see *two* copies of the Insert page in the Image Thumbnail control.

- Determines the page number of the unwanted, “leftover” copy of the image page (from the value of `lngInsertPage`) so it can be deleted from the file.

If the unwanted page is after the Insert Before page, the procedure increments `lngInsertPage` by one to delete the correct page.

- Invokes the **DeletePages** method of the Image Admin control, passing to it the:
 - Number of the page to delete (from `lngInsertPage`).
 - 1 (which specifies the number of pages to delete).

The **DeletePages** method deletes the unwanted, “leftover” copy of the image page from the file.

Note: You must call the **DeletePages** method of the Image Admin control before calling the **DeleteThumbs** method of the Image Thumbnail control.

- Invokes the **DeleteThumbs** method of the Image Thumbnail control, passing to it the:
 - Number of the page to delete (from `lngInsertPage`).
 - 1 (which specifies the number of pages to delete).

The **DeleteThumbs** method refreshes the control without the unwanted page.


```

Private Sub cmdExecute_Click()
    Dim strName As String
    Dim lngInsertPage As Long
    Dim lngInsertBeforePage As Long

    'Get the path and file name of the displayed image
    strName = ImgThumbnail.Image

    'Get the Insert page and the Insert Before page
    lngInsertPage = CLng(txtInsertPage.Text)
    lngInsertBeforePage = CLng(txtInsertBeforePage.Text)

    'Check to see if the Insert page is to be inserted
    'before itself. If it is, abort processing.
    If lngInsertPage = lngInsertBeforePage Or lngInsertPage =
    lngInsertBeforePage - 1 Then
        Exit Sub
    End If

    'Place the Insert page before the Insert Before page in the current
    'image file
    ImgAdmin1.Insert strName, lngInsertPage, lngInsertBeforePage, 1

    'Refresh the Image Thumbnail control to display the reordered
    'image file
    ImgThumbnail.InsertThumbs lngInsertBeforePage, 1

    'Delete the "leftover" page
    If strName = ImgThumbnail.Image Then
        'If the InsertPage number is greater than the Insert Before
        'number, increment the lngInsertPage variable by 1 to set
        'the appropriate page for deletion
        If lngInsertPage > lngInsertBeforePage Then
            lngInsertPage = lngInsertPage + 1
        End If
        'Delete the "leftover" page from the image file and the Image
        'Thumbnail control
        ImgAdmin1.DeletePages lngInsertPage, 1
        ImgThumbnail.DeleteThumbs lngInsertPage, 1
    End If
End Sub

```

Sorting an Image File (Using Drag and Drop)

If necessary, start the Thumbnail Sorter project and select a multipage image file.



You can select multiple thumbnails by holding down the **Shift** or **Ctrl** keys.

Point to the thumbnail of an image page you want to insert before another page in the file. Then hold down the left mouse button. The **MouseDown()** event procedure of the Image Thumbnail control fires and invokes the **Drag** (extender) method of Visual Basic, which starts the Drag operation.

```
Private Sub ImgThumbnail_MouseDown(ByVal Button As Integer, _
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single, _
    ByVal ThumbNumber As Long)

    'Invoke the Drag operation
    ImgThumbnail.Drag

End Sub
```

Note: As an alternative, you can point to an image file in Explorer and then hold down the left mouse button.

Next, drag the thumbnail (or image file) to the desired position and release the left mouse button. The **ThumbDrop()** event procedure of the Image Thumbnail control fires and performs the following actions:

- Gets the complete source path and file name from the **ThumbDropNames** property of the Image Thumbnail control for each image page being inserted and assigns it to the `strName` local variable.

If you're dragging and dropping pages within the current image file, the **ThumbDropNames** property returns the path and file name of the displayed image file.

If you're dragging and dropping pages from Explorer, the **ThumbDropNames** property returns the path and file name of the source image file.

- Gets the Insert page number from the **ThumbDropPages** property of the Image Thumbnail control for each image page being inserted, and assigns it to the `lngInsertPage` local variable.
- Invokes the **Insert** method of the Image Admin control, passing to it the:
 - Path and file name of the current image file (from `strName`).
 - Insert Page number (from `lngInsertPage`).
 - Insert Before page number (from the `InsertBefore` argument of the **ThumbDrop()** event).
 - 1 (which specifies the number of pages to insert).

The **Insert** method inserts a *copy* of the Insert page before the Insert Before page in the current image file.

Note: You must call the **Insert** method of the Image Admin control before calling the **InsertThumbs** method of the Image Thumbnail control.

- Invokes the **InsertThumbs** method of the Image Thumbnail control, passing to it the:
 - Insert Before page number (from `InsertBefore`).
 - 1 (which specifies the number of pages to insert).

The **InsertThumbs** method refreshes the control. As in the previous section, were you to set a breakpoint after invoking this method, you'd see *two* copies of the Insert page in the Image Thumbnail control.

- Determines the page number of the unwanted, “leftover” copy of the image page (from the value of `lngInsertPage`) so it can be deleted from the file.

If the unwanted page is after the Insert Before page, the procedure increments `lngInsertPage` by one to delete the correct page.

- Invokes the **DeletePages** method of the Image Admin control, passing to it the:
 - Number of the page to delete (from `lngInsertPage`).

– 1 (which specifies the number of pages to delete).

The **DeletePages** method deletes the unwanted, “leftover” copy of the image page from the file.

Note: You must call the **DeletePages** method of the Image Admin control before calling the **DeleteThumbs** method of the Image Thumbnail control.

- Invokes the **DeleteThumbs** method of the Image Thumbnail control, passing to it the:
 - Number of the page to delete (from `lngInsertPage`).
 - 1 (which specifies the number of pages to delete).

The **DeleteThumbs** method refreshes the control without the unwanted page.

```

Private Sub ImgThumbnail_ThumbDrop(ByVal InsertBefore As Long, _
ByVal DropCount As Long, ByVal Shift As Integer)
    Dim X As Integer
    Dim strName As String
    Dim lngInsertPage As Long

    'Move all selected pages or insert from Explorer
    For X = 0 To DropCount - 1
        'Get the path and name of the file containing the Insert page
        strName = ImgThumbnail.ThumbDropNames(X)
        'Get the Insert page
        lngInsertPage = ImgThumbnail.ThumbDropPages(X)

        'Check to see if the Insert page is to be inserted
        'before itself. If it is, abort processing.
        If strName = ImgThumbnail.Image Then
            If lngInsertPage = InsertBefore Or lngInsertPage = InsertBefore - 1 Then
                Exit Sub
            End If
        End If

        'Place the Insert page before the Insert Before page in the image file
        ImgAdmin1.Insert strName, lngInsertPage, InsertBefore, 1

        'Refresh the Image Thumbnail control to display the reordered image file
        ImgThumbnail.InsertThumbs InsertBefore, 1

        'Delete the "leftover" page
        If strName = ImgThumbnail.Image Then
            'If the InsertPage number is greater than the Insert Before
            'number, increment the lngInsertPage variable by 1 to set
            'the appropriate page for deletion
            If lngInsertPage > InsertBefore Then
                lngInsertPage = lngInsertPage + 1
            End If
            'Delete the "leftover" page from the image file and the Image
            'Thumbnail control
            ImgAdmin1.DeletePages lngInsertPage, 1
            ImgThumbnail.DeleteThumbs lngInsertPage, 1
        End If
    Next X
End Sub

```

Deleting Image Pages (Using Drag and Drop)

If necessary, start the Thumbnail Sorter project and select a multipage image file.



You can select multiple thumbnails by holding down the **Shift** or **Ctrl** keys.

Point to the thumbnail of an image page you want to delete. Then hold down the left mouse button. The **MouseDown()** event procedure of the Image Thumbnail control fires and invokes the **Drag** (extender) method of Visual Basic, which starts the Drag operation.

```
Private Sub ImgThumbnail_MouseDown(ByVal Button As Integer, _
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single, _
    ByVal ThumbNumber As Long)

    'Invoke the Drag operation
    ImgThumbnail.Drag

End Sub
```

Next, drag the thumbnail to the Image Thumbnail control resembling a trash bin and release the left mouse button. The **ThumbDrop()** event procedure of the Image Thumbnail (trash bin) control fires and performs the following actions for each selected page in the For...Next loop:

Note: The DropCount argument of the **ThumbDrop()** event provides the array-maximum value. Subtracting 1 from DropCount is required because the For...Next loop is 0-relative.

- Gets the complete path and file name from the **ThumbDropNames** property of the Image Thumbnail control for each image page being deleted and assigns it to the strName local variable.
- Gets the Delete page number from the **ThumbDropPages** property of the Image Thumbnail control for each image page

being deleted and assigns it to the `lngDeletePage` local variable.

- Makes sure that only pages from the current image file are deleted by comparing the value returned by the **Image** property of the source Image Thumbnail control to the value returned by the **ThumbDropNames** property of the destination Image Thumbnail (trash bin) control (from `strName`). If the path and file name values don't match, the event procedure is exited.

This action prevents using the Thumbnail Sorter to delete image files from Windows Explorer.

- Invokes the **DeletePages** method of the Image Admin control, passing to it the:
 - Number of the page to delete (from `lngDeletePage-X`).
 - 1 (which specifies the number of pages to delete).

The **DeletePages** method deletes the image page from the file.

Note: You must call the **DeletePages** method of the Image Admin control before calling the **DeleteThumbs** method of the Image Thumbnail control.

- Invokes the **DeleteThumbs** method of the source Image Thumbnail control, passing to it the:
 - Number of the page to delete (from `lngDeletePage-X`).
 - 1 (which specifies the number of pages to delete).

The **DeleteThumbs** method refreshes the control without the deleted page.

Subtracting the Value of X

When calling the **DeletePages** and **DeleteThumbs** methods, the procedure subtracts the value of counter X from the page number to ensure that the proper page is deleted from the file and the control. The necessity of this action becomes apparent when you have selected more than one page for deletion.

For example, assume you delete pages two and three from a five-page file.

The first-page-to-delete value passed to both methods is 2 [(lngDeletePage = 2) minus (X = 0)]. The **DeletePages** method deletes page 2 from the file and the **DeleteThumbs** method refreshes the control without the deleted page.

The second-page-to-delete value passed to both methods is also 2 [(lngDeletePage = 3) minus (X = 1)]. Subtracting 1 from the lngDeletePage value of 3 compensates for the page that was previously deleted.

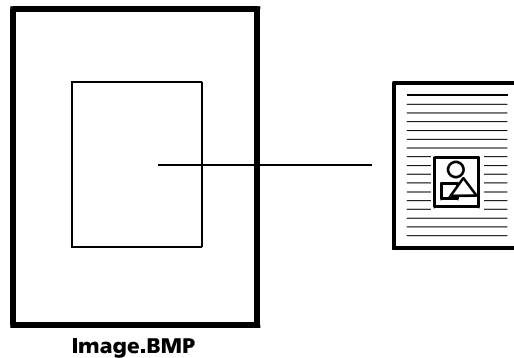

```
Private Sub ImgThumbnailTrash_ThumbDrop(ByVal InsertBefore As Long, _  
    ByVal DropCount As Long, ByVal Shift As Integer)  
    Dim X As Integer  
    Dim strName As String  
    Dim lngDeletePage As Long  
  
    'Delete the selected pages  
    For X = 0 To DropCount - 1  
        'Get the path and name of the file containing the Delete page  
        strName = ImgThumbnailTrash.ThumbDropNames(X)  
        'Get the Delete page  
        lngDeletePage = ImgThumbnailTrash.ThumbDropPages(X)  
  
        'If page dropped from Explorer, exit the procedure  
        If strName <> ImgThumbnail.Image Then  
            Exit Sub  
        Else  
            'Delete the selected page from the image file and  
            'the source Image Thumbnail control  
            ImgAdmin1.DeletePages lngDeletePage - X, 1  
            ImgThumbnail.DeleteThumbs lngDeletePage - X, 1  
        End If  
    Next X  
  
End Sub
```

Unloading a Multipage Image File

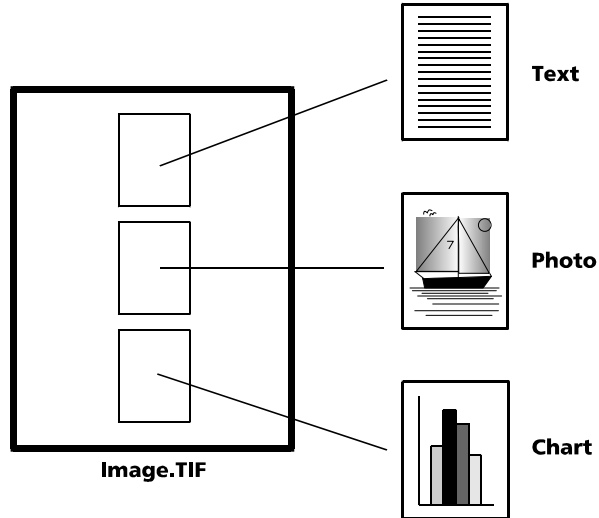
The Unload demonstration project shows how to save the individual pages of a multipage image file as a series of single-page files, in effect, extracting — or *unloading* — the pages of a multipage image file. Before walking through the demonstration project, read the following sections, which explain the concept of multipage image files and describe the properties and methods that enable you to add page management and manipulation functions to your applications.

Multipage Image Files Defined

Some image file types, such as BMP, can contain only one image page per file.



Other file types, such as TIFF, can contain several image pages per file.



The following table describes the multipage support provided by the file types that the Imaging software can read and read/write.

Multipage Support By File Type

File Type	Supports Multiple Pages?	Read/Write Status
BMP	No	Read/Write
TIFF	Yes	Read/Write
JPG-JFIF	No	Read/Write
PDF	Yes	Read Only
GIF	No	Read Only
PCX	No	Read Only

Multipage Support By File Type (continued)

File Type	Supports Multiple Pages?	Read/Write Status
WIFF	Yes	Read Only
XIF	Yes	Read Only
DCX	Yes	Read Only

Page-Related Properties and Methods

The Imaging ActiveX controls have several properties and methods that enable you to work with multipage image files — either to create them or perform some sort of Imaging action on their individual pages.

Note: Because each thumbnail image represents an image page, the Image Thumbnail control — in conjunction with the Image Admin control — is particularly useful for working with image pages. (Refer to “Managing an Image File Using Thumbnails” to see a sample application.)

The following list briefly describes the properties and methods you’ll find useful when working with multipage image documents.

Image Admin

PageCount property — Returns the number of pages in an image file.

PageNumber property — Returns or sets a page number in an image file.

PageType property — Returns the page type — also known as the color type or data type — of a specified page.

Append method — Adds one or more pages to an image file.

DeletePages method — Deletes a range of pages from an image file.

Replace method — Replaces one or more pages in an image file.

Image Edit

Page property — Returns or sets the page number of an image file where an imaging action was or will be performed.

PageCount property — Returns the number of pages in the displayed image file.

PageType property — Returns the page type of the image specified in the **Image** property and the page specified in the **Page** property of the Image Edit control.

ConvertPageType method — Converts a displayed image page to a specific page type.

SavePage method — Saves the displayed image page to the path and file name specified.

ShowPageProperties method — Displays the **Page Properties** dialog box, which enables users to view or modify the properties of the displayed image page (color, compression, resolution, and size).

Image Scan

MultiPage property — Determines whether multiple image pages will be scanned to an image file.

Page property — Returns or sets the starting page for a scanning session.

PageCount property — Returns or sets the number of pages scanned per image file. Works in conjunction with the **MultiPage** property to determine how many pages are scanned to how many files. (Refer to the “PageCount and Multipage Property Influence” table earlier in this chapter for more information.)

PageOption property — Returns or sets whether a page will be appended, inserted, or overwritten during a scanning session.

ShowScanPage method — Displays the **Scan Page** dialog box, which lets users scan a page into an image file.

Image Thumbnail

FirstSelectedThumb property — Returns the page number of the first selected thumbnail.

LastSelectedThumb property — Returns the page number of the last selected thumbnail.

SelectedThumbCount property — Returns the number of thumbnails currently selected.

ThumbCount property — Returns the total number of pages in the current image file.

ThumbDropNames property — Returns the file names of image pages dropped on the Thumbnail control.

ThumbDropPages property — Returns a list of pages for the file names dropped on the Thumbnail control.

ThumbSelected property — Returns or sets the selection status of a specified thumbnail.

DeleteThumbs method — Refreshes the Thumbnail control without the image pages that have been deleted from an image file.

GetManualThumbPage method — Returns the page within the image file corresponding to the thumbnail array subscript.

InsertThumbs method — Refreshes the Thumbnail control with the image pages that have been inserted into the current image file.

Example

Users of your application may want to *unload* a multipage image file when the individual pages have no logical relationship to each other or when the individual pages need to be routed to different people.

Scenario

As described in an earlier scenario, Chris regularly reviews large multipage fax files as part of his job.

Sometimes Chris encounters pages that contain memos, letters, or forms that are of interest to different people but have no relationship with the remainder of the file.

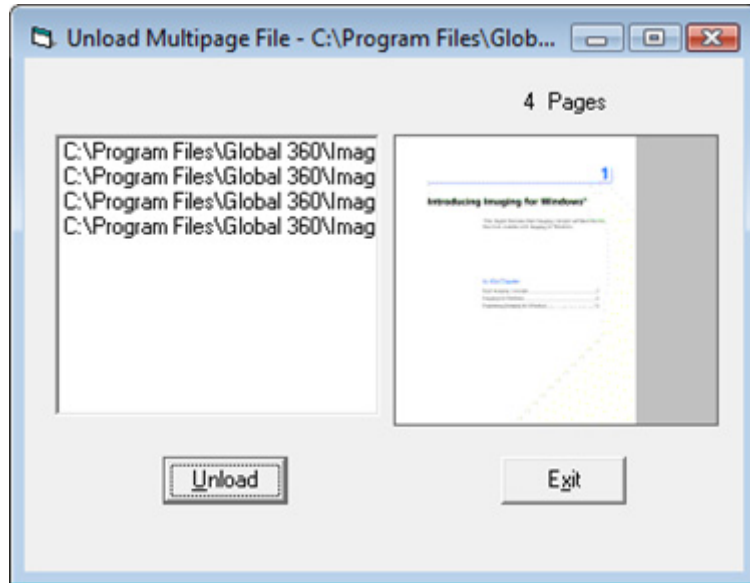
Using your application, Chris can quickly unload these pages and save them to disk as individual image files — or send them to the appropriate people via e-mail if you have included e-mail support in your application.

Unload Project



The file name for the Unload project is `UnLoad.vbp`.

The Unload project demonstrates unloading the pages of a multipage image file and saving them to disk as a series of single-page image files.



The project consists of one form and the following controls:

- One Image Admin control
- One Image Edit control
- Two Command button controls
- One list box control

And it uses the following methods of the Image Admin control to provide the unload functions:

Append — To create the individual image files.

Delete — To remove existing image files.

VerifyImage — To see whether an image file already exists.

Unloading an Image File

Start the Unload project. The `Form_Activate()` event procedure displays an **Open** dialog box to let you select the TIFF image file you want to unload. Be sure to select a multipage image file.

After you select the image file, the **Unload** form appears.

To unload the image file, click the **Unload** button. The `cmdUnload_Click()` event procedure fires and executes its code.

The procedure begins by setting two local string variables to the path and file name of the multipage image file you selected (the source file). The `strSourceFile` variable will retain the path and file name throughout the procedure. The `strPrefix` variable will be gradually parsed until it contains only the first three letters of the source image file name — becoming a *prefix* of the destination file names.

The procedure continues by obtaining the number of image pages in the source file from the **PageCount** property of the Image Edit control, assigning it to the `intPageCount` local variable. It also obtains the current path, assigning it to the `strCurrentPath` local variable. It is into this path that the individual destination file names will be written.

Next, the procedure parses the `strPrefix` variable until it consists of a three-character prefix:

- First, it gets rid of the path information.
- Second, it gets rid of all of the remaining characters — including the file extension, which is assigned to the `strExt` variable for later use.

Before building the individual (Unloaded) file names, the procedure appends a slash to the current path variable, `strCurrentPath`.

With all of these preliminaries out of the way, the procedure is now ready to build the Unloaded file names.

A For...Next statement executes for each page in the image file — starting at 1 and continuing until it exceeds the maximum number of pages in the image file (from `intPageCount`).

For each iteration of the For...Next loop, the procedure concatenates the following six items to build an Unloaded path and file name, which it assigns to the `strUnloadedFileName` local variable (the examples assume a *source* path and file name of `c:\Images\Faxes.tif`):

- 1 Current path and slash (from `strCurrentPath`).

Example: `c:\Images\`

- 2 Three-character prefix (from `strPrefix`).

Example: `c:\Images\Fax`

- 3 “Pag” (a hard-coded preface to the page number).

Example: `c:\Images\FaxPag.`

- 4 Current page number (from `strPageNum`, which is the counter value converted to a string).

Example: `c:\Images\FaxPag1`

- 5 “.” (the dot before the file extension).

Example: `c:\Images\FaxPag1.`

- 6 File extension (from `strExt`).

Example: `c:\Images\FaxPag1.tif`

Now that the complete path and file name exist for the unloaded page, the procedure assigns the Unloaded path and file name (from `strUnloadedFileName`) to the **Image** property of the Image Admin control.

Next, the procedure invokes the **VerifyImage** method of the Image Admin control to see if the Unloaded file already exists in the current path. If it does, the procedure invokes the **Delete** method of the Image Admin control to remove it.



The procedure invokes the **VerifyImage** method with a parameter value of 0 (Verify Existence). Other parameter values let you check an image file's read/write attributes.

The procedure invokes the **Append** method of the Image Admin control to add the source page to the unloaded file, creating it automatically.

Finally, the procedure adds the path and file name of the first **Unloaded** image to the list box control and executes the next iteration in the For...Next loop (the second Unloaded image).

```
Private Sub cmdUnload_Click()
    Dim intPageCount As Integer, intPageNumber As Integer
    Dim intSlashPos As Integer, intDotPos As Integer
    Dim intVerifyExistence As Integer
    Dim strPrefix As String, strCurrentPath As String
    Dim strSourceFile As String, strUnloadedFileName As String
    Dim strExt As String, strPageNum As String

    'Get the path and file name of the source file
    strSourceFile = ImgEdit1.Image
    strPrefix = ImgEdit1.Image

    'Get the number of pages in the source file
    intPageCount = ImgEdit1.PageCount

    'Get the current path
    strCurrentPath = CurDir

    'Establish an appropriate prefix for the Unloaded file names
    intSlashPos = 7
    'First, eliminate the characters to the left of the slashes
    Do While intSlashPos <> 0
        intSlashPos = InStr(1, strPrefix, "\", 1)
        strPrefix = Right(strPrefix, Len(strPrefix) - intSlashPos)
    Loop
    'Second, eliminate the characters to the right of the dot
    intDotPos = InStr(1, strPrefix, ".", 1)
    strExt = Right(strPrefix, Len(strPrefix) - intDotPos)
    If intDotPos > 4 Then
        strPrefix = Left(strPrefix, 3)
    Else
        strPrefix = Left(strPrefix, intDotPos - 1)
    End If

    'Append a slash to the current path specification
    intSlashPos = InStr(1, strCurrentPath, "\", 1)
    If intSlashPos <> Len(strCurrentPath) Then
        strCurrentPath = strCurrentPath + "\"
    End If

```

(Continued next page)

```
'Build and populate one image file for each of the pages in the source image
'file
  For intPageNumber = 1 To intPageCount

    'Get the current page number
    strPageNum = Str$(intPageNumber)
    'Build the Unloaded path and file name
    strUnloadedFileName = strCurrentPath + strPrefix + "Pag" + strPageNum + _
      "." + strExt
    'Assign the Unloaded path and file name to the Image property
    ImgAdmin1.Image = strUnloadedFileName
    intVerifyExistence = 0
    'If the Unloaded file exists, delete it
    If ImgAdmin1.VerifyImage(intVerifyExistence) = True Then
      ImgAdmin1.Delete strUnloadedFileName
    End If

    'Populate the Unloaded image file
    ImgAdmin1.Append strSourceFile, intPageNumber, 1

    'Add the new image path and file name to the list box control
    lstFiles.AddItem strUnloadedFileName

  Next intPageNumber

End Sub
```

Developing Client/Server Applications

This chapter explains how to use the Imaging ActiveX controls to develop applications that can access and interact with Global 360 Imaging Server (1.x) and Execute360 servers. It also explains how to add zoom and annotation functions to your applications. Even if you are not going to include Imaging server access in your applications, you'll find the sections on these functions useful.

In This Chapter

Imaging Server Concepts	84
Imaging 1.x Server Programming Considerations	88
Execute360 Server Programming Considerations.....	109
Demonstration Project	113

Imaging Server Concepts

Using the Imaging ActiveX controls, you can develop applications that can access and interact with both Global 360 Imaging Server (1.x) and Execute360 servers. Using the controls, you can enable your users to:

- Read and display image files and server documents from Imaging 1.x servers.
- Write image files and documents to Imaging 1.x servers.
- Read and display documents from Execute360 servers.
- Save 1.x image files and server documents and Execute360 documents to local and network drives.

Note: To use the Imaging ActiveX controls with Imaging Server (1.x), you and your users must install and configure Imaging Server Access when installing Imaging for Windows®.

Before getting into the specifics, read the following sections that:

- List the file types supported by each Imaging server.
- Describe the standard, server-related dialog boxes available with the Imaging ActiveX controls.
- Explain the difference between image files and server documents.
- Describe how your program can interact with each Imaging server.

Note: Because a wide-ranging discussion of each Imaging server is beyond the scope of this chapter, you should also review the documentation that came with the server that you and your users use.

File Type Support

1.x and Execute360 servers support the file types described in the following table.

Imaging Server File Types

Server	File Types Supported
Imaging 1.x	TIFF 6.0
	BMP
	DCX (read only)
	GIF (read only)
	JPG-JFIF
	PCX (read only)
	WIFF (read only)
	XIF (read only)
Execute360	TIFF 6.0 with Execute360

Standard Dialog Boxes

Several methods in the Image Admin control display server-related dialog boxes to your end users. These dialog boxes enable your users to:

- Log on to the desired server — either Imaging 1.x or Execute360.
- Set Imaging 1.x server options.
- Browse for Imaging 1.x file and document volumes.
- Browse the Imaging 1.x file and document volumes for files and documents to open.

- Query Imaging 1.x document volumes by document name, location, creation date, modification date, or keywords to locate documents to open.
- Query Execute360 servers by document name or field values to locate documents to open.
- Save 1.x image files, 1.x documents, and Execute360 documents to an Imaging 1.x file volume, an Imaging 1.x document volume, or a local or network drive, respectively.

Image Files and Server Documents

An *Image file* is simply a binary file that contains one or more images. You or your users can use operating system commands to save, organize, copy, rename, delete, and otherwise operate on files of this type.

A *Server document* is a collection of related images, logically organized as pages within the document. The Imaging software stores the actual images as image files; a server document simply contains references — or pointers — to the location and name of each associated image page.

Interacting with Imaging 1.x Servers

When interacting with Imaging 1.x servers, your program can read images from — and write images to — image file or document volumes.

Image File Volume

Otherwise known as a *file repository*, a file volume is the location where the actual image files are stored.

Your program can process image files in the file repository directly, using a file specification. File specifications consist of a server name, volume name, one or more directory names (optional), and a file name in the following format:

```
Image://server/file volume:/directory/filename.tif
```


Document Volume

Formerly known as a *document manager database*, a document volume is where server documents are stored. A document volume does not contain the actual image files, only references to the files in the file repository.

Server documents are stored using a familiar hierarchy. Documents are stored within a Folder; Folders are stored within a Drawer; and Drawers are stored within a Cabinet; using the following format:

Image://server/doc volume:\cabinet\drawer\folder\doc

Interacting with Execute360 Servers

When interacting with Execute360 servers, your program can read documents from those servers.

Users cannot edit these documents unless they save them on their local or network drives first or on an Imaging 1.x server if one is available to them.

Execute360 servers store documents using a flat syntax that consists of a prefix and document name; for example,
Imagex://claim234.

Imaging 1.x Server Programming Considerations

Several properties and methods in the Imaging ActiveX controls let you provide Imaging 1.x server access functions to your end users. Specifically, they permit your users to:

- Log on to the server.
- Set server options.
- Browse for Imaging 1.x file and document volumes.
- Browse the Imaging 1.x file and document volumes for files and documents to open.
- Query Imaging 1.x document volumes by document name, location, creation date, modification date, or keywords to locate documents to open.
- Save 1.x image files and server documents to Imaging 1.x file or document volumes or to local or network drives.

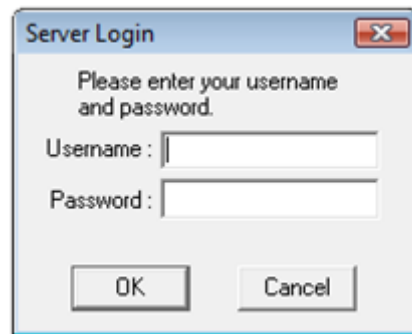
The following sections describe each function in detail by pointing out the properties and methods you can use.

Logging On To the Server

Before users can interact with an Imaging 1.x server, they must log on to it. The Image Admin control provides a **LoginToServer** method that lets you programmatically log your users on to an Imaging 1.x server.

Note: The **LoginToServer** method does not support non-unified logins — it is intended to be used in UNIX environments.

In your call to the **LoginToServer** method, you can optionally display the standard **Server Login** dialog box, which permits users to enter their user name and password and then click **OK** to log on to the server.



The **Username** text box can accommodate up to 20 alphanumeric characters; the **Password** text box can contain up to 36 alphanumeric characters.



The Image Admin control also provides a **LogOffServer** method that lets you programmatically log your users off an Imaging 1.x server.

You can also bypass the dialog box and simply pass the user name and password as parameters to the **LoginToServer** method, most likely in response to a user completing and closing a logon dialog box of your own design.

If a user is not logged on and attempts to access the server, the Imaging software displays the standard **Server Login** dialog box automatically — thereby prompting the user to log on to the server.

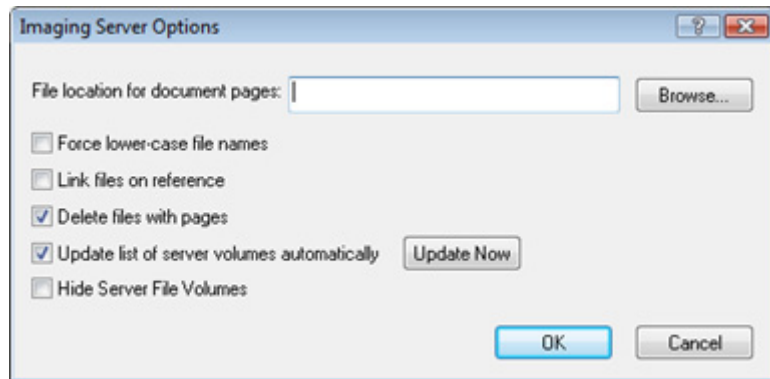
Setting Imaging 1.x Server Options

Setting server options is a task that users should perform after they install your program and whenever necessary thereafter.

You, or your users, should specify:

- The path to the 1.x file repository, where your program stores new or copied image files that comprise server documents.
- Whether to force lower-case path and file names.
- Whether to link server documents to the original image files or to copies of the original files.
- Whether to delete image files that were linked to deleted server document pages.

The Image Admin control provides a **Show1xServerOptDlg** method that lets you display the standard **Imaging Server Options** dialog box to your end users. The dialog box lets them set server options.



When users click **OK** on the **Imaging Server Options** dialog box, the Image Admin control sets several of its server-related properties to values that correspond to the selections made on the dialog box. The following table lists the properties set.

Imaging 1.x Server-Related Properties Set By the Server Options Dialog Box

Image Admin Property Set	Associated Field on Dialog Box	Value Property Contains
FileStgLoc1x	File location for document pages text box	The path to the file repository where your program stores <i>new</i> or <i>copied</i> image files that comprise server documents.
ForceLowerCase1x	Force lower-case file names check box	True or False — Indicating whether paths and file names are converted to lower case before being passed to the Imaging 1.x server.

Imaging 1.x Server-Related Properties Set By the Server Options Dialog Box (continued)

Image Admin Property Set	Associated Field on Dialog Box	Value Property Contains
ForceFileLinking1x	Link files on reference check box	True or False — Indicating whether pages being added to an Imaging 1.x document <i>from an existing 1.x image file</i> are linked (True) or copied (False).
ForceFileDeletion1x	Delete files with pages check box	True or False — Indicating whether the file referenced by an Imaging 1x server document page is deleted when the document page is deleted.

You can set these properties in advance to present default dialog box settings to your users.

You can also bypass the standard dialog box and set these properties within your code — most likely in response to a user completing and closing a server options dialog box of your own design.

Note: Update List of Server Volumes

There are no properties or methods associated with the **Update list of server volumes automatically** check box or the **Update Now** button on the **Server Options** dialog box. Both invoke a facility that updates the list of server volumes in the current domain. Users can access this list by clicking the **Browse** button on the dialog box. They use the list to locate and enter the file location for document pages.

Hide Server File Volumes

Likewise, there are no properties or methods associated with the **Hide server volume files** check box. This check box specifies whether Imaging 1.x server file volumes are displayed or hidden on the **Open** dialog box, which you can display by invoking the **ShowFileDialog** method of the Image Admin control.

Hiding server file volumes is useful when users only want to browse server document volumes.

Invoke the **Show1xServerOptDlg** method if you want to provide the **Update List of Server Volumes** function and the **Hide Server File Volumes** function to your end users.

The following sections explain each server option setting and related property in detail.

File Location for Document Pages (FileStgLoc1x Property)

When your program saves a server document page, it *may* have to create a file that contains the actual image. Such files must reside somewhere on an Imaging 1.x server and only your users know exactly where that should be. As a result, you need to let your users enter the location where your program stores these files.

The **FileStgLoc1x** property of the Image Admin control contains the location where your program stores the new image files. You can set this property yourself in response to user input or you can have users set it via the **Imaging Server Options** dialog box.

Depending on how you code your program, the following methods use this location to save images: the **SaveAs** and **SavePage** methods of the Image Edit control; the **SaveAs** method of the Image Thumbnail control; and the **Append**, **Replace**, and **Insert** methods of the Image Admin control. Each of these methods creates a new image file with a unique file name in the following situations:

- When a page from a local or redirected file is being inserted, appended, or saved to an Imaging 1.x document.
- When the **ForceFileLinking1x** property is set to **False** and a page from an Imaging 1.x image file or document is being inserted, appended, replaced, or saved in another Imaging 1.x document.

Force Lower-Case File Names (ForceLowerCase1x Property)

Older Imaging 1.x 16-bit clients have traditionally converted path and file names to lower-case when communicating with the Imaging 1.x file system.

To maintain backward compatibility with the 16-bit clients and to ensure that Imaging for Windows clients can access Imaging 1.x file repositories created by the 16-bit clients, Imaging for Windows includes a facility that performs this conversion.

The **ForceLowerCase1x** property of the Image Admin control determines whether your program converts path and file names to lower-case when communicating with the Imaging 1.x file system. You can set this property yourself in response to user input or you can have users set it via the **Imaging Server Options** dialog box.

When the **ForceLowerCase1x** property is set to **True**, case conversion occurs. When set to **False**, no case conversion occurs.

Note: You or your users should set the **ForceLowerCase1x** property to **False** unless you or they have specific compatibility problems with older document volumes (document manager databases).

Link Files On Reference (ForceFileLinking1x Property)

An Imaging 1.x document is basically a list of references to the files that contain the actual images.

Certain operations, such as that performed by the **Append** method of the Image Admin control, enable your program to add pages to an Imaging 1.x document.

There are two different ways to add such a page:

- Linking the document directly to the actual image file, as long as the file resides in a 1.x file volume (*link on reference*).
- Copying the page to another image file and then linking the document to the copied image file (*copy on reference*).

The **ForceFileLinking1x** property of the Image Admin control determines how pages being added to an Imaging 1.x document are referenced. You can set this property yourself in response to user input or you can have users set it via the **Imaging Server Options** dialog box.

Then, to add pages to an Imaging 1.x document, use one of the following: the **Append**, **Replace**, or **Insert** method of the Image Admin control; the **SaveAs** or **SavePages** method of the Image Edit control; or the **SaveAs** method of the Image Thumbnail control.

When you do, the value of the **ForceFileLinking1x** property *and* where the source image file resides determine the behavior of the Imaging software, as follows:

- When the **ForceFileLinking1x** property is set to **True** and the source image file resides on an Imaging 1.x server, the Imaging software links the document to the source image file where the file resides on the server.
- When the **ForceFileLinking1x** property is set to **False** and the source image file resides on an Imaging 1.x server, the Imaging software copies the image file to the location specified within the **FileStgLoc1x** property. Then it links the document to the copied file.
- Regardless of the setting of the **ForceFileLinking1x** property, when the source image file resides outside of an Imaging 1.x server (for example, on a local or redirected drive), the Imaging software copies the image file to the location specified within the **FileStgLoc1x** property. Then it links the document to the copied file.

- When the **SaveAs** or **SavePage** method is used to save changes to an existing 1.x document page, the Imaging software replaces the original file page regardless of the setting of the **ForceFileLinking1x** property (unless the original file is write-protected).

Delete Files With Pages (ForceFileDeletion1x Property)

When users delete a page from an Imaging 1.x document, they may also want to delete the linked image file.

The **ForceFileDeletion1x** property of the Image Admin control determines whether the Imaging software deletes the file referenced by a deleted Imaging 1.x server document page. You can set this property yourself in response to user input or you can have users set it via the **Imaging Server Options** dialog box. The **Delete** or **DeletePages** method of the Image Admin control performs the actual deletion.

When set to **True**, the Imaging software deletes the source file linked to an Imaging 1.x document when a user deletes the corresponding document page.

When set to **False**, the Imaging software does not delete the linked source file when a user deletes the corresponding document page.

Referential Integrity Behavior

The Imaging software ignores the **True** setting of the **ForceFileLinking1x** property when two or more pages of a server document are linked to the same image file and not all of these pages have been deleted from the document. The Imaging software does not delete the linked image file because one or more pages in the document are still linked to the file.

Example

Assume a server document named `Claims` contains 5 pages and that pages 1 and 4 in the document are linked to the `autoclaims.tif` image file.

Assume you delete page 1 of `Claims`.

Even though the **ForceFileDeletion1x** property is set to **True**, the Imaging software does not delete `autoclaims.tif`. Why? Because the file contains the image that is still linked to page 4 of the `Claims` document.

Note: Keep in mind that the referential integrity behavior of the **ForceFileDeletion1x** property provides protection against inadvertently deleting image files that are referenced *within the same document* only. It provides no protection against deleting image files that are referenced *within two or more documents*.

Browsing for Volumes or Image Files and Server Documents

The Image Admin control provides two ways to browse Imaging 1.x servers. Using them, you can let your users:

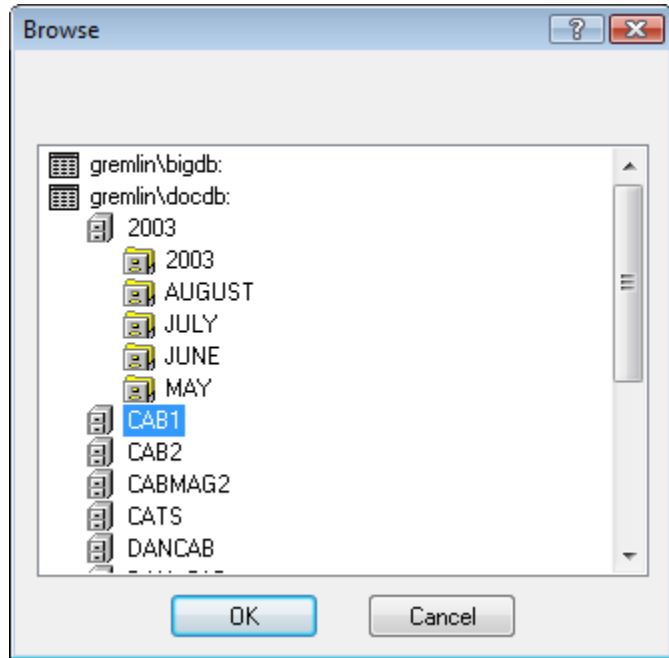
- Browse the server for image file or server document *volumes*, or
- Browse the server for *image files* and/or *server documents*.

Browsing for Volumes

The **Browse1x** method of the Image Admin control displays a dialog box that lets your users browse the Imaging 1.x server for the file and/or document *volumes* they want.

Use this method whenever you want users to select a desired volume for a particular purpose in your program. A good example of using this method exists on the **Server Options** dialog box, which is described in the previous section. After users click the **Browse** button, they can use the **Browse 1.x** dialog box to

navigate to and then select the location where they want new image files to be stored.



In your call to the **Browse1x** method, you can specify:

- Whether to browse:

File volumes — By passing the `BrowseFiles` constant or a literal value of 0.

Document volumes — By passing the `BrowseDocuments` constant or a literal value of 1.

Both file and document volumes — By passing the `BrowseBoth` constant or a literal value of 2.

- A string that becomes the title bar caption of the dialog box (for example, “Browse” in the preceding figure).
- Another string that instructs the user to browse the server.
- The handle to the parent window (optional).



Keep in mind that the **Get VolumeType** method of the Image Admin control lets you determine whether a specified Imaging 1.x volume is a *file* volume or a *document* volume.

After users make their selection and click **OK**, the Imaging software writes the path selected to the **Browse1xReturnedPath** property of the Image Admin control.

The Imaging software also writes the *type of volume* selected to the **Browse1xReturnedPathType** property, which can contain one of the following integer values:

- 0 — To indicate that users selected a file volume.
- 1 — To indicate that users selected a document volume.

Browsing for Files and Documents

The **ShowFileDialog** method of the Image Admin control displays an **Open** dialog box that lets users browse for and then select (open) the image file they want to display.

When you install Imaging 1.x Server Access, the Imaging software alters the **Open** dialog box slightly by adding a **Look for** list box that lets users select for browsing and display:

Desktop Files — The image files stored on local or network drives.

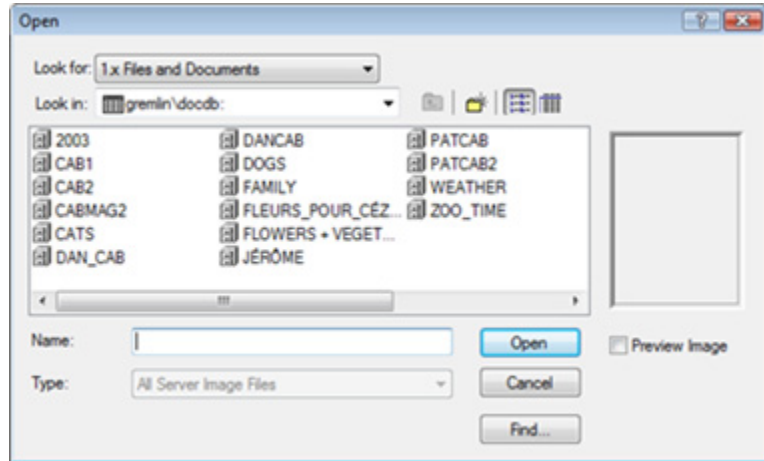
1.x Files and Documents — The image files or server documents stored on Imaging 1.x servers.

Invoke the **ShowFileDialog** method, therefore, whenever you want users to browse for and then select an Imaging 1.x file or server document they want to display.

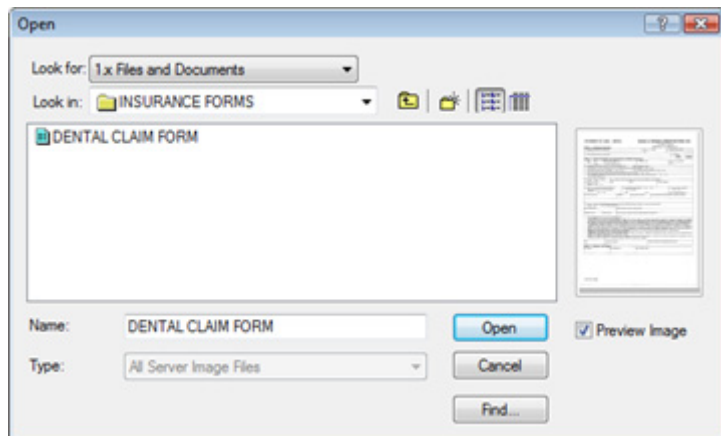


The **Look for** list box also contains an Execute360 Documents selection. When clicked, a message appears instructing users to click the **Find** button to perform a query.

When users select 1.x Files and Documents from the **Look for** list box, they can use the **Look in** list box to browse the file and document volumes in the current domain, as illustrated in the following figure.



Once users select the desired directory or folder, the area below the **Look in** box lists the files or documents contained within it, as illustrated in the following figure.



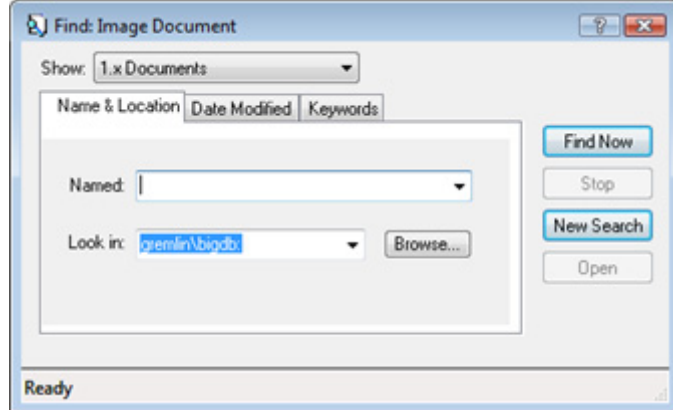
When users select the desired file or document and then click **Open**, the Imaging software writes the path and name of the file or document to the **Image** property of the Image Admin control.

Use the **Image** property and the **Display** method of the Image Edit control and/or the **Image** property and the **DisplayThumbs** method of the Image Thumbnail control to display the image file or document.

Refer to Chapter 2 for more information about displaying images and thumbnails as well as for more information about the **ShowFileDialog** method.

Querying Imaging 1.x Documents

The **ShowFindDialog** method of the Image Admin control displays a dialog box that enables users to query Imaging 1.x document volumes for the documents they want.



Prior to your call to the **ShowFindDialog** method, you can set the **Init1xFindDir** property of the Image Admin control to the name of the document volume you want to initially display in the



Clicking the **Find** button on the **Open** dialog box (described in the previous section) also displays the **Find Image Document** dialog box.

Look in list box (as a default). You can also include a cabinet; cabinet and drawer; or cabinet, drawer, and folder.

In your call to the **ShowFindDialog** method, you can specify the handle to the parent window. Doing so sets the state of the dialog box to *application-modal*; not doing so sets the state of the dialog box to *modeless*.

Finding Imaging 1.x Documents



You can browse for 1.x documents. On the **File** menu, click **Open**. In the **Look for** list, click **1.x Files and Documents**.

You can search for 1.x Documents by using the options provided on the Name & Location, Date Modified, and Keywords tabs.

The following procedures describe how to use these options.

Note: The search uses selection criteria specified on all tabs in the Find Image Document dialog box, including tabs that are not displayed.

Name and Location Tab

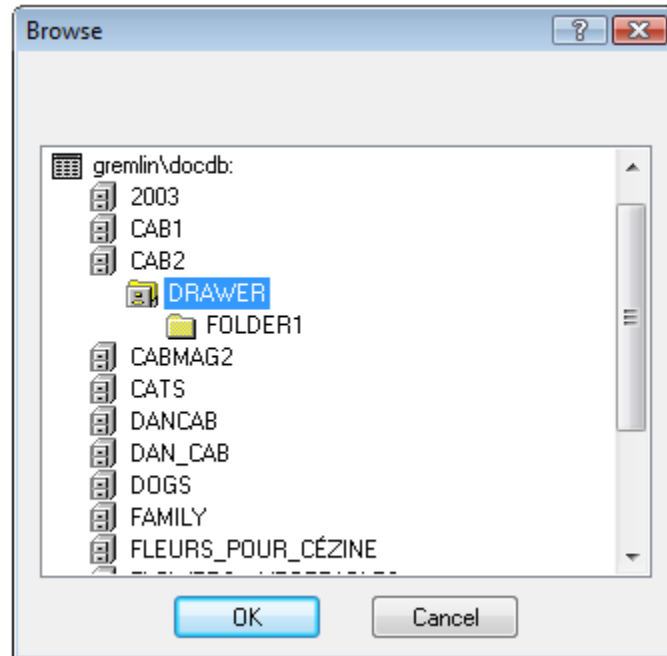
The Name and Location tab contains server and database information.



Use wildcard characters to identify a range of documents, or when you cannot remember the exact document name. An asterisk (*) represents a group of characters. A question mark (?) represents one character in a specified position.

- 1 In the **Named** text box, type a full or partial document name or leave the text box blank to retrieve all documents.
- 2 From the **Look in** list, select a database or type a database name string. The string must include the name of the server and the database (refer to sample entry in the **Find: Image** dialog box).

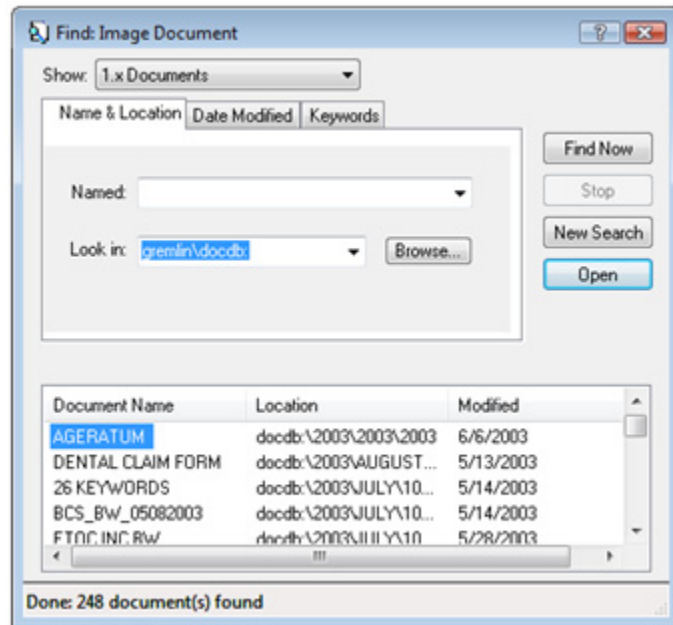
You can also click **Browse** to open the **Browse** dialog box, and then navigate through a database hierarchy.





If the search is taking too long, click **Stop** to end the operation. The documents found up to this point are displayed.

- 3 Click **Find Now** to initiate the search. A list of documents matching your search criteria is displayed in the lower portion of the **Find Image Document** dialog box.

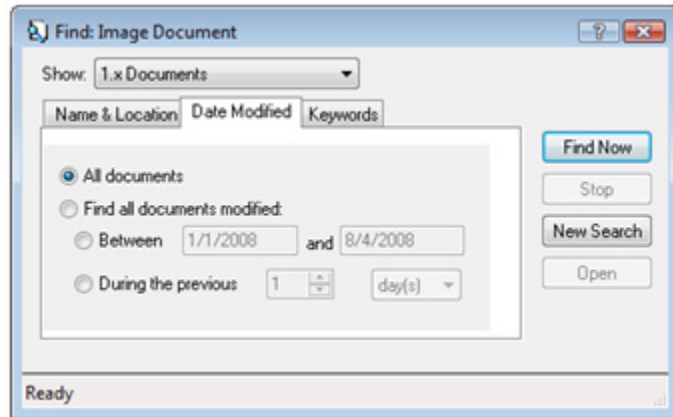


To sort a column of data in ascending or descending order, alternately click the column heading.

Note: Although your search returns a list of all documents that meet the specified criteria, you cannot open them unless you have access rights.

Date Modified Tab

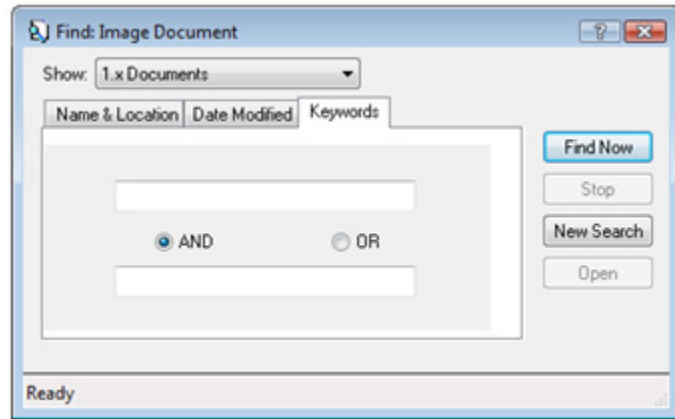
The **Date Modified** tab enables you to specify date modified criteria for your document search.



- 1 Click **All Documents** and then click **Find Now**.
or:
Click **Find All Documents Modified** and then follow the remaining steps.
- 2 Click **Between**, and then specify a range of dates in which a document was modified or click **During the Previous**, and specify the number of days or months to search back.
- 3 Click **Find Now** to initiate the search. A list of documents matching your search criteria is returned.

Keywords Tab

The Keywords tab enables you to search for documents using keywords that were previously associated with the document.



- 1 Type a single keyword into either text box. You can use wildcard characters or type a single keyword into each text box and then choose AND or OR.
 - AND returns documents with *both* specified keywords.
 - OR returns documents with *either* specified keyword.
- 2 Click **Find Now** to initiate the search. A list of documents matching your search criteria is returned.



The **ImgQuery** and **ImgQueryEnd** methods of the Image Admin control enable you to query a 1.x document volume programmatically. The “Demonstration Project” section of this chapter describes and demonstrates these methods.

Opening the Selected Document

After users make their selection and click **Open**, the Imaging software writes the path and name of the document to the **Image** property of the Image Admin control.

Note: The Imaging software does not alter the value of the **Init1xFindDir** property.

Use the **Image** property and the **Display** method of the Image Edit control and/or the **Image** property and the **DisplayThumbs** method of the Image Thumbnail control to display the Imaging 1.x document.

Refer to Chapter 2 for more information about displaying images and thumbnails.

Saving 1.x Image Files and Documents

The **ShowFileDialog** method of the Image Admin control also displays a dialog box that lets users enter the path and name for the image files they want to save.

When you install Imaging 1.x Server Access, the Imaging software alters the **SaveAs** dialog box slightly by adding a **Look for** list box that lets users select for saving:

Desktop Files — Image files on their computers.

1.x Files and Documents — Image files or server documents on Imaging 1.x servers.

When users select 1.x Files and Documents, they can use the **Look in** list box to browse file and document volumes in the current domain.

Once users select the desired directory or folder, the area below the **Look in** box lists the files or documents contained within it.

When users select or enter the name of the file or document and then click **Save**, the Imaging software writes the path and name of the file or document to the **Image** property of the Image Admin control.

Use the **Image** property and the **SaveAs** or **SavePage** method of the Image Edit control to save the image file or server document.

Execute360 Server Programming Considerations

Several properties and methods in the Imaging ActiveX controls let you provide Execute360 server access functions to your end users. Specifically, they permit your users to:

- Log on to the server.
- Query Execute360 servers by document name as well as by Class and Index field values.

The following sections describe each function in detail by pointing out the properties and methods you can use.

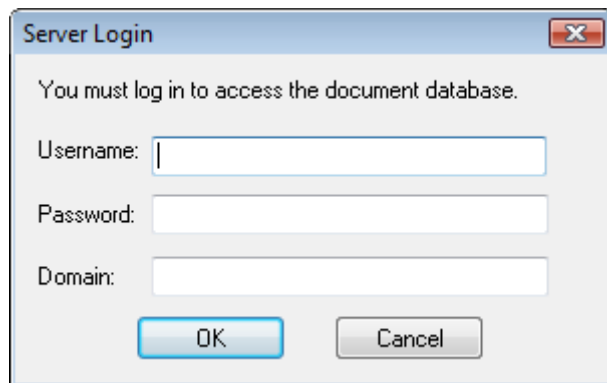
Logging On To the Server



The **Domain** property is initialized to either an empty string or the name of the Execute360 server last accessed.

Before users can interact with an Execute360 server, they must log on to it. The Image Admin control provides a **LoginToServer** method that lets you programmatically log your users on to the Execute360 server specified in the **Domain** property.

In your call to the **LoginToServer** method, you can optionally display the standard **Server Login** dialog box, which permits users to enter their user name, password, and desired domain and then click **OK** to log on to the server.





The Image Admin control also provides a **LogOffServer** method that lets you programmatically log your users off an Execute360 server.

The **Username**, **Password**, and **Domain** text boxes can contain up to 80 alphanumeric characters.

You can also bypass the dialog box and simply pass the user name and password as parameters to the **LoginToServer** method, most likely in response to a user completing and closing a logon dialog box of your own design.

If a user is not logged on and attempts to access the server, the Imaging software displays the standard **Server Login** dialog box automatically — thereby prompting the user to log on to the server.

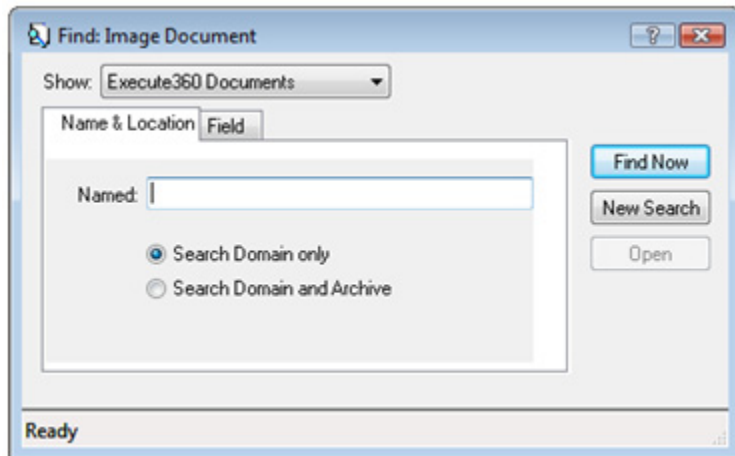
Note: If the unified logon facility is enabled on the Execute360 server being accessed, there is no need to explicitly call the **LoginToServer** method. Each user will be logged on automatically using the Windows user name and password.

Querying Execute360 Documents

The **ShowFindDialog** method of the Image Admin control displays a dialog box that enables users to query an Execute360 server for the documents they want.



If users attempt to access Execute360 documents via the **Open** dialog box, the Imaging software prompts them to click the **Find** button to access the **Find Image Document** dialog box.



In your call to the **ShowFindDialog** method, you can specify the handle to the parent window. Doing so sets the state of the dialog box to *application-modal*; not doing so sets the state of the dialog box to *modeless*.

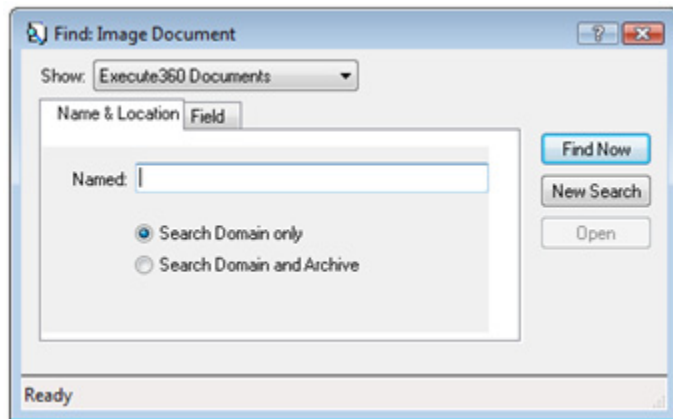
Finding Execute360 Documents

You can search for Execute360 Documents using options provided on the **Name & Location** and **Field** tabs. The following procedures describe how to use these options.

Note: The search uses selection criteria specified on all tabs in the Find Image Document dialog box, including tabs that are not displayed.

Name and Location Tab

The **Name and Location** tab contains a text box for document name and options to specify where to search.



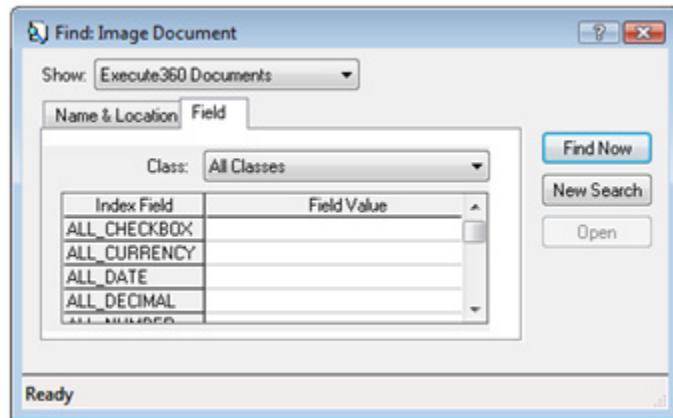


Use the percent (%) wildcard character to represent an unspecified group of characters. Use the underscore (_) to replace one character in a specified position.

- 1 In the **Named** text box, type a full or partial document name. You can use wildcard characters or you can leave the text box blank to retrieve all documents.
- 2 Select **Search Domain Only** or **Search Domain and Archive**.
- 3 Click **Find Now** to initiate the search. A list of documents matching your search criteria is returned.

Field Tab

The Field tab enables you to search by specifying Class and Index field values.



- 1 In the **Class** list box, select the class of documents you want to search or select **All Classes**.
- 2 In the **Field Value** area, adjacent to an entry in the Index Field, type a value. You can specify a complete value or a partial value combined with wildcard values (%) or (_).
- 3 Click **Find Now** to initiate the search. A list of documents matching your search criteria is returned.



To toggle sorting a column of data in ascending or descending order, click the column heading.



The **ImgQuery** and **ImgQueryEnd** methods of the Image Admin control enable you to query an Execute360 server programmatically. The “Demonstration Project” section of this chapter describes and demonstrates these methods.

Opening the Selected Execute360 Document

After users make their selection and click **Open**, the Imaging software writes the path and name of the document to the **Image** property of the Image Admin control.

Use the **Image** property and the **Display** method of the Image Edit control and/or the **Image** property and the **DisplayThumbs** method of the Image Thumbnail control to display the Execute360 document.

Refer to Chapter 2 for more information about displaying images and thumbnails.

Demonstration Project



The demonstration project was developed using Microsoft Visual Basic.

Even if you are not going to include Imaging Server Access in your applications, you will find the sections on adding zoom and annotation functions useful.

To help you use the Imaging ActiveX controls to interact with Imaging 1.x servers, a demonstration project — called Image Server — shows you how to:

- Set server options.
- Browse for Imaging 1.x file and document volumes.
- Browse Imaging 1.x file and document volumes for files and documents to open.
- Query Imaging 1.x document volumes to locate documents to open.
- Zoom an image.
- Invoke the standard annotation tool palette.
- Show and hide annotations.

Note: The Imaging ActiveX Controls online help system identifies the properties, methods, events, parameters, and constants that are available in Imaging for Windows.

Before walking through the demonstration project, read the following sections, which explain the concepts of zooming an image and of working with annotations. Chapter 2 of this guide explains the concepts of displaying an image in the Image Edit control and of multipage image files.

Zooming an Image



You can provide your users with even more control over a displayed image by using the **FitTo** method of the Image Edit control *in addition to* the **Zoom** property. (Refer to Chapter 2 for more information.)

Zoom options affect the way images appear in an Image Edit control. You can zoom an entire image page or just a portion of an image page.

Zooming an Entire Image Page

The **Zoom** property of the Image Edit control lets you set — usually in response to user input — the zoom factor that is applied to image pages when they're displayed or refreshed.

After you set the **Zoom** property, invoke the **Display** method or **Refresh** method of the Image Edit control (as appropriate) to display the image at the new zoom factor.

Most image application developers make zoom functions available to their end users. These functions let users maximize or minimize the image page so that it can be seen more clearly. This is particularly important when users will be *reading* scanned documents or faxes.

Zooming a Portion of an Image Page

The **SelectionRectangle** property and the **ZoomToSelection** method of the Image Edit control let you provide your users with a zoom-to-selection function. The zoom-to-selection function enables them to zoom a *selected* portion of an image page rather than the entire image page.

After you set the **SelectionRectangle** property to **True**, users can draw a selection rectangle on the portion of an image page they want to zoom. Then — usually in response to users clicking a menu item — you invoke the **ZoomToSelection** method.

The **ZoomToSelection** method scales the selected portion of the image so it fits into the current size of the Image Edit control. Then it updates the **Zoom** property with the zoom factor it applied.

Example

Users of your application may want control over the display of image documents to make them easy to read.

Scenario

Eileen receives several scanned business documents in her role as product manager for a major company.

Because the documents contain important information, she needs to be able to read them, which is why you included all of Image Edit's fit-to options in the first version of your application.

But now you realize that users like Eileen need even more control over how image files or documents are displayed. So, in the second version of your application, you include a wide range of zoom options in addition to the fit-to options provided earlier. Users can now select the fit-to or zoom option that produces the best display quality.

Annotations

The Image Edit and Image Annotation Tool Button controls provide several ways to add annotation functions to image-enabled applications. Using them you can:

- Create an annotation tool palette of your own design.
- Invoke a single method that displays a fully-functional, **standard annotation tool palette** to your users.
- Implement custom annotations programmatically.

The method you choose depends on the annotation requirements of your users.

Annotations are digitized versions of the marks or items commonly applied to paper-based documents; for example, highlighting, rubber stamps, lines, and post-it notes. People typically use annotations to emphasize important portions of documents or to add their comments to documents being circulated for review.

Digital annotations go well beyond the capabilities of paper-based annotations. With digital annotations, users can:

- Add, move, and delete annotations at will.
- Modify annotation attributes — such as color, size, text, and visibility.
- Add hypertext links to other pages in the same file, to other files, and to pages on the World Wide Web.

The following table lists the annotation types that are available with the Imaging ActiveX controls.

Imaging for Windows Annotation Types

Annotation Type	Description
Attach-a-Note	Enters text into a background rectangle on an image.
Auto Polygon	Covers a portion of the image with a polygon that can be stretched to the desired size and shape. An auto polygon is text-aware, so if text is detected during creation, the text boundaries will be used to set the polygon boundaries. Because an Auto Polygon annotation starts with only two points, it is easy to create.
Filled Polygon	Covers a portion of an image with a polygon that can be stretched to the desired size and shape.

Imaging for Windows Annotation Types (continued)

Annotation Type	Description
Filled Rectangle	Covers a portion of an imageHighlights text when drawn using the transparent line style.
Freehand Line	Draws a freehand line on a section of text or a portion of an image for emphasis.
Hollow Polygon	Places a polygon around an area of an image for emphasis. The polygon can be stretched to the desired size and shape.
Hollow Rectangle	Places a border around areas of an image for emphasis.
Hyperlink	Enters a hypertext link directly on an image; invokes the Link To dialog box to permit end users to specify the desired link.
Image Embedded	Embeds an actual copy of another image in an image file.
Image Reference	Includes another image in an image file by reference (that is, it links to an external file that contains the image).
Initials	Places system-generated data onto the image including the initials of the user who is logged on, the date, and the time.
OCR Zone	Draws an OCR Text or Picture zone on an image.
Select Annotations	Selects annotation marks for deleting, modifying, moving, or resizing.
Straight Line	Underlines text, demarcates a section of a page, or draws callout linesHighlights text when drawn using the transparent line style.

Imaging for Windows Annotation Types (continued)

Annotation Type	Description
Text	Enters text directly on an image.
Text From File	Enters text from a file on an image.
Text Stamp	Places a text stamp directly on an image.



When using a Filled Rectangle annotation to redact images, make sure that your users select the opaque fill style and that they burn-in the annotation to cover the image permanently.

Users can save annotations separately from the image data within TIFF image files only.

Users can also merge annotations with the image data in a process known as *burning-in*. To save annotations to any file type other than TIFF, the annotations must be burned-in.

Image Annotation Tool Button Control

The Image Annotation Tool Button control lets you create a custom annotation toolbar or palette. Each control is actually a button that invokes an annotation type when your end user clicks it.

Your custom annotation toolbar or palette can contain buttons that provide:

- Discrete annotation types; for example, a Freehand Line annotation type and a Rubber Stamp annotation type.
- The same annotation type with different annotation styles; for example, two buttons that each invoke an Attach-a-Note annotation — one with a yellow background and black text, the other with a red background and white text.
- A combination of both.

The Image Edit control shares several properties and one method with the Image Annotation Tool Button control. Together they let you manage annotation functions when creating a toolbar or palette of your own design. Refer to the next section for more information.

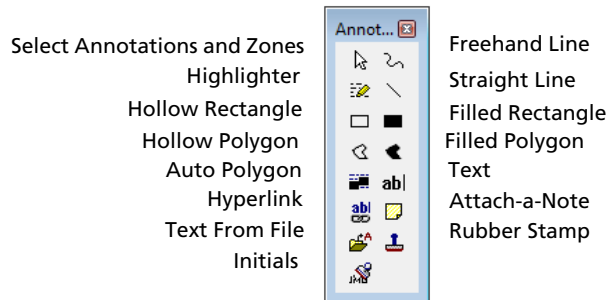
Image Edit Control

The Image Edit control has a fully-functional, **standard annotation tool palette** as well as several properties, methods, and events that let you provide a wide range of annotation functions to your end users.

Standard Annotation Tool Palette

You can invoke the **ShowAnnotationToolPalette** method to display the **standard annotation tool palette** to your users.

The following illustration lists the annotation types that the tool palette provides.



Once displayed, users can right-click a button on the tool palette to set an annotation's properties. Then they can left-click the button to draw the annotation.

Note: The Highlighter annotation is actually a Filled Rectangle annotation with its transparent property selected and its background color property set to yellow.

Programmatic Annotations

The Image Edit control also has an array of properties, methods, and events that enable you to add annotation functions to your applications programmatically. The properties, methods, and events also let you and your users edit and manage existing annotations whether they were drawn programmatically, drawn using the Image Annotation Tool Button control, or drawn using the **standard annotation tool palette**.

Example

Users of your application may want to annotate image files with important comments. They may also want to link image pages to related Web pages.

Scenario

Kim manages a QA (quality assurance) group in a software company that produces applications for engineer-to-order manufacturing firms around the world. Some of the development and testing of the applications is performed in the United States and the remainder is performed in Ireland.

As part of their jobs, Kim and her staff regularly distribute and peer-review scanned specification and test plan documents. Because the QA group is spread across two continents, each analyst relies on e-mail exclusively to exchange the image documents.

Because you included e-mail and image annotation functions in your program, Kim and her staff can use it to retrieve, annotate, and send the image documents. Analysts use its text-related annotation types to enter their review comments directly on the documents. And they use its Hyperlink annotation type to link their comments to related reference pages on the company's intranet site.

The Image Server Project



The file name for the Image Server project is `ImgServer.vbp`.

The Image Server project demonstrates:

- Setting server options.
- Browsing for Imaging 1.x file and document volumes.
- Browsing Imaging 1.x file and document volumes for files and documents to open.
- Querying Imaging 1.x document volumes for documents to open.
- Zooming an image.
- Invoking the standard annotation tool palette.
- Showing and hiding annotations.

The project consists of the following forms and modules:

frmMain — Lets users open image files that reside on their computers or image files and server documents that reside on Imaging 1.x servers. It also lets users browse Imaging 1.x servers for file or document volumes, as well as zoom and annotate the image files or documents they open.

frm1xCDFD — Enables users to query an Imaging 1.x document volume for documents by location, using the following hierarchy: `Cabinet\Drawer\Folder\Document`

frm1xQuery — Enables users to query an Imaging 1.x document volume for documents by name, creation date, modification date, or keyword.

The project uses the following Imaging controls:

- One Image Admin control
- One Image Edit control

It uses the following methods of the Image Admin control to provide setup, open, browse, and query functions:

Show1xServerOptDlg method — To display the **Imaging Server Options** dialog box, which lets users set Imaging 1.x server options.

ShowFileDialog method — To display the **Open** dialog box, which lets users select the image files or server documents they want to open.

Browse1x method — To display the **Browse 1.x** dialog box, which lets users browse the Imaging 1.x server for server file and/or document volumes.

CreateDirectory method — To create a cabinet, drawer, and/or folder.

ConvertDate method — To convert conventional (Gregorian) dates to Julian dates when using the **ImgQuery** method to query 1.x document volumes.

ImgQuery method — To query Imaging 1.x document volumes.

ImgQueryEnd method — To complete a query and free associated resources.

And it uses the following methods in the Image Edit control to provide the image display and annotation functions:

Display method — To display the image file or server document specified in the **Image** property of the Image Edit control.

Refresh method — To redisplay the current image in the Image Edit control.

ShowAnnotationToolPalette method — To show the **standard annotation tool palette**.

HideAnnotationToolPalette method — To hide the **standard annotation tool palette**.

ShowAnnotationGroup method — To show annotations.

HideAnnotationGroup method — To hide annotations.

Setting Server Options

Start the Image Server project. On the **Server** menu, click **Imaging Server Options**.

The `mnuServerItem_Click()` event procedure of `frmMain` executes the appropriate code in its `Select Case` statement (as shown in the following code snippet). Each `Case` expression corresponds to the `Index` value of an option on the **Server** menu.

In this case, the procedure invokes the `Show1xServerOptDlg` method of the Image Admin control, which displays the Imaging Server Options dialog box described in the “Setting Imaging 1.x Server Options” section of this chapter.

Make the appropriate entries on the **Imaging Server Options** dialog box and then click **OK**.

```
Private Sub mnuServerItem_Click(Index As Integer)
    Dim strPath As String
    Dim strPathType As String

    On Error Resume Next

    Select Case Index

        Case 0 'Access Imaging Server Options dialog box

            ImgAdmin1.Show1xServerOptDlg

            .
            .
            .
        End Select
    End Sub
```

Browsing for Imaging 1.x File and Document Volumes

To browse an Imaging 1.x server, on the **Server** menu, click:

Browse 1.x Files — To browse for file volumes.

Browse 1.x Documents — To browse for document volumes.

Browse 1.x Files and Documents — To browse for both file and document volumes.

The `mnuServerItem_Click()` event procedure of `frmMain` (shown in the following code snippet) executes the appropriate code in the `Case 2,3,4` statement — depending on the Index value of the **Server** menu option clicked. For each Index value, the procedure invokes the **Browse 1.x** method of the Image Admin control with the appropriate parameter values:

Browse1xScope parameter — Determines the volume type to browse:

- Index = 2 passes the `BrowseFiles` constant (literal 0).
- Index = 3 passes the `BrowseDocuments` constant (literal 1).
- Index = 4 passes the `BrowseBoth` constant (literal 2).

Title parameter — Determines the text that appears in the title bar of the `Browse1.x` dialog box:

- Index = 2 passes “Browse 1.x File Volumes”.
- Index = 3 passes “Browse 1.x Document Volumes”.
- Index = 4 passes “Browse 1.x File and Document Volumes”.

Caption parameter — Determines the prompt that appears in the title bar of the **Browse 1.x** dialog box. Each invocation of the **Browse1.x** method passes “Select the Desired Path”.

hParentWnd parameter — Assigns a parent window handle to the **Browse 1.x** dialog box. Each invocation of the **Browse1.x** method passes `frmMain.hWnd`, which is the handle to the main form.



When the user clicks **Cancel**, a “Cancel is pressed” error condition occurs. The `mnuServerItem_Click()` event shows you how to handle it.

The **Browse1x** method displays the **Browse 1.x** dialog box described in the “Browsing for Volumes” section of this chapter.

Browse the file and/or document volumes. Then make your selection and click **OK**. The Imaging software writes the path selected to the **Browse1xReturnedPath** property of the Image Admin control and the type of path selected to the **Browse1xReturnedType** property.

```

Private Sub mnuServerItem_Click(Index As Integer)
    Dim strPath As String
    Dim strPathType As String

    On Error Resume Next

    Select Case Index

        Case 0 'Access Imaging Server Options dialog box
            ImgAdmin1.ShowIxsServerOptDlg

        Case 2, 3, 4 'Browse 1.x File, Document, or File and Document Volumes
            If Index = 2 Then
                ImgAdmin1.BrowseIxs BrowseFiles, "Browse 1.x File Volumes", _
                    "Select the Desired Path", frmMain.hWnd
            ElseIf Index = 3 Then
                ImgAdmin1.BrowseIxs BrowseDocuments, _
                    "Browse 1.x Document Volumes", _
                    "Select the Desired Path", frmMain.hWnd
            Else
                ImgAdmin1.BrowseIxs BrowseBoth, _
                    "Browse 1.x File and Document Volumes", _
                    "Select the Desired Path", frmMain.hWnd
            End If
            If Err.Number = CANCEL_PRESSED Then      '32755 = Cancel pressed
                Exit Sub
            ElseIf ImgAdmin1.BrowseIxsReturnedPath = "" Then 'Not installed.
                Exit Sub
            ElseIf ImgAdmin1.StatusCode <> 0 Then
                MsgBox Err.Description & " (ImgAdmin error " & _
                    Hex(ImgAdmin1.StatusCode) & ")", vbCritical
                Exit Sub
            End If

            strPath = ImgAdmin1.BrowseIxsReturnedPath
            strPathType = ImgAdmin1.BrowseIxsReturnedType

            .
            .
            .
        End Select

    End Sub

```


Opening 1.x Files and Documents

To simply browse for and then open an Imaging 1.x image file or document for display, on the **File** menu, click **Open**.

The `mnuFileOpen_Click()` event procedure of `frmMain` executes its code (as shown in the following code snippet). Specifically, it invokes the **ShowFileDialog** method of the Image Admin control with the following parameter values:

DialogOption parameter — Passes the `OpenDlg` constant (literal 0) to display the **Open** dialog box.

hParentWnd parameter — Passes the handle to the main window (`frmMain.hWnd`).

The **ShowFileDialog** method displays the standard **Open** dialog box described in the “Browsing for Files and Documents” section earlier in this chapter.



When the user clicks **Cancel**, a “Cancel is pressed” error condition occurs. The `mnuFileOpen_Click()` event shows you how to handle it.

Browse and then make your image file or server document selection, then click **OK**. The Imaging software writes the path and file (or document) selected to the **Image** property of the Image Admin control.

Next, the `cmdFileOpen_Click()` event procedure invokes the public subroutine, `PerformFileOpen(ImgAdmin1.Image)`, which opens and displays the image file or server document selected.

```
Private Sub mnuFileOpen_Click()

    On Error Resume Next

    '-----
    ' Set Flags to 0, and then show the Open dialog box.
    '-----
    ImgAdmin1.Flags = 0
    ImgAdmin1.ShowFileDialog OpenDlg, frmMain.hWnd

    '-----
    ' If the Cancel button was pressed, exit the subroutine.
    ' If a different error occurred, display a message box and
    ' exit the subroutine.
    '-----
    If Err.Number = CANCEL_PRESSED Then      '32755 = Cancel pressed
        Exit Sub
    ElseIf ImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(ImgAdmin1.StatusCode) & ")", vbCritical
        Exit Sub
    End If

    '-----
    ' Display the image.
    '-----
    Call PerformFileOpen(ImgAdmin1.Image)

End Sub
```

Querying 1.x Document Manager Databases

Note: This portion of the Image Server project refers to document volumes as document manager databases or document managers. Document volumes were referred to as document manager databases in earlier versions of the Imaging 1.x software.

To query an Imaging 1.x database, on the **Server** menu, click:

1.x Query by Cabinet\Drawer\Folder\Document — To query 1.x document manager databases by Cabinet, Drawer, Folder, and Document.

1.x Query — To query 1.x document manager databases by document name, date, or keyword.

The following sections discuss each type of query.

Performing a 1.x Query by Cabinet\Drawer\Folder\Document

After you make your menu selection, the **1.x Cabinet\Drawer\Folder\Document** window (frm1.xCDFD) loads without being shown. Its `Form_Load()` event procedure (shown in the following code snippet) invokes the **ImgQueryEnd** method of the Image Admin control to clear any previous Imaging queries and to free associated system resources.



Refer to the online help for more information about the **ImgQuery** method of the Image Admin control.

Next, the procedure invokes the **ImgQuery** method to initiate a new query, passing to it the following parameters:

vScope parameter — Constant `DMVOLUMES` (literal 1), which sets the method so it performs a query for Imaging 1.x document manager databases.

szQueryTerms parameter — A blank string, which makes the method return the available Imaging 1.x databases.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query. You can extract the results of a query by using a `For Each...Next` statement in the following format:

```
For Each VariantItem In objResults  
    ' Your code that processes each VariantItem  
Next VariantItem
```

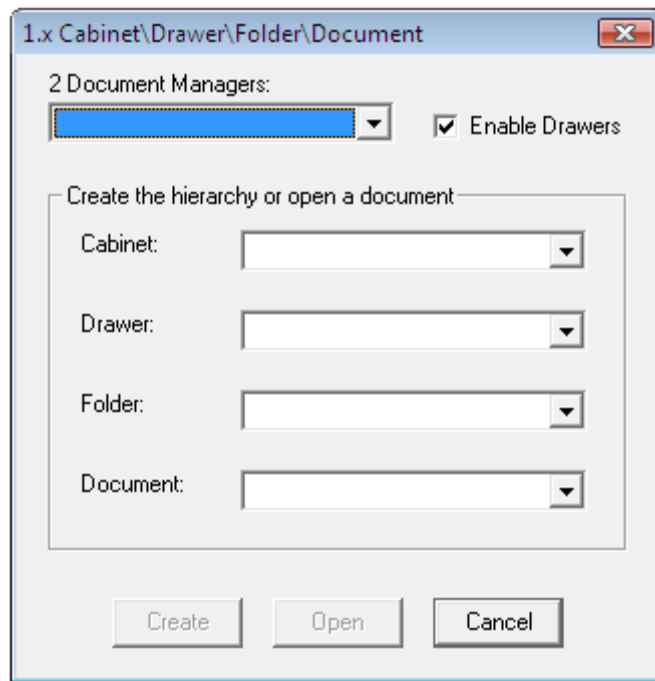
The query finds the available Imaging 1.x databases. Then the `Form_Load()` event procedure loads them from the `objResults` object variable into the **Document Manager** combo box (`cboDocManager`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method. Both actions free system resources associated with the query.

The procedure wraps up its work by:

- Showing the **1.x Cabinet\Drawer\Folder\Document** window.
- Displaying the number of document manager databases found.

- Giving focus to the **Document Manager** combo box.



```

Private Sub Form_Load()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    ' Perform an ImgQuery for all Document Manager databases.
    '
    ImgAdmin1.ImgQueryEnd
    ImgAdmin1.ImgQuery "DMVOLUMES", "", objResults
    '
    ' If an error occurred, display a message box and exit.
    '
    If ImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(ImgAdmin1.StatusCode) & ")", vbCritical
        Exit Sub
    End If
    '
    ' Store the results in the cboDocManager combo box.
    '
    For Each vntItem In objResults
        If vntItem <> "" Then
            cboDocManager.AddItem vntItem
        End If
    Next vntItem
    '
    ' End the query.
    '
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '
    ' Display the number of Document Managers found.
    '
    If cboDocManager.ListCount = 1 Then
        strSinglePlural = " Document Manager:"
    Else
        strSinglePlural = " Document Managers:"
    End If
    lblDocumentManager.Caption = cboDocManager.ListCount & strSinglePlural
    '
    ' Show the form, set focus to cboDocManager.
    '
    Me.Show
    cboDocManager.SetFocus
End Sub

```



Once you select a document manager database, you can create a new cabinet, drawer, and/or folder by entering the names in the respective combo boxes and clicking the **Create** button.

The cmdCreate_Click() event procedure concatenates a string containing your entries and invokes the **Create Directory** method of the Image Admin control to create the cabinet, drawer, and/or folder you specified (code not shown).

On the **1.x Cabinet\Drawer\Folder\Document** window, click the desired document manager database in the **Document Managers** combo box. The `cboDocManager_Click()` event procedure fires and executes its code (as shown in the following code snippet).

The basic task of this event procedure is to query the selected document manager database for all of its cabinets and to load them in the **Cabinets** combo box. To accomplish this task, it saves the path to the document manager you selected in the `mstrDocManager` module variable and then invokes the **ImgQuery** method with the following parameters:

vScope parameter — The result of `Mid(mstrDocManager, 9)`, which sets the method so it performs a query on the selected document manager.

Note: Using the **Mid** function with a **Start** value of 9 is required because the `mstrDocManager` variable contains the path to the document manager in the following format:

Image://server/database:

... and the **ImgQuery** method is only interested in the `server/database:` part. Accordingly, the **Mid** function eliminates the `Image://` part and returns a variant (string) value of `"server/database:"`.

szQueryTerms parameter — The string `"findcabinets"`, which is contained in the `mstrQuery` module variable. This parameter value makes the method return the cabinets in the referenced database.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds the cabinets in the selected database and returns them via the collection object in the following format:

Image://server/database:\cabinet

Because we're only interested in cabinet names, the procedure invokes the public function `GetEndString()`, which returns just the cabinet names. The `cboDocManager_Click()` event procedure then loads the cabinet names into the **Cabinets** combo box (`cboCabinet`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method.

The procedure wraps up its work by displaying the number of cabinets found and giving focus to the **Cabinets** combo box.


```

Private Sub cboDocManager_Click()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    mstrDocManager = cboDocManager.Text
    '-----
    ' Perform an ImgQuery for Cabinet names, load them in the
    ' cboCabinet combo box.
    '-----
    mstrQuery = "findcabinets"
    ImgAdmin1.ImgQuery Mid(mstrDocManager, 9), mstrQuery, objResults
    For Each vntItem In objResults
        If vntItem <> "" Then
            cboCabinet.AddItem GetEndString(vntItem, "\")
        End If
    Next vntItem
    '-----
    ' End the query.
    '-----
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '-----
    ' Display the number of Cabinets found.
    '-----
    If cboCabinet.ListCount = 1 Then
        strSinglePlural = " Cabinet:"
    Else
        strSinglePlural = " Cabinets:"
    End If
    lblCabinet.Caption = cboCabinet.ListCount & strSinglePlural
    '-----
    ' Set the focus to cboCabinet.
    '-----
    cboCabinet.SetFocus
End Sub

```



Be sure that the **Enable Drawers** check box has a check mark next to it. If it does not, the `cboCabinet_Click()` procedure performs a query for folders.

On the **1.x Cabinet\Drawer\Folder\Document** window, click the desired cabinet in the **Cabinets** combo box. The `cboCabinet_Click()` event procedure fires and executes its code (as shown in the following code snippet).

The basic task of this event procedure is to query the selected database and cabinet for all of its drawers and to load them in the **Drawers** combo box. To accomplish this task, it saves the path to the document manager and cabinet you selected in the `mstrDocManager` and `mstrCabinet` module variables respectively. Then it invokes the **ImgQuery** method with the following parameters:

vScope parameter — The result of `Mid(mstrDocManager, 9)`, which sets the method so it performs a query on the selected document manager.

szQueryTerms parameter — The concatenated string `"finddrawers cabinet=" & mstrCabinet`, which is contained in the `mstrQuery` module variable. This parameter value makes the method return the drawers in the referenced database and cabinet.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds the drawers in the selected database and returns them via the collection object in the following format:

```
Image://database:\cabinet\drawer
```

Because we're only interested in drawer names, the procedure invokes the public function `GetEndString()`, which returns just the drawer names. The `cboCabinet_Click()` event procedure then loads the drawer names into the **Drawers** combo box (`cboDrawer`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method.

The procedure wraps up its work by displaying the number of drawers found and giving focus to the **Drawers** combo box.

```

Private Sub cboCabinet_Click()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    mstrDocManager = cboDocManager.Text
    mstrCabinet = cboCabinet.Text
    '-----
    ' Perform an ImgQuery, and store the Drawer or Folder names
    ' in the appropriate combo box.
    '-----
    If chkEnableDrawers.Value = Checked Then
        mstrQuery = "finddrawers cabinet=" & mstrCabinet
        ImgAdmin1.ImgQuery Mid(mstrDocManager, 9), mstrQuery, objResults
        For Each vntItem In objResults
            If vntItem <> "" Then
                cboDrawer.AddItem GetEndString(vntItem, "\")
            End If
        Next vntItem
    .
    .
    .
End If
'-----
' End the query.
'-----
Set objResults = Nothing
ImgAdmin1.ImgQueryEnd
.
.
.
End Sub

```

On the **1.x Cabinet\Drawer\Folder\Document** window, click the desired drawer in the **Drawers** combo box. The `cboDrawer_Click()` event procedure fires and executes its code (as shown in the following code snippet).

The basic task of this event procedure is to query the selected database, cabinet, and drawer for all of its folders and to load them in the **Folders** combo box. To accomplish this task, it saves the path to the document manager, cabinet, and drawer you selected in the `mstrDocManager`, `mstrCabinet`, and `mstrDrawer` module variables respectively. Then it invokes the **ImgQuery** method with the following parameters:

vScope parameter — The result of `Mid(mstrDocManager, 9)`, which sets the method so it performs a query on the selected document manager.

szQueryTerms parameter — The concatenated string `"findfolders cabinet=" & mstrCabinet & ":drawer=" & mstrDrawer`, which is contained in the `mstrQuery` module variable. This parameter value makes the method return the folders in the referenced database, cabinet, and drawer.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds the folders in the selected database and returns them via the collection object in the following format:

```
Image://database:\cabinet\drawer\folder
```

Because we're only interested in folder names, the procedure invokes the public function `GetEndString()`, which returns just the folder names. The `cboDrawer_Click()` event procedure then loads the folder names into the **Folders** combo box (`cboFolder`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method.

The procedure wraps up its work by displaying the number of folders found and giving focus to the **Folders** combo box.

```

Private Sub cboDrawer_Click()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    mstrDocManager = cboDocManager.Text
    mstrCabinet = cboCabinet.Text
    mstrDrawer = cboDrawer.Text
    '-----
    ' Perform an ImgQuery, and store the Folder names in the
    ' cboFolder combo box.
    '-----
    mstrQuery = "findfolders cabinet=" & mstrCabinet & ";drawer=" & mstrDrawer
    ImgAdmin1.ImgQuery Mid(mstrDocManager, 9), mstrQuery, objResults

    For Each vntItem In objResults
        If vntItem <> "" Then
            cboFolder.AddItem GetEndString(vntItem, "\")
        End If
    Next vntItem
    '-----
    ' End the query.
    '-----
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '-----
    ' Display the number of Folders found.
    '-----
    If cboFolder.ListCount = 1 Then
        strSinglePlural = " Folder:"
    Else
        strSinglePlural = " Folders:"
    End If
    lblFolder.Caption = cboFolder.ListCount & strSinglePlural
    '-----
    ' Set the focus to cboFolder.
    '-----
    cboFolder.SetFocus
End Sub

```

On the **1.x Cabinet\Drawer\Folder\Document** window, click the desired folder in the **Folders** combo box. The `cboFolder_Click()` event procedure fires and executes its code (as shown in the following code snippet).

The basic task of this event procedure is to query the selected database, cabinet, drawer, and folder for all of its documents and to load them in the **Documents** combo box. To accomplish this task, it saves the path to the document manager, cabinet, drawer, and folder you selected in the `mstrDocManager`, `mstrCabinet`, `mstrDrawer`, and `mstrFolder` module variables respectively. Then it invokes the **ImgQuery** method with the following parameters:

vScope parameter — The result of `Mid(mstrDocManager, 9)`, which sets the method so it performs a query on the selected document manager.

szQueryTerms parameter — The concatenated string `"finddocs cabinet = " & mstrCabinet & " drawer = " & mstrDrawer & " folder = " & mstrFolder`, which is contained in the `mstrQuery` module variable. This parameter value makes the method return the documents in the referenced database, cabinet, drawer, and folder.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds the documents in the selected database and returns them via the collection object in the following format:

```
Image://database:\cabinet\drawer\folder\document
```

Because we're only interested in document names, the procedure invokes the public function `GetEndString()`, which returns just the document names. The `cboFolder_Click()` event procedure then loads the document names into the **Documents** combo box (`cboDocument`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method.

The procedure wraps up its work by displaying the number of documents found and giving focus to the **Documents** combo box.

```
Private Sub cboFolder_Click()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    mstrDocManager = cboDocManager.Text
    mstrCabinet = cboCabinet.Text
    mstrDrawer = cboDrawer.Text
    mstrFolder = cboFolder.Text
    '-----
    ' Perform an ImgQuery, and store the Document names in the
    ' cboDocument combo box.
    '-----
    mstrQuery = "finddocs cabinet = " & mstrCabinet & " drawer = " _
        & mstrDrawer & " folder = " & mstrFolder
    ImgAdmin1.ImgQuery Mid(mstrDocManager, 9), mstrQuery, objResults

    For Each vntItem In objResults
        If vntItem <> "" Then
            cboDocument.AddItem GetEndString(vntItem, "\")
        End If
    Next vntItem
    '-----
    ' End the query.
    '-----
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '-----
    ' Display the number of Documents found.
    '-----
    If cboDocument.ListCount = 1 Then
        strSinglePlural = " Document:"
    Else
        strSinglePlural = " Documents:"
    End If
    lblDocument.Caption = cboDocument.ListCount & strSinglePlural
    '-----
    ' Set the focus to cboDocument.
    '-----
    cboDocument.SetFocus
End Sub
```

On the **1.x Cabinet\Drawer\Folder\Document** window, click the desired document in the **Document** combo box and then click the **Open** button. The `cmdOpen_Click()` event procedure invokes the public subroutine, `PerformFileOpen(strDocManagerCDFD)`, which opens and displays the document selected.

Performing a 1.x Query by Name, Date, or Keyword

After you select **1.x Query** on the **Server** menu, the **1.x Query window** (`frm1.xQuery`) loads without being shown. Its `Form_Load()` event procedure (shown in the following code snippet) invokes the **ImgQueryEnd** method of the Image Admin control to clear any previous Imaging queries and to free associated system resources.

Next, the procedure invokes the **ImgQuery** method, passing to it the following parameters:

vScope parameter — Constant `DMVOLUMES` (literal 1), which sets the method so it performs a query for Imaging 1.x document manager databases.

szQueryTerms parameter — A blank string, which makes the method return the available Imaging 1.x databases.

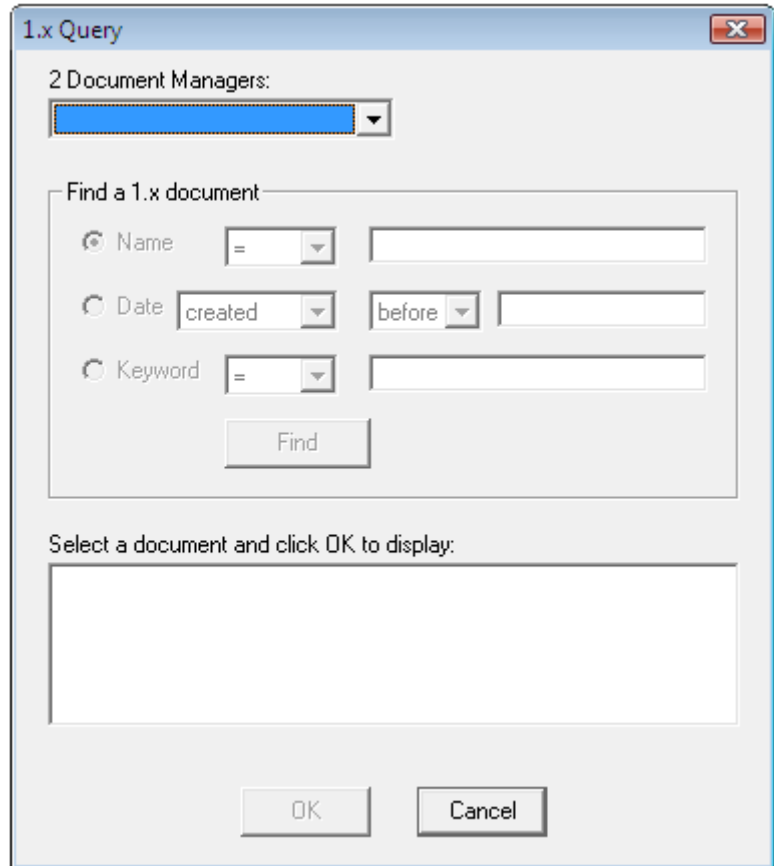
iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds all of the available Imaging 1.x databases. The `Form_Load()` event procedure loads them from the `objResults` object variable into the **Document Manager** combo box (`cboDocManager`).

The procedure ends the query by setting `objResults` to `Nothing` and by invoking the **ImgQueryEnd** method. Both actions free system resources associated with the query.

The procedure wraps up its work by:

- Showing the **1.x Query** window.
- Displaying the number of document manager databases found.
- Giving focus to the **Document Manager** combo box.



The screenshot shows a dialog box titled "1.x Query". At the top, it says "2 Document Managers:" followed by a blue dropdown menu. Below this is a section titled "Find a 1.x document" which contains three radio buttons: "Name", "Date", and "Keyword". The "Name" radio button is selected. Each radio button has associated dropdown menus for operators and values. For "Name", the operator is "=" and the value field is empty. For "Date", the operator is "created", the comparison is "before", and the value field is empty. For "Keyword", the operator is "=" and the value field is empty. A "Find" button is located below these options. At the bottom of the dialog, there is a text label "Select a document and click OK to display:" above a large empty text area. At the very bottom are "OK" and "Cancel" buttons.

```

Private Sub Form_Load()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    ' Perform an ImgQuery for all Document Manager databases.
    '
    ImgAdmin1.ImgQueryEnd
    ImgAdmin1.ImgQuery "DMVOLUMES", "", objResults
    '
    ' If an error occurred, display a message box and exit.
    '
    If ImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(ImgAdmin1.StatusCode) & ")", vbCritical
        Exit Sub
    End If
    '
    ' Store the results in the cboDocManager combo box.
    '
    For Each vntItem In objResults
        If vntItem <> "" Then
            cboDocManager.AddItem vntItem
        End If
    Next vntItem
    '
    ' End the query.
    '
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '
    ' Display the number of Document Managers found.
    '
    If cboDocManager.ListCount = 1 Then
        strSinglePlural = " Document Manager:"
    Else
        strSinglePlural = " Document Managers:"
    End If
    lblDocumentManager.Caption = cboDocManager.ListCount & strSinglePlural
    '
    ' Show the form, set focus to cboDocManager.
    '
    Me.Show
    cboDocManager.SetFocus
End Sub

```

On the **1.x Query** window, click the desired document manager database in the **Document Managers** combo box. The `cboDocManager_Click()` event procedure assigns the document manager database you selected to the `mstrDocManager` module variable (code not shown).

In the **Find 1.x Document** area, click the type of query you want to perform.

If you clicked the:



When you select the **Like** operator, you can use the asterisk (*) wildcard character to represent a group of characters and a question mark (?) to match any single character.

Name option button (Query by Document) — Click the desired boolean operator and then enter the name of the document you are trying to locate in the adjacent text box.

Date option button (Query by Date) — Click whether to search for documents that were *created* or *modified*, then click the desired boolean operator (including *before*, *after*, and *on*). Finally, enter the desired date in the adjacent text box.

Keyword option button (Query by Date) — Click the desired boolean operator and then enter, in the adjacent text box, the keyword whose documents you want to search for. Use the date format set in Regional Settings.

Click the **Find** button. The `cmdFind_Click()` event procedure fires and executes its code (as shown in the following code snippet).

The basic task of this event procedure is to find all of the Imaging 1.x documents that satisfy the parameters you specified and to load them in the list box control at the bottom of the window.

To accomplish this task, the procedure evaluates the **Value** property of each option button on the form. When it finds the option button you clicked, it builds an appropriate **Query Terms** string and assigns it to the `mstrQuery` module variable. (The procedure passes the value of this variable to the **ImgQuery** method later as the `szQueryTerms` parameter.)



When building your own **QueryTerms** strings, be sure to include a space between each element.

The composition of the **Query Terms** string depends on the type of query you are performing. If you are performing a:

Query by Document — The string contains:

- The `finddocs document` qualifier.
- The selected boolean operator from the adjacent combo box.
- The document name entered in the adjacent text box.

Query by Date — The string contains:

- The `finddocs created` or `finddocs modified` qualifier, depending on whether you selected *created* or *modified* in the adjacent combo box.
- The selected boolean operator from the next combo box.
- The date returned by the **ConvertDate** method of the Image Admin control (which converted the Gregorian date you entered in the adjacent text box to a Julian date).

Query by Keyword — The string contains:

- The `finddocs keyword` qualifier.
- The selected boolean operator from the adjacent combo box.
- The keyword entered in the adjacent text box.

With the **Query Terms** string now composed and assigned, the `cmdFind_Click()` event procedure invokes the **ImgQuery** method, passing to it the following parameters:

vScope parameter — The result of `Mid(mstrDocManager, 9)`, which sets the method so it performs a query on the selected document manager.

szQueryTerms parameter — The concatenated **Query Terms** string from the `mstrQuery` module variable, which sets the method so it performs the query you specified.

iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

The query finds the documents in the selected database and returns them via the collection object in the following format:

Image://database:\cabinet\drawer\folder\document

Then, the `cmdFind_Click()` event procedure ends the query by setting `objResults` to `Nothing` and by invoking the

ImgQueryEnd method.

The procedure wraps up its work by displaying the documents in the list box at the bottom of the window.

```

Private Sub cmdFind_Click()
    Dim objResults As Object
    Dim vntItem As Variant
    Dim strConvertedDate As String

    lstResults.Clear
    '-----
    ' Perform a query; store Doc names in the lstResults listbox.
    '-----
    If optQuery(0).Value = True Then      'Query by Document
        mstrQuery = "finddocs document " & _
            cboName.List(cboName.ListIndex) & " " & txtName.Text

    ElseIf optQuery(1).Value = True Then  'Query by Date
        strConvertedDate = ImgAdmin1.ConvertDate(txtDate.Text)
        mstrQuery = "finddocs " & _
            cboDateName.List(cboDateName.ListIndex) & " " & _
            cboDate.List(cboDate.ListIndex) & " " & strConvertedDate

    ElseIf optQuery(2).Value = True Then  'Query by Keyword
        mstrQuery = "finddocs keyword " & _
            cboKeyword.List(cboKeyword.ListIndex) & " " & txtKeyword.Text
    End If

    ImgAdmin1.ImgQuery Mid(mstrDocManager, 9), mstrQuery, objResults

    For Each vntItem In objResults
        If vntItem <> "" Then
            lstResults.AddItem vntItem
        End If
    Next
    '-----
    ' End the query.
    '-----
    Set objResults = Nothing
    ImgAdmin1.ImgQueryEnd
    '-----
    ' Display the number of documents found.
    '-----
    If lstResults.ListCount = 0 Then
        lblResults.Caption = SELECT_NONE
    ElseIf lstResults.ListCount = 1 Then
        lblResults.Caption = SELECT_SINGULAR
    Else
        lblResults.Caption = SELECT_PLURAL1 & lstResults.ListCount &
            SELECT_PLURAL2
    End If
End Sub

```

Select a document and click **OK**. The `cmdOK_Click()` event procedure invokes the public subroutine, `PerformFileOpen (ImgAdmin1.Image)`, which opens and displays the server document you selected (code not shown).

Zooming an Image

Open an image file or server document. After the image appears in the Image Edit control, on the **Zoom** menu, click the desired zoom factor.

The `mnuZoomFactorItem_Click` event procedure fires and executes the appropriate code in its `Select Case` statement (as shown in the following code snippet).

Each `Case` expression corresponds to the `Index` value of a **Zoom** menu item. Further, each `Case` expression sets the **Zoom** property of the Image Edit control to an appropriate zoom factor.

With the **Zoom** property now set, the procedure completes its work by invoking the **Refresh** method of the Image Edit control, which redisplay the image at its new zoom factor.

```
Private Sub mnuZoomFactorItem_Click(Index As Integer)
    Dim intIndex As Integer
    '-----
    ' Uncheck all the zoom menu items.
    '-----
    For intIndex = 0 To 5
        mnuZoomFactorItem(intIndex).Checked = False
    Next intIndex
    '-----
    ' Set the zoom factor.
    '-----
    Select Case Index
        Case 0 '25%
            ImgEdit1.Zoom = 25
        Case 1 '50%
            ImgEdit1.Zoom = 50
        Case 2 '75%
            ImgEdit1.Zoom = 75
        Case 3 '100%
            ImgEdit1.Zoom = 100
        Case 4 '200%
            ImgEdit1.Zoom = 200
        Case 5 '400%
            ImgEdit1.Zoom = 400
    End Select
    '-----
    ' Check the menu item that was clicked.
    '-----
    mnuZoomFactorItem(Index).Checked = True
    '-----
    ' Refresh the image at the new zoom factor.
    '-----
    ImgEdit1.Refresh
End Sub
```


Invoking the Standard Annotation Tool Palette

Open an image file or server document. After the image appears in the Image Edit control, on the **Annotations** menu, click **Show Annotation Toolbar**.



You can include parameters in a call to the **ShowAnnotationToolPalette** method. The parameters control:

- Whether users can set annotation properties.
- Where the tool palette will appear on the screen.
- The tool tip text that appears when the mouse pointer hovers over a button on the tool palette.

The `mnuAnnotationItem_Click` event procedure fires and executes the appropriate code in its `Select Case` statement (as shown in the following code snippet).

Each `Case` expression corresponds to the `Index` value of an **Annotation** menu item.

As long the corresponding menu item is not checked, the `Case 1` expression invokes the **ShowAnnotationToolPalette** method of the Image Edit control, which displays the **standard annotation tool palette**. (Refer to the “Annotations” section earlier in this chapter for more information about annotations and the **annotation tool palette**.)

If the corresponding menu item is checked, the `Case 1` expression invokes the **HideAnnotationToolPalette** method, which closes the **annotation tool palette**.

Closing or hiding the **standard annotation tool palette** causes the **ToolPaletteHidden()** event of the Image Edit control to fire.

Code within it removes the check mark from the **Show Annotation Toolbar** menu item (code not shown).

```
Private Sub mnuAnnotationItem_Click(Index As Integer)

    Select Case Index

        Case 0 'Show Annotations
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                ImgEdit1.HideAnnotationGroup
            Else
                mnuAnnotationItem(Index).Checked = True
                ImgEdit1.ShowAnnotationGroup
            End If

        Case 1 'Show Annotation Toolbar
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                ImgEdit1.HideAnnotationToolPalette
            Else
                mnuAnnotationItem(Index).Checked = True
                ImgEdit1.ShowAnnotationToolPalette
            End If

    End Select

End Sub
```

Image-Enabling a Web Page

This chapter explains how to use HTML source code to image-enable a Web page. It assumes that you are using Microsoft Internet Explorer to display the page.

In This Chapter

Defining the Controls	154
Demonstration Project	156

Defining the Controls

The process of defining the Imaging ActiveX controls enables you to include them on your Web page.

As a minimum, each object definition must contain a class identifier (classid attribute), name identifier (ID attribute), as well as its initial dimensions in pixels (width, height attributes). It can also contain other attributes, such as the alignment of the object (align attribute).

An object definition can also contain one or more param element definitions along with the name and value attributes of each. If desired, you can use the param elements to pass values to the object.

To define the Imaging ActiveX controls

- 1 Create an HTML file using the program of your choice.
- 2 Include the usual initial HTML command elements, such as `<html>`, `<head>`, `<title>`, `<body>`
- 3 Define each Imaging ActiveX control you want to use by including the `<object>` command element and setting its properties.

Class Identifiers for Imaging Controls

The following table lists the class identifiers for the Imaging controls.

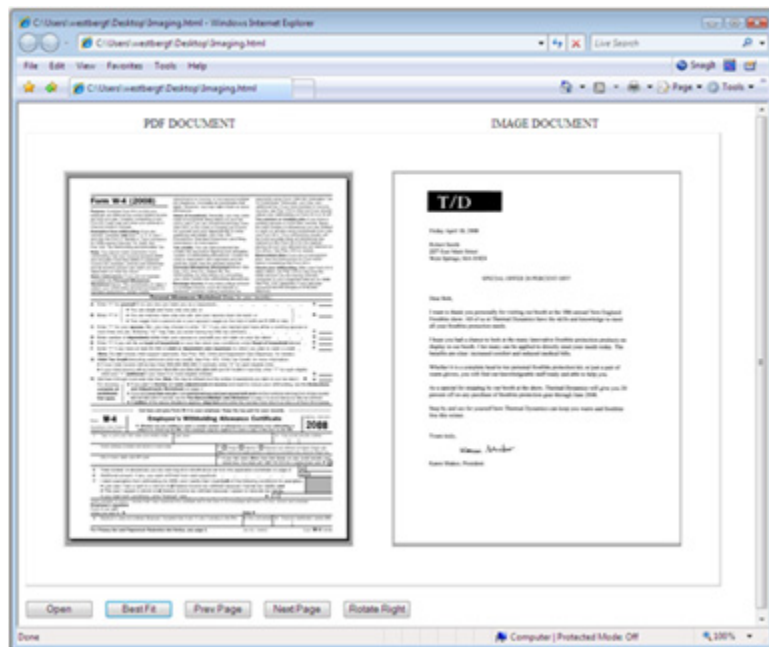
Identifier	Control
{009541A0-3B81-101C-92F3-040224009C02}	Image Admin
{6D940285-9F11-11CE-83FD-02608C3EC08A}	Annotation Tool Button
{6D940280-9F11-11CE-83FD-02608C3EC08A}	Image Edit
{8FC248E3-D4D9-11CF-8727-0020AFA5DCA7}	Image OCR
{CDDC07D5-9475-4049-A48C-33CC516EF038}	Image PDF
{84926CA0-2941-101C-816F-0E6013114B7F}	Image Scan
{E1A6B8A0-3603-101C-AC6E-040224009C02}	Image Thumbnail

Demonstration Project

The `Imgctls.html` project demonstrates how to perform the following functions:

- Display a multipage image file or PDF document file on a Web page.
- Scale the displays for Best Fit.
- Navigate the pages of the displayed documents.
- Rotate an image page 90 degrees to the right.

The following illustration shows the `Imgctls` project running in Microsoft Internet Explorer.



Project Overview

The Imgctls project consists of HTML and Javascript.

It contains the following components:

- One Image Edit control
- One Image PDF control
- One Image Admin control
- Five Button objects

Starting Imgctls

Start the Imgctls project by opening the Imgctls.html file in Internet Explorer. The browser begins loading the code.

Initialization

First, the code within Imgctls.html directs the browser to define the Image ActiveX controls. Each Imaging ActiveX control object definition contains the ID, width, height, and classid attributes, as shown in the following code snippet. The height and width of the Admin control are 0.

```
<table border cellspacing = 50 >
<tr align = center>
<td><OBJECT
    ID="PDFCtrl"
    WIDTH=384
    HEIGHT=498
    CLASSID="CLSID:CDDC07D5-9475-4049-A48C-33CC516EF038">
</OBJECT>
</td>

<td>
<OBJECT
    ID="EditCtrl"
    WIDTH=384
    HEIGHT=498
    CLASSID="CLSID:6D940280-9F11-11CE-83FD-02608C3EC08A">
</OBJECT>

<OBJECT
    ID="AdminCtrl"
    WIDTH=0
    HEIGHT=0
    ALIGN=CENTER
    CLASSID="CLSID:009541A0-3B81-101C-92F3-040224009C02">
</OBJECT>
</td>
</tr>
</table>
```


When the browser finishes loading the HTML code, the function defined in `window.onload` is called. Javascript within that function does some housekeeping and then the browser awaits some action from the user.

```

window.onload = init;
.
.
.

function init()
// Housekeeping
{

// Make a call to unlock critical features of the controls.
// Enter the filename of your license file if there is one. Otherwise,
// The 14 day evaluation period is in effect.
AdminCtrl.LicenseCheck ("C:\\Program Files\\Common Files\\Global 360\\Imaging\\
");

// Tell the Admin control not to throw an error if the operator cancels
// from the File Open dialog
AdminCtrl.CancelError = 0;

// Get rid of all scrollbars
PDFCtrl.showscrollbars = 0;
EditCtrl.Scrollbars = 0;

// Set the edit control to display black & white pages as grayscale.
EditCtrl.DisplayScaleAlgorithm = 4;
}

```

Loading an Image

To load a document, click the **Open** button. The onclick event of that button runs the `open()` javascript function, which displays the File Open dialog box from the Image Admin control.

Once the user has selected either an image file or a PDF file, the function routes the filename to the proper display control. The Open button can be used repeatedly to display both an image file and a PDF document at the same time.

```
<input type="button" value="Open" style="width:90px" title="Open"
  onclick="Open()">
.
.
.
Function Open()
{

// Put this in a try-catch structure in case user picks an invalid file
try
// Show the Open File dialog so user can browse for and pick a file
{AdminCtrl.ShowFileDialog (0);}

// If user chooses an unsupported file type catch the error, display the
// error description and exit the function
catch (e)
{alert(e.description);
return;
}

// Check to see if user selected an image file.
if (AdminCtrl.Image.length > 0)
{
// Yes, it's an image. Display it in the Edit control
EditCtrl.Image = AdminCtrl.Image;
EditCtrl.Display();
return;
}

// Not an image. Maybe a PDF?
if (AdminCtrl.PDFFile.length > 0)
{

// Yes, it's a PDF. Display it in the PDF control
PDFCtrl.Open(AdminCtrl.PDFFile);
}
}
```

Invoking the FitTo method

Once pages are displayed, clicking the BestFit button invokes the Fit() javascript function from the onclick event. This code simply calls methods on the two display controls to scale for best fit within the client area.

```
<input type="button" value="Best Fit" style="width:90px" title="Best Fit"
      onclick="Fit()">
.
.
.
function Fit()
{
PDFCtrl.ZoomBestFit();
EditCtrl.FitTo(0);
}
```

Visiting Other Pages

The Prev Page and Next Page buttons call functions that set display control properties so other pages can be seen while not exceeding the last page or attempting to display page 0.

The PDF control simply displays the new page after its `CurrentPage` property is updated while the Edit control needs to have its `Display` method called after the `Page` property is changed.

```
<input type="button" value="Prev Page" style="width:90px" title="Prev Page"
onclick="prevpage()">

<input type="button" value="Next Page" style="width:90px" title="Next Page"
onclick="nextpage()">
.
.
.
function nextpage()
{
if (PDFCtrl.CurrentPage < PDFCtrl.PageCount)
{
PDFCtrl.CurrentPage = PDFCtrl.CurrentPage + 1;
}
if (EditCtrl.Page < EditCtrl.PageCount)
{
EditCtrl.Page = EditCtrl.Page + 1;
EditCtrl.Display();
}
}

function prevpage()
{
if (PDFCtrl.CurrentPage > 1)
{
PDFCtrl.CurrentPage = PDFCtrl.CurrentPage - 1;
}
if (EditCtrl.Page > 1)
{
EditCtrl.Page = EditCtrl.Page - 1;
EditCtrl.Display();
}
}
```

Rotating a Page

The Rotate Right button invokes a function that calls the appropriate display control methods for clockwise rotation. The Edit control has a RotateRight method that acts on the current page. The PDF control has a more general RotatePage method where the function needs to specify the page to be rotated (.CurrentPage) and the type of rotation (90).

```
<input type="button" value="Rotate Right" style="width:90px" title="Rotate  
Right" onclick="rotright()">  
.  
.  
.  
function rotright()  
{  
PDFCtrl.RotatePage(PDFCtrl.CurrentPage,90);  
EditCtrl.RotateRight();  
}
```

Persistence

An interesting difference between the Edit control and the PDF control can be observed if you rotate a page, go to another page, and return. The Edit control has “forgotten” the earlier rotation while the PDF control has not.

To retain the rotation of an image page the programmer would have to write each page to a temp file before going to another page or maintain an array of page rotation values. The PDF control tracks this internally.

Compatibility with Microsoft .NET

This chapter describes Imaging for Windows® compatibility with Microsoft® .NET™.

In this Chapter

Introduction.....	166
Software Development Tips	167

Introduction

Imaging for Windows is compatible with Microsoft .NET Framework 2.0 and greater. It provides the .NET client interface to Imaging for Windows ActiveX Controls. Its design resembles the Imaging for Windows interface while providing .NET compatibility.

Imaging for Windows compatibility with Microsoft .NET adheres to the requirements of the .NET Common Language Specification (CLS); that is, it is CLS-compliant. Therefore, it should work in any language in the Common Language Runtime (CLR).

Package Contents

The Imaging for Windows compatibility with Microsoft .NET software package includes the following Primary Interop Assembly files:

- `ImgEdit.Interop.dll`
- `ImgAdmin.Interop.dll`
- `ImgThumb.Interop.dll`
- `ImgScan.Interop.dll`
- `ImgOCR.Interop.dll`
- `ImgPDF.Interop.dll`
- `Imgs vacc.Interop.dll` (for Server Access COM object)

Software Development Tips

If you are already familiar with the Imaging OCXs and have developed applications outside of the Microsoft .NET environment, you will notice a change in the default control names.

- In Visual Basic 6.0, the default reference to the Image Edit control is `ImgEdit1`.
- In Visual Basic .NET, the default reference to the Image Edit control is `AxImgEdit1`.

Several methods in the controls support optional parameters, such as `ImgEdit.SaveAs()`. Rules for using optional parameters differ between Visual Basic 6.0 and Visual Basic .NET.

- In Visual Basic 6.0, any or all optional parameters to a method can be omitted, generally in any combination, although there may be control-enforced restrictions depending on the method.
- In Visual Basic .NET, optional parameters to a method must be either all included or all omitted; no other combinations are supported. Therefore, it is not possible to supply some optional arguments to a method and omit others.

When using Visual Basic .NET in situations that require the specification of optional parameters not originally specified in the Visual Basic 6.0 code, the user can choose to either supply a specific value (typically the method's default value for that parameter) or use the `ErrorWrapper()` class, part of `System.Runtime.InteropServices`, to supply a `VT_ERROR`-like value that the control will treat as an “omitted” (default) parameter. For example:

```
Dim errParam As New _
    System.Runtime.InteropServices.ErrorWrapper(0)

frmMain.DefInstance.ImgEdit1.SaveAs(Img, 1,
    errParam, errParam, errParam, errParam)
```

To upgrade an existing Visual Basic 6.0 application that uses the Global 360 Imaging for Windows ActiveX controls to Visual Studio 2005, additional modifications may be required after running the Visual Studio upgrade wizard.

The following list describes some of the issues you may encounter:

- References to [FormName].[ControlName] need to be updated to [Formname].DefInstance.[ControlName]. For example, in Visual Basic 6.0, a reference to an image edit control on frmMain is: frmMain.ImgEdit1. In Visual Studio 2005, this needs to be changed to frmMain.DefInstance.AxImgEdit1. The upgrade wizard updates most, but not all, occurrences of this.
- Declarations using “As Any” are not supported and should be changed to appropriate types. For example, use “As String”.

A Visual Basic .NET coding example that uses the Imaging for Windows .NET compatible controls follows. This example displays a user-selected image file in an Image Edit control.

```
Public Class Form1
Private Sub DisplayImage_Click(ByVal sender As System.Object, _
    By Val e As System.EventArgs) Handles DisplayImage.Click
    'Use the ImgAdminOpen dialog box to display a file
    'in the Image Edit and Thumbnail Controls
    AxImgAdmin1.ShowFileDialog(Imgadmin.Interop.DlgOptionValue.OpenDlg)
    'Set the Image property in the Image Edit and Thumbnail controls
    AxImgEdit1.Image = AxImgAdmin1.Image
    AxImgThumbnail1.Image = AxImgAdmin1.Image
    'Set the page to 1 just in case
    AxImgEdit1.Page = 1
    'Scale the image page so the entire page appears in the window of the edit control
    AxImgEdit1.FitTo(Imgedit.Interop.FitToConstants.BEST_FIT)
    'Set the display algorithm to optimize for the page type
    AxImgEdit1.DisplayScaleAlgorithm = _
        Imgedit.Interop.DisplayScaleConstants.wiScaleOptimize
    'Finally display the image page
    AxImgEdit1.Display()
    'Put the focus on the thumbnail control so user can click on other pages
    AxImgThumbnail1.Focus()
End Sub

Private Sub AxImgThumbnail1_ClickEvent(ByVal sender As Object, ByVal e As _
    AxImgthumb.Interop._DImgThumbnailEvents_ClickEvent) Handles AxImgThumbnail1.ClickEvent
    'Display the image page that corresponds to the thumbnail clicked by the user
    AxImgEdit1.Page = e.thumbNumber
    AxImgEdit1.Display()
End Sub
End Class
```

The DestImageControl property of the Annotation and Scan controls must be set to the value of the ImageControl property of the target Edit control. This ImageControl property value is assigned automatically by the development system and continues to be ImgEdit1, ImgEdit2, and so forth, even though the control name on the form may be AxImgEdit1.

Some control events enable the Event Handler to influence post-event processing by assigning values to event arguments. Let's examine the PasteClip event of the ImgEdit control, which is fired when the data to be pasted will go beyond the current boundaries of the displayed image. Its prototype in Visual Basic 6.0 was

```
Private Sub ImgEdit1_PasteClip(Mode As Long)
```

but its .NET form is

```
Private Sub ImgEdit1_PasteClip(ByVal sender As Object, ByVal e As _  
    AxImgeditLibCtl._DImgEditEvents_PasteClipEvent) _  
    Handles ImgEdit1.PasteClip
```

e.mode is the variable available to the Event Handler (see the Object Browser for AxImgeditLibCtl).

Setting e.mode = vbYes (member of Microsoft.VisualBasic.Constants) causes the base image to be enlarged to accommodate the pasted data. vbNo pastes the data without changing the base image size, and vbCancel cancels the paste operation.

Imaging ActiveX Sample Applications

This appendix describes the Imaging ActiveX sample applications that are available on the media on which your software was distributed.

In This Appendix

Overview	172
Sample Applications	172

Overview

The Imaging ActiveX sample applications are relatively large Visual Basic projects that demonstrate how to use the Imaging ActiveX controls to build comprehensive and useful, image-enabled applications.

It is beyond the scope of this appendix to walk you through each and every application. Global 360 suggests that you run each one and analyze its code to determine whether you can use it:

- Directly in your applications, or
- As a guide to writing your own, related code.

Requirements

With the exception of the sample application, to compile and run the Imaging ActiveX sample applications, you must use:

- Microsoft Visual Basic 6.0
- Imaging for Windows[®]

To compile and run the sample application, you must use:

- Microsoft Visual Basic 6.0 with Service Pack 3 or later
- Imaging for Windows

Sample Applications

The code in each sample application is highly organized, commented, and written using Hungarian notation. There are eight sample applications in the following categories:

- Image Editor samples
- Function Specific samples
- Imaging Flow samples

The following sections describe them.

Image Editor Samples

This section describes the Image Editor sample applications.

Sample Application



The file name for the sample project is `ImagSamp.vbp`.

The sample application emulates the look and feel of the Imaging for Windows application. It is a baseline image editor that — due to its simplicity — is the best one from which to study and learn.

Sample functions include:

Delete Pages — Deletes selected pages from a displayed image document file. Note that looping through the image pages is performed from last to first to prevent the renumbering of image pages as they are deleted. See the code within the `mnuEditActionItem_Click` event procedure.

Drag Hand — Emulates the Drag Hand behavior evident in the Imaging for Windows application. The drag hand enables you to pan an image page; that is, to scroll the image page horizontally and vertically without using the scroll bars. See the code within the `ImgEdit1_MouseMove` event procedure.

Splitter Bar — Emulates the splitter bar behavior evident in the Imaging for Windows application. Note that the splitter bar (`imgDivider`) has a `Top` value of `-20000` and a `Height` value of `40000` to prevent the top and bottom of the splitter bar from being visible as you drag it. See the code within the `ImgEdit1_DragDrop` and `ImgThumbnail1_DragDrop` event procedures.



The file name for the Image Editor project is
`ImgEditr.vbp`.

Image Editor

Image Editor is a more sophisticated application that emulates the look and feel of the Imaging for Windows application.

After you master the sample application, investigate Image Editor, which includes toolbars and advanced features, such as:

- Contact sheet creation
- Image enhancement
- Magnification
- Optical Character Recognition (OCR)
- Summary properties (TIFF image document files only)

Function Specific Samples

This section describes the Function Specific sample applications.

All Function Specific applications have a common menu template. Because the menus are common, you can ignore the “standard” code and concentrate on the unique features of each application. The common menu selections include:

- **File**
- **“Generic”**
- **Page**
- **Zoom**
- **Annotation**
- **Tools**

The **“Generic”** menu provides access to functions that are specific to each application. Its caption changes appropriately within each one.

The **Tools** menu of the Image Scan and Image Thumbnails sample applications provides access to an interesting facility called the Event Tracker.

The Event Tracker lets you track the events fired by any of the Imaging ActiveX controls.

The tracker consists of two functions:

Track [control name] Events — Lets you select the Imaging ActiveX control events you want to track.

Show Event Log — Lists the selected Imaging ActiveX control events as they fire. You can view the events and their associated parameter values within a dialog box or in a hard-copy report.

Image Print



The file name for the Image Print project is
ImgPrint.vbp.

Image Print shows you how to print image document files from the standard **Print** dialog box, as well as programmatically from a custom **Print Settings** dialog box (frmSettings).

The application has two functions that the Imaging for Windows application doesn't have:

Page with Header — Prints an image with a header at the top. The header is created by programmatically generating an annotation and then shifting the image down so that it begins below the header. You can find the code that performs this task in the `GenerateHeaderWorkFile` procedure of `frmMain`.

Displayed Portion — When you zoom in on an image in the display window, this function prints only the portion of the image that is displayed. You can find the code that performs this task in the `GenerateDisplayedPortionWorkFile` procedure of `frmMain`.

You can select the **Page with Header** and **Displayed Portion** functions from the **Area to Print** frame on the **Print Settings** dialog box. They are functional only when you invoke printing via the **Print via Program Control** option on the **Print** menu.



The file name for the Image Properties project is `ImgProp.vbp`.

Image Properties

Image Properties shows you how to display and print the general, summary, and page property values of image files.

It uses the standard dialog boxes provided by the controls, as well as a few custom dialog boxes, to display the property values.

You can display and print the properties of:

- A single image file.
- All of the image files contained within a single folder.
- All of the image files contained within a folder and all of its subfolders.

In addition to the general, summary, and page property values, the application displays additional information, such as the:

- Total number of bytes the files consume.
- Minimum/maximum file size.
- Average (mean) file size.
- Total number of pages.
- Smallest and largest page size.
- Total number of annotations by type and group.

Image Scan



The file name for the Image Scan project is `ImgScan.vbp`.

Image Scan demonstrates a variety of scanning functions.

In addition to using the standard scanning dialog boxes provided by the controls, Image Scan also shows you how to get and set scanner capabilities programmatically using a series of custom dialog boxes.

The application also demonstrates how to scan double-sided originals on a simplex scanner and how to collate the pages into the correct order. This feature makes a simplex scanner function like a duplex scanner.



The file name for the Image
Thumbnails project is
`ImgThumb.vbp`.

Image Thumbnails

Image Thumbnails shows you how to manage and manipulate the individual thumbnail images within an Image Thumbnail control.

Specifically, the application shows you how to:

- Display thumbnails.
- Drag and drop thumbnails.
- Change thumbnail format and size.

In addition to displaying as thumbnail images the pages of a single image file, the program can also display as thumbnail images the first page of every image file within a specified folder.

Imaging Flow Samples

This section describes the sample applications designed to work with Imaging Flow.

Flow Program



The file name for the Flow
Program project is
`FlowPgm.vbp`.

The Flow Program demonstrates how a third-party program can be invoked from within a flow and how it may be used to control — or affect — the current flow.

The program has two operating modes:

Separator Page mode — Locates separator pages so it can assemble scanned pages into discreet, single- or multi-page image document files.

Form Number mode — Reads form numbers by performing zoned OCR on images. And then uses the OCR results to name the image document files.

If you invoke the program with command line arguments from the process version of the Run Program tool, it functions in the background. Two command line arguments are available; the one you pass selects the operating mode of the program:

`/separatorpage` — Places the program in Separator Page mode.

`/formnumber` — Places the program in Form Number mode.

If you invoke the program without command line arguments, it assumes that you want to change its settings. Accordingly, it provides a user interface for doing so.

Note: The **OCR** function within Flow Program is an excellent example of using OCR text zones to perform targeted OCR processing.

Refer to the Imaging Flow online help system for more information on the Run Program flow tool (Process version).

Flow Variables



The file name for the Flow Variables project is `FlowVar.vbp`.

The Flow Variables program also demonstrates how a third-party program can be invoked from within a flow. The program shows you how to monitor and set flow variables.

This program is designed as a diagnostic tool only; however, you may get some interesting ideas from analyzing it.

Refer to the “Flow Variables Reference” within the Imaging Flow online help system to learn more about flow variables.

Imaging ActiveX Tips and Tricks

This appendix describes some useful tips and tricks for working with the Imaging ActiveX controls.

In This Appendix

Overview	180
Miscellaneous Programming Tips	180
Image File Management Tips	184
Annotation Tips	188
Optical Character Recognition Tips	190

Overview

Use the tips and tricks in this section as guidelines when you use the Imaging ActiveX controls to image-enable your applications. Refer to the remainder of this guide for more information about the controls.

Miscellaneous Programming Tips

Specify *tenths* of degrees when calling rotation methods

The **RotateAll**, **RotateLeft**, and **RotateRight** methods of the Image Edit control permit you to specify the degree of image rotation to apply. When you do so, keep in mind that you must specify the rotation amount in *tenths* of degrees.

For example, to rotate an image page 45 degrees to the right, specify 450 when you invoke the **RotateRight** method.

Use the **DisplayScaleAlgorithm** property to scale black-and-white image pages to gray

Global 360 recommends that you set the **DisplayScaleAlgorithm** property of the Image Edit control to `wiScaleOptimize` (literal 4) when you want to make black-and-white images appear in gray scale. Color images continue to appear in color using this setting.

Catch errors properly when working with the **ShowFileDialog** method

When invoked, the **ShowFileDialog** method of the Image Admin control displays an **Open** or **Save As** dialog box to your end users. When users click the **Cancel** button on one of these dialog boxes, a “Cancel button is pressed” error condition occurs. Other Image Admin error conditions can also occur as the procedure containing the **ShowFileDialog** method continues its processing.

It is important to understand the difference between catching the “Cancel button is pressed” error condition and any other Image Admin error condition that may occur.

When users click the **Cancel** button on the **Open** or **Save As** dialog box, the **Number** property of Visual Basic’s **Err** object contains the literal value for the “Cancel button is pressed” error condition. The **Description** property of the **Err** object contains any other Image Admin error condition that may have occurred.

The following code snippet from the sample application demonstrates the recommended way of handling **ShowFileDialog** and Image Admin error conditions:

```
'-----  
' If the Cancel button was pressed, exit the subroutine.  
' If a different error occurred, declare a message box and  
' exit the subroutine.  
'-----  
If Err.Number = CANCEL_PRESSED Then      '32755 = Cancel pressed  
    Exit Sub  
ElseIf ImgAdmin1.StatusCode <> 0 Then  
    MsgBox Err.Description & " (ImgAdmin error " & _  
        Hex(ImgAdmin1.StatusCode) & ")", vbCritical  
    Exit Sub  
End If
```

Always pass the parent window handle when invoking the ShowPrintDialog method

Even though passing the handle to the parent window is optional, for best results *always* include it when you invoke the **ShowPrintDialog** method of the Image Admin control. The **ShowPrintDialog** method displays a **Print** dialog box, which enables users to print image files.

Retain generated file names when template scanning

You can assign a template to the **Image** property of the Image Admin control to perform template scanning. The Imaging software uses the template you assign to automatically generate the file names for each document it scans.

Unfortunately, if you need to know the name of each generated file, you cannot use the **Image** property to return the file names when template scanning.

However, you can use the following alternative procedure to *retain* each file name while template scanning. Perform the following steps:

- 1 Use the **GetUniqueName** method of the Image Admin control to generate a unique file name.
- 2 Assign the generated file name to the **Image** property of the Image Scan control, which is a prerequisite for scanning.
- 3 Assign the generated file name to a local or module variable so you can retain it for your use.
- 4 Invoke scanning manually.

Clear a selection rectangle

To clear a selection rectangle, use the **DrawSelectionRect** method of the Image Edit control to draw a small, one-pixel selection rectangle on the Image Edit control.

The resulting selection rectangle is too small to be seen and clears the original one from the display.

Prevent flicker when using the Image Edit and Image Thumbnail controls simultaneously

To prevent flicker when using the Image Edit and Image Thumbnail controls simultaneously in your program, set the **ImagePalette** property of the Image Edit control to the appropriate setting.

The setting you use depends on the page type of the image being displayed in the Image Edit control, as follows:

When the page type of the displayed image is RGB (24-bit)

— Set the **ImagePalette** property to the RGB24 palette by entering a constant value of `wiPaletteRGB24` or a literal value of 3.

When the page type of the displayed image is *not* RGB —

Set the **ImagePalette** property to the Common palette by entering a constant value of `wiPaletteCommon` or a literal value of 1.

Image File Management Tips

Provide file type and page property options to your users

When saved to disk, image files can require a large amount of storage space.

The size of an image file depends on several factors; among these are its:

- File type.
- Color type (also known as its page type or data type).
- Resolution.
- Compression.

Giving your users the capability of changing these factors enables them to control file size and appearance.

The table on the following page lists the results of varying these factors on a one-page newsletter. The newsletter consists of:

- An image that occupies approximately one-quarter of the page.
- Three columns of text that fill the remainder of the page.

Use the table as a guideline when providing file type and page property options to your end users. You may want to include a similar table in your documentation.

Results of Varying Image File Type and Page Property Options

File Type	Color Type	Compression Applied	Resolution 100 x 100	Resolution 200 x 200	Resolution 300 x 300
BMP	Black & White	Not available	114 KB	448 KB	1.0 MB
	Pallettized 8-bit	Not available	898 KB	3.5 MB	7.9 MB
	BGR 24-bit	Not available	2.6 MB	10.5 MB	23.6 MB
TIFF	Black & White	None	112 KB	449 KB	1.0 MB
		Group3 (1d)	68 KB	134 KB	226 KB
		Group3 Mod. Huffman	67 KB	132 KB	223 KB
		Group4 (2d)	68 KB	92 KB	115 KB
		PackBits	59 KB	193 KB	389 KB
	Gray Scale 4-bit	None	449 KB	1.8 MB	3.9 MB
		LZW	132 KB	261 KB	579 KB
	Gray Scale 8-bit	None	898 KB	3.5 MB	7.9 MB
		LZW	585 KB	1.5 MB	2.5 MB
		JPEG — Medium ¹	128 KB	367 KB	703 KB
	Pallettized 8-bit	None	899 KB	3.5 MB	7.9 MB
		LZW	137 KB	367 KB	670 KB
	RGB 24-bit	None	2.6 MB	10.5 MB	23.6 MB
		LZW	897 KB	2.2 MB	3.9 MB
		JPEG — Medium ¹	144 KB	448 KB	907 KB

¹By adjusting JPEG resolution and quality compression options, you can increase or decrease the file size by 20 to 30%. In our newsletter example, the JPEG compression option applied was medium resolution and medium quality. The image file would be larger if we applied high resolution and high quality and smaller if we applied low resolution and low quality.

Several properties and methods within the Imaging ActiveX controls permit you to manage image file type and page property options.

Pay particular attention to the **SaveAs**, **SavePage**, and **ShowPageProperties** methods of the Image Edit control because they provide the quickest and easiest ways to provide file type and page property options to your users.

Use the Append method to assemble several image files into one multipage TIFF image file

You can use the **Append** method of the Image Admin control to copy image pages from one image file to another.

The following Visual Basic statements copy the first two pages of the `source.tif` file to the `destination.tif` file:

```
ImgAdmin1.Image = "c:\images\destination.tif"  
ImgAdmin1.Append "c:\images\source.tif", 1, 2
```

The copied pages become the last two pages of the `destination.tif` file.

Always clear a displayed image page before deleting it

You must clear an image page from the Image Edit control before you delete it. Failure to do so results in a run-time error.

The following code snippet from the sample program demonstrates the recommended way of deleting an image page:

```
'-----
' Delete: Loop through the image pages, deleting
' those pages which have been selected.
'-----
For intCounter = ImgThumbnaill1.ThumbCount To 1 Step -1
  If ImgThumbnaill1.ThumbSelected(intCounter) = True Then
    lngDeletedPageNo = intCounter
    ImgEdit1.ClearDisplay
    ImgAdmin1.DeletePages lngDeletedPageNo, 1
    ImgThumbnaill1.DeleteThumbs lngDeletedPageNo, 1
  End If
Next intCounter
```

Use caution when copying *selected* image data to the Clipboard

When copying the image data within a selection rectangle to the Clipboard, do not pass the parameters received from the **SelectionRectDrawn** event to the **ClipboardCopy** method. If you do, the **ClipboardCopy** method may not copy the expected image area.

Although the **SelectionRectDrawn** event and the **ClipboardCopy** method both use Left, Top, Width, and Height parameters, they each work with different base locations. The **SelectionRectDrawn** event uses the image pixel position relative to the upper-left corner of the Image Edit control, while the **ClipboardCopy** property uses the absolute image pixel position.

When you want to copy the data in a selection rectangle, invoke the **ClipboardCopy** method with its Left, Top, Width, and Height parameters empty.

Annotation Tips

Let your users modify the properties of a drawn annotation

After drawing an annotation, your users may want to change its properties. For example, they may want to change a line color from red to blue.

To provide this capability, invoke the **ShowAttribsDialog** method of the Image Edit control. It displays an annotation attributes dialog box that lets your end users change the properties (attributes) of a selected annotation.

For text annotations, invoke the **EditSelectedAnnotationText** method of the Image Edit control. It enables end users to modify the text.

Guidelines for making annotations permanent

You can use the **BurnInAnnotations** method of the Image Edit control to make annotations a permanent part of the image. When you or your users burn-in annotations, keep the following points in mind:

- Once annotations are burned-in, they cannot be removed or modified *as annotations*. They can, however, be edited or manipulated just like image data — clearing as well as copying and cutting to the Clipboard is available.
- **BurnInAnnotations** works only on the currently displayed image page. If you or your users want to burn-in the annotations of an entire image file, you must apply the **BurnInAnnotations** method to each individual page.
- Annotation data can be saved separately from the image data only when you or your users save image files in the TIFF file format. For all other file formats, the annotations must be burned-in prior to saving.

How to retain annotations when users navigate multipage image files

When users draw annotations on a page in a multipage image file and then navigate to another page in the file, the Imaging software does not automatically retain the annotations drawn. When users return to the “annotated” image page, the annotations are no longer there.

To retain annotations under these circumstances, create a temporary image file and use it as a work file. When users draw annotations on a page, invoke the **Save** method of the Image Edit control to save the entire image file to the work file. This action forces the Imaging software to retain the annotations because they have been saved to a file.

Then, when users close the image file or exit your application, prompt them to indicate whether they want to save any changes.

If users respond with Yes — Copy the work file to the original image file and then delete the work file.

If users respond with No — Simply delete the work file.

When printing annotations

Keep in mind that some printers may not print annotations correctly, especially Image Embedded and Image Reference annotations.

The reason for this behavior is that printer drivers function differently, making it impossible to print annotations consistently on all printers. The solution to this problem is to have your users:

- 1 Burn the annotations onto the image page.
- 2 Save the image file.
- 3 Print the image file.

If users do not want to permanently alter an image, have them save the image to a temporary file before performing the preceding steps.

Optical Character Recognition Tips

Working with Interactive Training

Keep the following points in mind when working with Interactive Training:

- When you assign a training file to the **TrainingFile** property of the Image OCR control, make sure that the training file actually exists. If it does not, training does not occur.
- Interactive Training is not available when performing OCR to the Clipboard.
- If you or your users perform Interactive Training on a skewed image, the yellow highlights that indicate each questionable word may not appear directly over the appropriate word. You or your users should invoke the **AutoDeskew** or **ManualDeSkew** method of the Image Edit control to straighten a skewed image and then save the image prior to performing Interactive Training.

Performing OCR

Keep the following points in mind when performing OCR processing:

- Do not attempt to OCR a severely skewed image. You or your users should invoke the **AutoDeskew** or **ManualDeSkew** method of the Image Edit control and then save the file prior to performing OCR. You or your users must save the image file because the OCR engine reads from the file and not from the display buffer.
- Communicate to your users that OCR processing occurs more efficiently when there is less blank space on an image. For example, if users want to OCR a 3x5-inch card, you should instruct them to scan the card as a 3x5-inch image instead of, for example, a letter-size image. In addition to saving disk space, scanning a 3x5-inch card as a 3x5-inch image makes OCR processing occur more efficiently because the OCR engine traverses less blank space as it looks for characters to convert.

- The OCR engine performs its processing on a black-and-white image. If your users are receiving poor OCR results from a page that looks good, provide a way for them to turn off scale-to-gray. With scale-to-gray turned off, users see what the OCR engine is actually analyzing and may come to the realization that rescanning or enhancing the image is necessary.
- The Imaging software can perform OCR processing in three ways. Providing your users with the appropriate OCR functions may improve OCR results:

When users want to OCR a small area of text — Set the **CopyToClipboard** property of the Image OCR control to **True**. Then have your users draw a selection rectangle and invoke OCR. The OCR engine recognizes only the area your users indicate rather than the entire page.

When users want to OCR most of the page — Set the **CopyToClipboard** property of the Image OCR control to **False**. Then set the **AnnotationType** property of the Image Edit control to **wiOcrRegion** (literal 13) and the **AnnotationOcrType** property to **wiOcrTypeText** (literal 0). Have your users draw OCR text zones over the areas of the page they want to recognize and invoke OCR. Again, the OCR engine recognizes only those areas your users indicate.

If users want to include graphics in the OCR results, set the **AnnotationOcrType** property to **wiOcrTypePicture** (literal 1). Then have your users draw OCR picture zones over the graphics they want to include.

To OCR the entire page — Set the **CopyToClipboard** property of the Image OCR control to **False** and invoke OCR. Performing OCR on the entire page is the default.

- Keep in mind that setting the **OutputFile** property of the Image OCR control to blank does not invoke the **Save As** dialog box. To have the Imaging software prompt the user to specify where the OCR results should be saved, simply omit the **OutputFile** property from your code.

Adding Imaging Using Automation

This appendix explains when and how to use Automation to image-enable your applications. It describes the object hierarchy of the Imaging application and 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.

Note: To use Automation, you must have access to the Imaging for Windows® Viewer.

In This Chapter

Overview	194
Imaging Components	195
Invoking Imaging for Windows	199
When to Use Automation	202
The Object Hierarchy	203
Imaging Application Modes	205
Demonstration Project	212

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: Appendix D in this guide describes the properties and methods of each object.

The Automation demonstration project described later in this appendix 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.

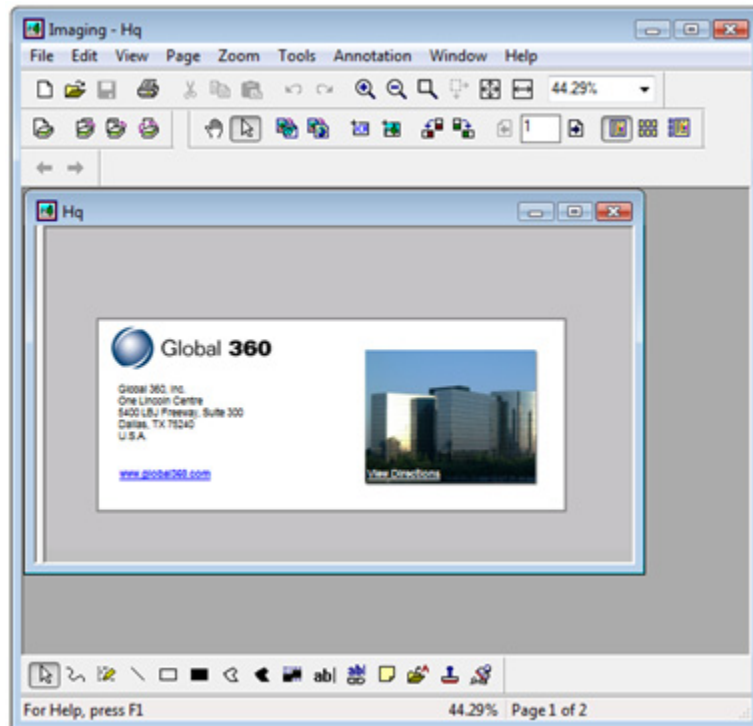
Imaging Components

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.

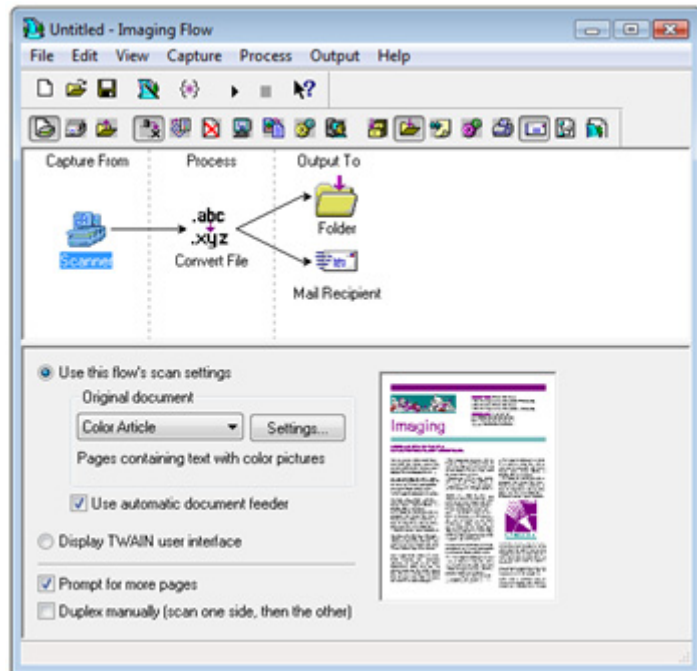
Imaging Application

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
- ActiveX controls

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, unlike using the Imaging ActiveX controls, 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.

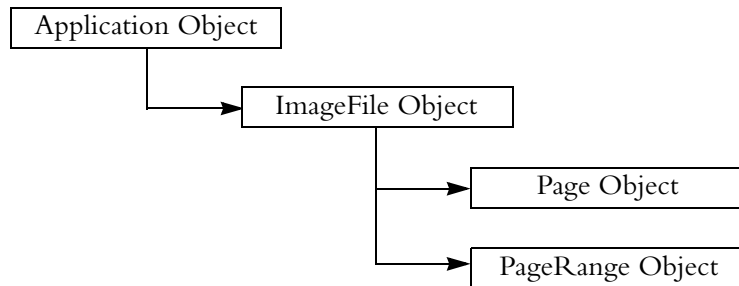
At flow design time, the author can set Review tool options that manipulate the Imaging application as well as the image it displays. These options include:

- 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 Appendix D 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: Appendix D 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



Developers using the Imaging ActiveX controls can use the Image Edit and Image Thumbnail controls to simulate the View Mode behavior described in this section. Refer to the ActiveX controls Help for more information.

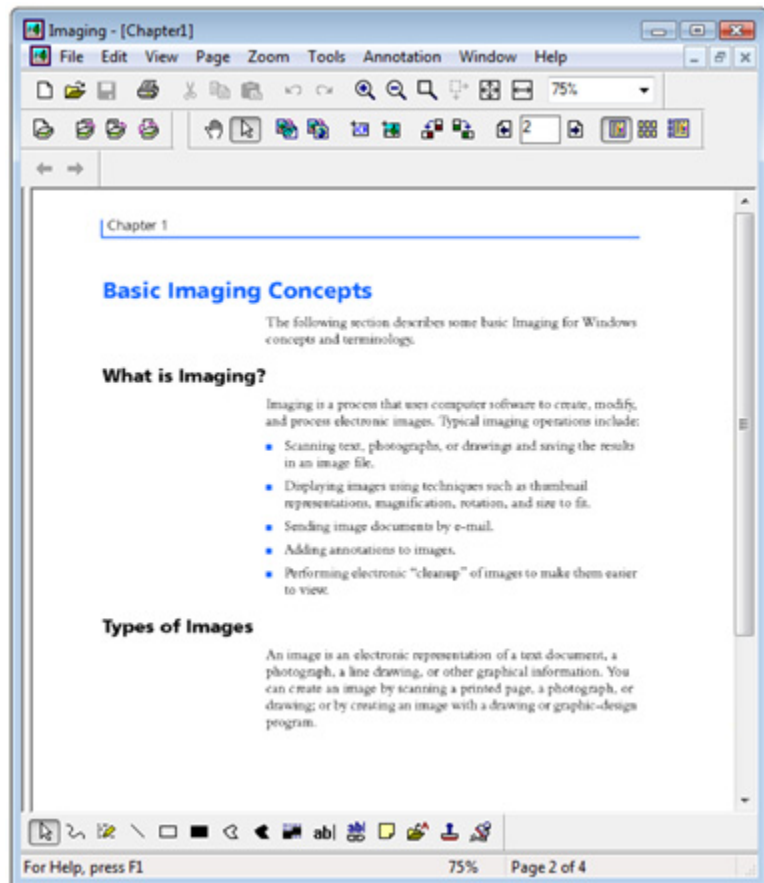
The Imaging application has three view modes that enable users to view and work with image files. Each view mode has its own set of advantages and capabilities.

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.

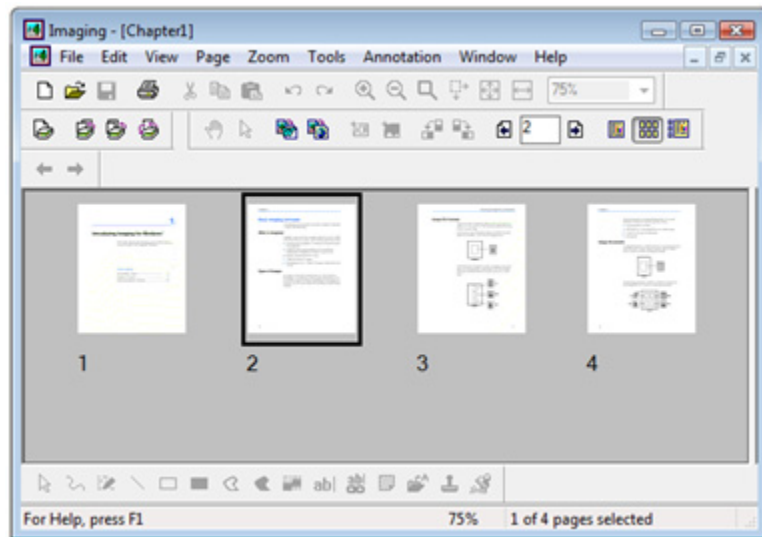


Thumbnail

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.

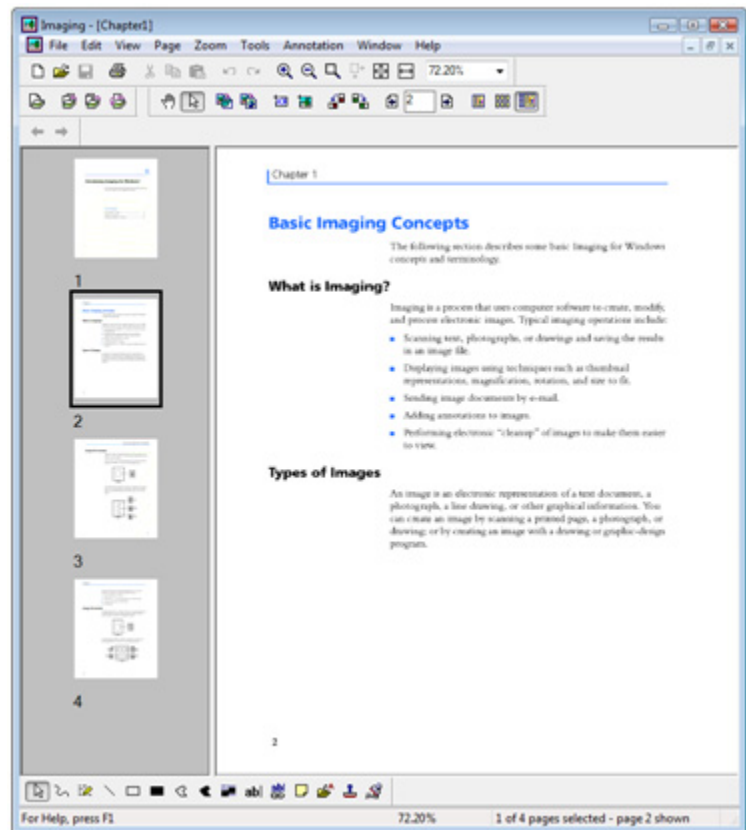
Keep in mind that some Imaging functions — such as annotation and zoom — are not available in this mode because they are not appropriate for use on such small images.



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.

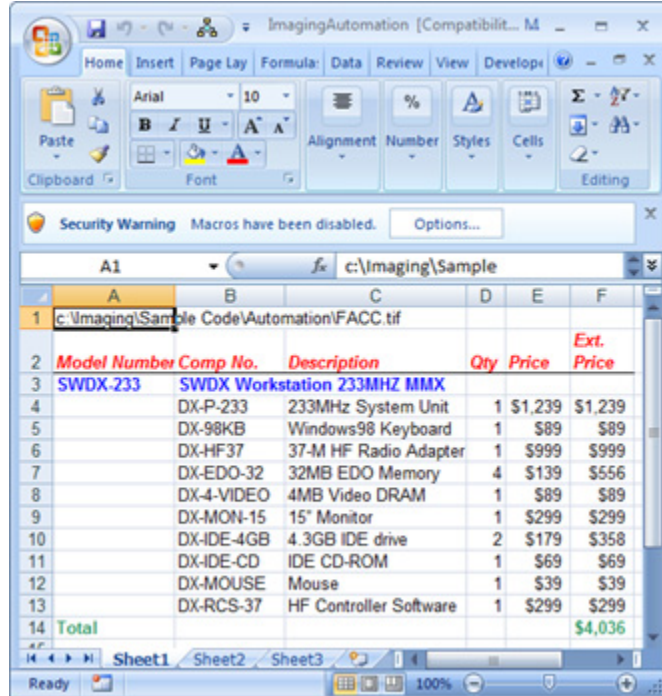
RotateLeft method (Page object) — Rotates the image 90 degrees counterclockwise.

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.

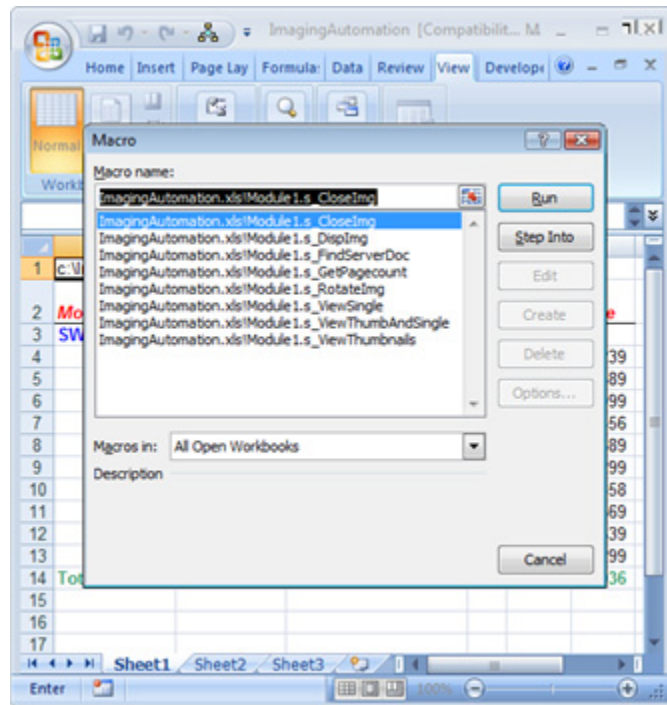


Model Number	Comp No.	Description	Qty	Price	Ext. Price
SWDX-233		SWDX Workstation 233MHZ MMX			
		DX-P-233 233MHz System Unit	1	\$1,239	\$1,239
		DX-98KB Windows98 Keyboard	1	\$89	\$89
		DX-HF37 37-M HF Radio Adapter	1	\$999	\$999
		DX-EDO-32 32MB EDO Memory	4	\$139	\$556
		DX-4-VIDEO 4MB Video DRAM	1	\$89	\$89
		DX-MON-15 15" Monitor	1	\$299	\$299
		DX-IDE-4GB 4.3GB IDE drive	2	\$179	\$358
		DX-IDE-CD IDE CD-ROM	1	\$69	\$69
		DX-MOUSE Mouse	1	\$39	\$39
		DX-RCS-37 HF Controller Software	1	\$299	\$299
Total					\$4,036

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_Displmg()** subroutine.

The **_Displmg()** 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
```


Automation Lexicon

This appendix describes the properties and methods of each Imaging for Windows[®] Automation object.

In this Chapter

Overview	230
Application Object.....	230
ImageFile Object	243
Page Object.....	258
PageRange Object	266

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 Appendix C 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.

Application Object Properties (cont.)

Property	Description
DisplayScaleAlgorithm	Sets or returns the scaling algorithm used for displaying images.
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.

Application Object Properties (cont.)

Property	Description
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 availability 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.
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.

ImageFile Object Methods (cont.)

Method	Description
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.
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
    ➤ "Image://nqa11\SYS:\tmp\3PAGES.tif", 1, 3

'This example appends pages to an Imaging Server 1.x document.
ImgFileObj.AppendExistingPages
    ➤ "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.
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.
    ➡ ImgFileObj.InsertExistingPages
      "Image://nqa11\SYS:\tmp\3PAGES.tif", 2, 3, 1

'This example inserts pages in an Imaging Server 1.x document.
ImgFileObj.InsertExistingPages
    ➡ "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 =
    ➡ 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.

 'This example returns this page's vertical scroll position.
 ypos =

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 =

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 Start-Page .

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`

