Eastman Software
Imaging

Professional Developer's Guide

715-B017

Eastman Software, Inc.
Tel. (978) 967-8000 ■ http://www.kodak.com and www.eastmansoftware.com

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

Use, duplication, or disclosure by the Government of this manual/documentation is subject to the restrictions and protections under paragraph (b)(2) or DoD FAR 252.227-7015, Technical Data-Commercial Items (Nov. 1995) or, as applicable, FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987). Use, duplication or disclosure by the Government of this Software is subject to the restrictions and protections set forth in Eastman Software, Inc.'s standard commercial license for such Software as prescribed in DoD FAR 227.7202-4 or, as applicable, FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

Important Notice Concerning the Use of LZW Compression

This product uses LZW, a compression/decompression technology that is covered by U.S. Patent 4,558,302 and its foreign counterparts, issued or pending, held by the Unisys Corporation. Eastman Software's grant of license from Unisys does not permit you to use the LZW compression or decompression capabilities from any version of Imaging for Windows for development purposes, or to use or derive the LZW capabilities from Eastman Software third party application or derivative software. Please contact Unisys for licensing information at:
Unisys Corporation
Welch Licensing Dept. - MSC1SW19
Township Line and Union Meeting Roads
P.O Box 500
Blue Bell, PA 19424-0001
Fax: (215) 986-3090
or
www.unisys.com

Contributers to this book include the following:

Project Leader
Jim Preftakes

Project Manager
Mark Sartanowicz

Lead Technical Writer
Bob Raymond

Contributing Technical Writers
Steve O'Neill, Leonard Turmel

Technical Editors
Charles Bann, Kathleen Sparr

Software Developers
Sue Cox, Guy Greese, Richard Guidoboni, Garry Sager, and many other talented members of the Imaging for Windows team

Graphic Designers
Patricia Israel, Maria Witt

Production Specialists
Jeanne Chandonnet, Denise Govoni

Contents

Professional Developer's Guide

4 Adding Imaging Using ActiveX Controls

11 Image Thumbnail Control

B Imaging ActiveX Tips and Tricks

1

Imaging In Development

This chapter introduces you to the *Eastman Software Imaging Professional Developer's Guide.*

The chapter begins by describing the guide and the Imaging for Windows product. It continues by explaining what Imaging for Windows lets you do and what it lets your users do. The chapter concludes by explaining the concepts of business and personal document imaging.

In This Chapter

Welcome to the Professional Developer's Guide

This section introduces you to the *Imaging Professional Developer's Guide*.

The *Eastman Software Imaging Professional Developer's Guide* provides software developers and MIS professionals with the information they need to understand, produce, and support image-enabled applications that run on Microsoft® Windows® 95, Windows 98, and Windows NT 4.0 and greater.

This guide is a technical resource that supplements the documentation included with the Imaging for Windows product. For instructions on how to use Imaging for Windows, refer to its on-line help system. You can access on-line help from the Help menu of any Imaging for Windows application.

For instructions on how to use the Imaging ActiveX controls, access the on-line help system for the controls from your development environment. (Refer to Chapters 4 and 6 in this guide for instructions.)

In addition to this printed book, the developers guide includes a compact disc (CD) that contains the following items:

- An updated Imaging ActiveX controls help system.
- Sample code that helps you image-enable your applications. (Refer to "Sample Code" later in this chapter for more information.)
- The *Imaging Professional Developer's Guide* in Portable Document Format (PDF).

Note: To view the developer's guide on-line, the Adobe® Acrobat Reader (Version 3.0 or greater) must be installed on your system. You can obtain the reader free of charge from Adobe Systems Incorporated, at:
`http://www.adobe.com`

Introducing Imaging for Windows

This section describes the Imaging for Windows product.

Eastman Software Imaging for Windows is a multifaceted product that lets users transform paper documents and faxes into electronic documents that can be viewed, annotated, edited, converted, printed, and shared.

There are several versions of Imaging for Windows. At the time of this writing, they are the following:

Imaging for Windows 95 — Created by Eastman Software, Inc. (then-Wang Software) for Microsoft Corporation, Imaging for Windows 95 ships as a standard component of the Windows 95 operating system (OSR2 release).

Note: If you or your users have a version of Windows 95 that is earlier than the OSR2 release, Imaging for Windows 95 is not included with the operating system.

To determine whether Imaging is installed, have your users check the Accessories menu on the PC. If Imaging does not appear on this menu, it must be downloaded from the Web and installed.

Imaging for Windows 95 is available free-of-charge from the Eastman Software Web site, at:

`http://www.eastmansoftware.com`

Imaging for Windows NT — Also created for Microsoft Corporation, Imaging for Windows NT ships as a standard component of the Windows NT operating system (Version 4.0 and greater).

Imaging for Windows 98 — Also created for Microsoft Corporation, Imaging for Windows 98 ships as a standard component of the Windows 98 operating system.

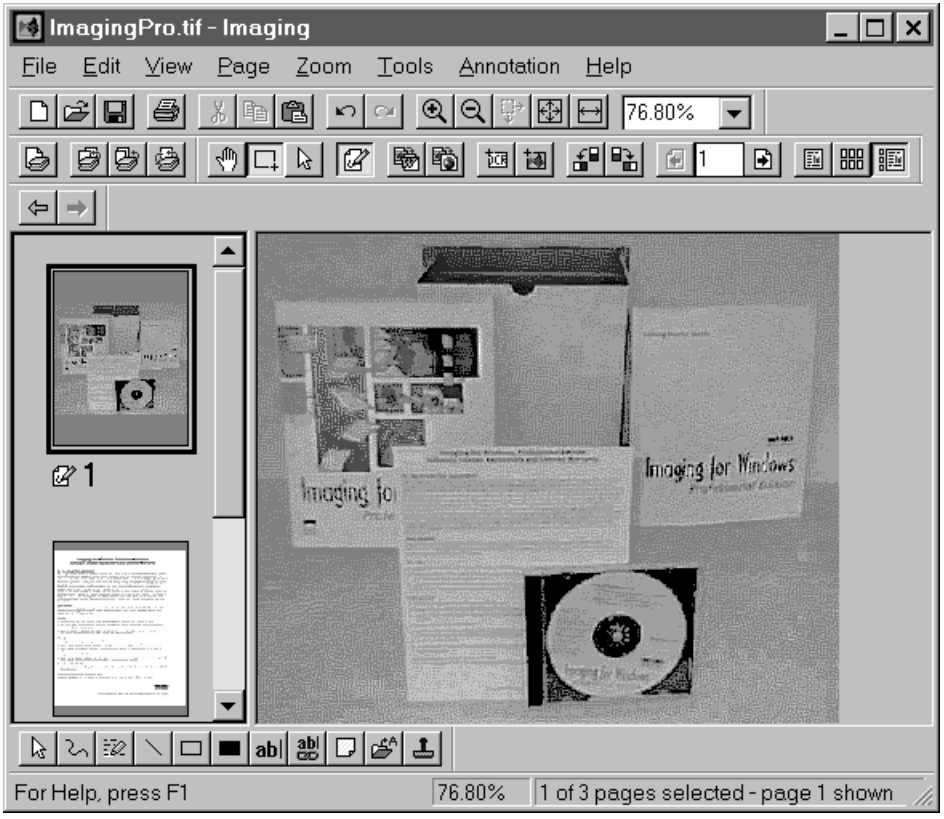
Imaging for Windows Professional Edition — Available for purchase from the Eastman Software Web site and other places, Imaging Professional provides increased capabilities over the standard version of the product. It runs on Windows 95, Windows 98, and Windows NT 4.0 and greater.

Imaging Components

Imaging for Windows has several components. The following sections briefly describe them.

Imaging Application

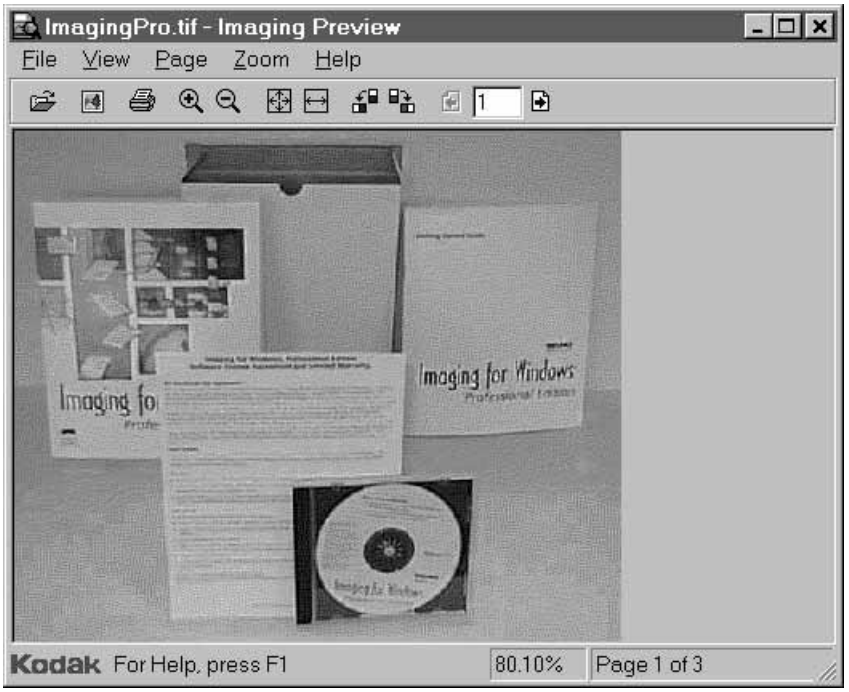
The Imaging application is the main component of Imaging for Windows. It enables users to scan, view, annotate, manipulate, store, and perform Optical Character Recognition (OCR) on faxes, paper documents, and electronic images. (OCR is available with Imaging Professional only.)



Imaging Preview

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

Imaging Preview is available with Imaging for Windows 98 and Imaging Professional only.



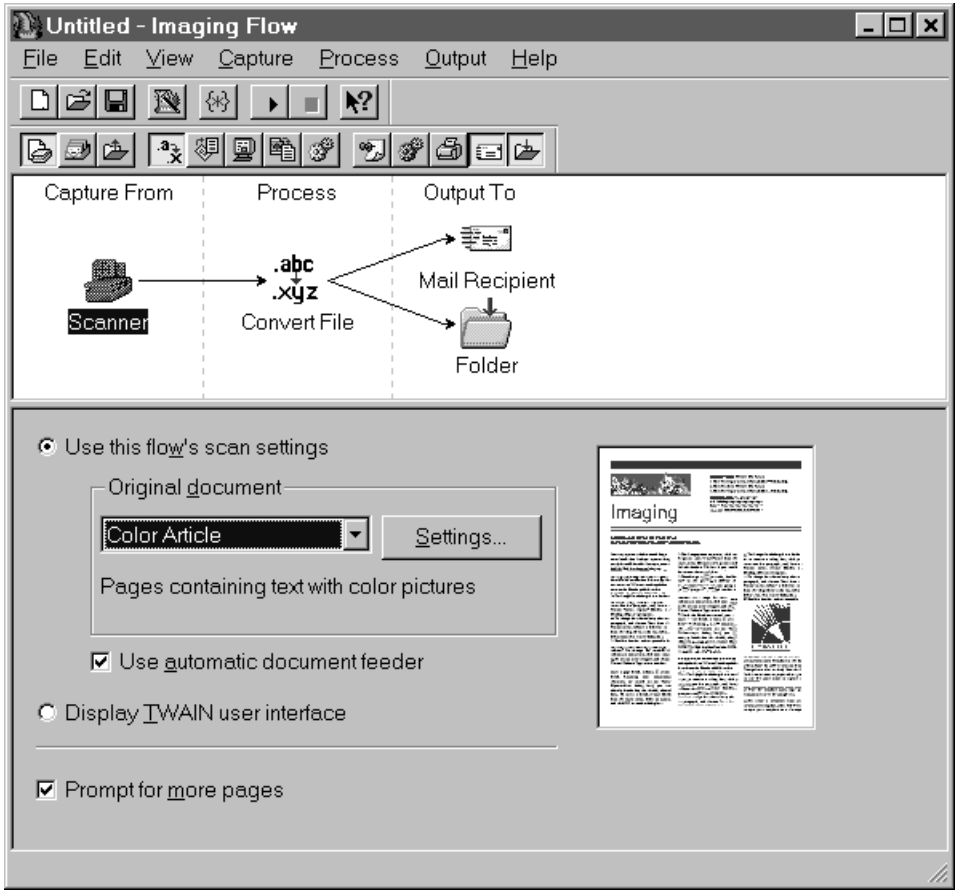
Imaging Flow

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.
- Output images by:
  - Posting them to Exchange folders.
  - Saving them to local or network folders.
  - Printing them.
  - Sending them to others via e-mail.

Imaging Flow is available with Imaging for Windows Professional Edition only.



Development Tools and Methods

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)
- Microsoft Visual J++
- Borland® Delphi
- Powersoft® PowerBuilder®

What Imaging Lets You Do

This section describes what Imaging for Windows lets developers do.

Imaging for Windows includes several development tools and methods that let you add Imaging functions to your 32-bit applications. The development tools and methods include:

- Command-line invocation
- OLE
- Automation
- ActiveX controls

The following sections describe them, help you determine which one to use, and briefly describe the sample code contained on the companion CD.

Command-line Invocation

You can invoke the Imaging application using its command line. 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. (Refer to the examples on page 13.) The following table lists the file names of the Imaging application by version.

Imaging for Windows File Names By Version

File Name	Product Version
wangimg.exe	Imaging for Windows 95
	Imaging for Windows NT 4.0
	Imaging for Windows Professional Edition V1.0 and 1.1
kodakimg.exe	Imaging for Windows NT 5.0 and greater
	Imaging for Windows 98
	Imaging for Windows Professional Edition V2.0 and greater

Note: The name Wang appears in the file name of early versions of the Imaging application because Wang Software, the software unit of Wang Laboratories, Inc., produced those versions.
The Eastman Kodak Company acquired Wang Software in 1997, creating Eastman Software, Inc.

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.

For example, you can use Visual Basic's OLE Container control to easily embed or link image files in your 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, which 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.

Automation

You can use Automation to image-enable your application.

Automation is a more powerful way to image-enable your application. It enables you to control the Imaging application programmatically from your application and provide your users with the capabilities contained within the Imaging application.

Note: You can also use Automation to control the Imaging application from other Automation-capable applications, such as Microsoft Word and Excel.
However, you cannot use Automation to control the Imaging Preview and Imaging Flow applications.

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

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

Note: Chapter 2 of this guide explains Imaging's implementation of Automation. Chapter 3 describes the properties and methods of each object.

ActiveX Controls

You can use ActiveX controls to image-enable your application.

Using ActiveX controls is another 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.

(Summary property functions are available with Imaging Professional only.)

Image Annotation Tool Button control — The Image Annotation Tool Button control lets you create customized annotation tool bars 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 provide 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.

(The Image OCR control is available with Imaging Professional only.)

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.

Note: Chapters 4 through 6 of this guide explain how to get started with the Imaging ActiveX controls and walk you through several sample programs.
Chapters 7 through 11 of this guide describe the properties, methods, and events of each Imaging ActiveX control. Every topic indicates the Imaging version that supports each property, method, and event.

Which to Use: Command-line Interface, OLE, Automation, or ActiveX Controls?

The following sections describe the major differences in using the command-line interface, OLE, Automation, or ActiveX controls to image-enable an application.

Command-line Interface

Command-line invocation is the most simple but least powerful way to implement Imaging functions in your application.

You can invoke the Imaging application using standard **Shell** functions, by including the executable file name of the Imaging application and the fully qualified file name of a supported image file.

For example, if you are developing under Imaging for Windows 95, Imaging for Windows NT 4.0, or Imaging for Windows Professional Edition V1.0 or V1.1, you can use the following statement to invoke the Imaging application and display an image file:

```
Shell("c:\Windows\Wangimg.exe c:\Quote.tif", 1)
```

If you are developing under Imaging for Windows 98 or Imaging for Windows Professional Edition V2.0 or greater, you can use the following statement to invoke the Imaging application and display an image file:

```
Shell("c:\Windows\Kodakimg.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

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 may be embedded or linked.

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.

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

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.

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 Professional Edition, 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.

ActiveX Controls

You can also use the Imaging ActiveX controls to add imaging functions to Web pages.

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

Example

The demonstration projects in Chapters 4 through 6 of this guide are excellent examples of using the Imaging ActiveX controls to image-enable applications.

Sample Code

The companion CD contains sample code designed to help you add Imaging functions to your applications.

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 97 and be familiar with Visual Basic for Applications (for the Automation demonstration project).
- Set up and be familiar with Visual Basic 5.0 or greater (for the ActiveX demonstration projects and the ActiveX sample applications).

Note: When you set up Visual Basic 5.0 or greater, you must perform a Typical installation. If you don't, the sample code supplied with the *Imaging Professional Developer's Guide* may not function correctly.

- Set up and be familiar with Visual J++ Version 1.1 or greater (for the ActiveX/Web demonstration project).

The following sections briefly describe the sample code on the CD.

Automation Demonstration Project

The Automation demonstration project shows you how to use Automation in Excel 97 to:

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

Chapter 2 of this guide walks you through the Automation project.

ActiveX Demonstration Projects

The ActiveX demonstration projects are small Visual Basic applications that 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 an interact with *Eastman Software* Imaging 1.x and Imaging 3.x servers.

Chapters 4 and 5 of this guide walks you through each demonstration project.

ActiveX/Web Demonstration Project

The ActiveX/Web demonstration project is a small Java applet that shows you how to:

- Display a multipage image file in a Web page.
- Rotate an image page.
- Navigate an image file.

Chapter 6 of this guide walks you through the demonstration project.

ActiveX Sample Applications

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

While walking you through each sample application is beyond the scope of this guide, Eastman Software, Inc. suggests that you 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 virtually mirrors 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.

What Imaging Lets Your Users Do

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

The following types of business documents are ideal subjects for image processing:

- Business cards
- Letters
- Legal documents
- Handwritten meeting notes
- Memoranda
- Newsclips
- Technical drawings

The following types of personal documents are also ideal subjects for image processing:

- Childrens' drawings
- Hobby-related documents
- Household bills
- Legal documents
- Letters from friends
- Magazine and newspaper articles
- Medical and insurance records
- Receipts
- Tax forms

Depending on how you design and code your application, you can enable your users to:

- Scan images.
- Retrieve and display images.
- Annotate, edit, and manipulate images.
- Convert, copy, and OCR images.
- Append, insert, and replace image pages.
- Display and manage thumbnail representations.
- Set the Summary properties of images for easier retrieval.
- Print, save, and send images.

What Exactly Is Document Imaging?

Imaging for Windows technology brings Imaging functions to many business and personal users — particularly where Imaging for Windows is on virtually every desktop running Windows 95, Windows 98, and Windows NT 4.0 and greater.

The following sections describe the concepts of business and personal document imaging.

Business Document Imaging

Business document imaging is a technology that converts paper documents into an electronic form, where they can be automated using standard computer technology.

Most business information is in the form of paper documents. Industry analysts report that about 94 percent of business information is on paper and that 2.7 billion new sheets of paper are filed into folders every single day.

Paper has obvious advantages, such as portability, ease of use, and low cost. However, paper also has serious drawbacks. At any given time, between three and five percent of a company's files are lost or misplaced. With the average cost of recreating a document at around $180, the cost of losing important business documents can be an expensive one indeed.

Perhaps paper's most serious drawback is that paper-based information is not as readily accessible as computer-based data. A manual business process, not an automated one, uses paper best. To make matters worse, paper-based documents and their respective data usually reside separate from related paper-based documents and their data.

Obtaining information readily or providing "the right information to the right person at the right time" is difficult and costly with the paper-based, manual business process. So, while paper is the major information base of a business, it remains outside of the business information system because it is not easily and reliably accessible.

Computer-based information, on the other hand, is always readily and reliably accessible. Business document imaging is the process of turning paper-based information into computer-based information.

By using a computer to capture paper documents as electronic images, you can apply all the benefits and power of database, e-mail, networking, fax, and storage technology to what was once manually processed information.

Personal Document Imaging

As is the case with business document imaging, personal document imaging is the process of turning paper-based information into computer-based information.

Like business information, most personal information is in the form of paper documents. From personal correspondence with companies to tax forms to hobby-related documents and childrens' drawings, the information in our personal lives is very much paper-centric.

Paper-based personal documents are also subject to permanent loss and temporary misplacement. They're also not as readily accessible as computer-based documents.

By using a computer to capture personal paper-based documents as electronic images, you can store letters, tax forms, and receipts in an organized manner that makes finding them easier. You can preserve college records, hobby-related documents, and childrens' drawings for many years — not to mention share them with family and friends by sending them over the Internet.

Compiling and Distributing Your Image-Enabled Application

You need to make sure that you compile your image-enabled application with the appropriate version of Imaging for Windows.

You also need to make sure that your end users acquire and set up the appropriate version of Imaging for Windows prior to installing your software.

The version of Imaging for Windows you need when compiling your application — and the version your end users need to install — largely depends on the features you have included in your software.

You Included Imaging for Windows Professional Edition Features

If you used a control, property, method, event, or parameter provided exclusively by a particular version of Imaging for Windows Professional Edition (as identified in this guide), you must compile your application with that version of Imaging for Windows Professional Edition.

In addition, you must make sure that your end users purchase and set up that version of Imaging for Windows Professional Edition before using your software.

Your users can obtain Imaging for Windows Professional Edition from the Eastman Software Web site, at:

```
http://www.eastmansoftware.com/
```

You Did Not Include Imaging for Windows Professional Edition Features

If you did *not* use a control, property, method, event, or parameter provided exclusively by Imaging for Windows Professional Edition — and you want your application to be used by Imaging for Windows 95, 98, or NT users — you must compile your application with the appropriate version of Imaging for Windows (95, 98, or NT).

The Imaging for Windows 95, 98, and NT controls are free; as such, there are no royalties or licensing fees required. Your end users must simply set up the appropriate version of Imaging for Windows as a prerequisite to using your program (if they haven't already).

Note: Do not distribute the Imaging OCX files with your application, because doing so will not install Imaging for Windows correctly. Instead, have your users set up the entire Imaging for Windows application to ensure that all of the required software is installed and registered properly.

This guide assumes that users of Imaging for Windows 95 are using the OSR2 release, available free of charge from the Eastman Software Web site, at:

`http://www.eastmansoftware.com/`

Imaging for Windows NT 4.0 ships as a component of Windows NT Workstation 4.0.

Imaging for Windows 98 ships as a component of Windows 98.

Important Notice Concerning the Use of LZW Compression

All versions of Imaging for Windows Professional Edition utilize LZW, a compression/decompression technology that is covered by U.S. Patent 4,558,302 (plus its foreign counterparts, issued or pending). All patents are held by Unisys Corporation.

Eastman Software, Inc.'s grant of license from Unisys does not permit you to use LZW compression or decompression capabilities from any version of Imaging for Windows for development purposes, or to use or derive the LZW capabilities from an Eastman Software third-party application or other derivative software.

Contact Unisys for licensing information at:

Unisys Corporation
Welch Licensing Dept. — MSC1SW19
Township Line and Union Meeting Roads
P.O. Box 500
Blue Bell, PA 19424-0001
Fax: (215) 986-3090
Web: www.unisys.com

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.
Object, arglist	In the syntax section, words in lowercase italics indicate placeholders for information you must provide.
[*expressionlist*]	In the syntax section, items appearing inside square brackets are optional.
{*True* | *False*}	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.

2

Adding Imaging Using Automation

This chapter explains how to use Automation to image-enable your applications. It begins by describing the object hierarchy of the Imaging application and continues by describing how the Imaging application can function as an Automation server application or an Embedded server application. The chapter concludes by walking you through a sample project to help you get started.

In This Chapter

Overview

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.

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

Using Automation, you can 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: The Imaging Preview and Imaging Flow applications cannot be automated.

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.

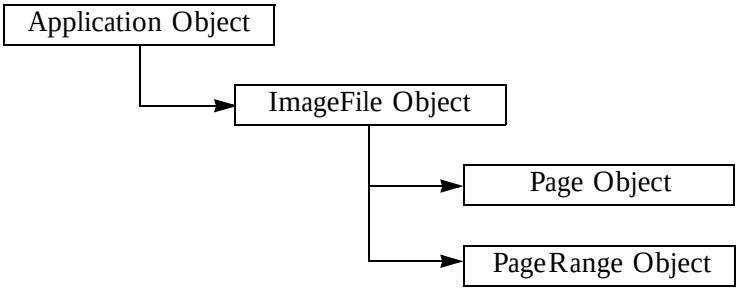
The remainder of this chapter:

- Outlines the Object Hierarchy of the Imaging application.
- Describes how the Imaging application can function as an Automation server application or an Embedded server application.
- Provides an example of automating the Imaging application from Microsoft Excel.

The Object Hierarchy

The object model of the Imaging application includes:

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



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

Each object in the hierarchy has its own set of properties and methods. Refer to Chapter 3 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.

Note: Contact sheets and OCR are available with Imaging for Windows Professional Edition only.

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 if a change has occurred. At your option, you can use them to update the corresponding objects known to Automation.

Automation Server and Embedded Server 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 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

Beginning with Imaging for Windows Professional Edition Version 2.0, you can use 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: Remember that 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 you 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 you 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.

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.

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

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, Version 5.0 and Excel 97.

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, Eastman Software, Inc. has prepared a demonstration project — called Automation From Excel — that shows you how to:

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

Note: Chapter 3 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. Chapter 4 of this guide describes the concepts of image display, page counts, multipage image files, and image rotation.

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 Chapters 7 through 11 in this guide for more information.

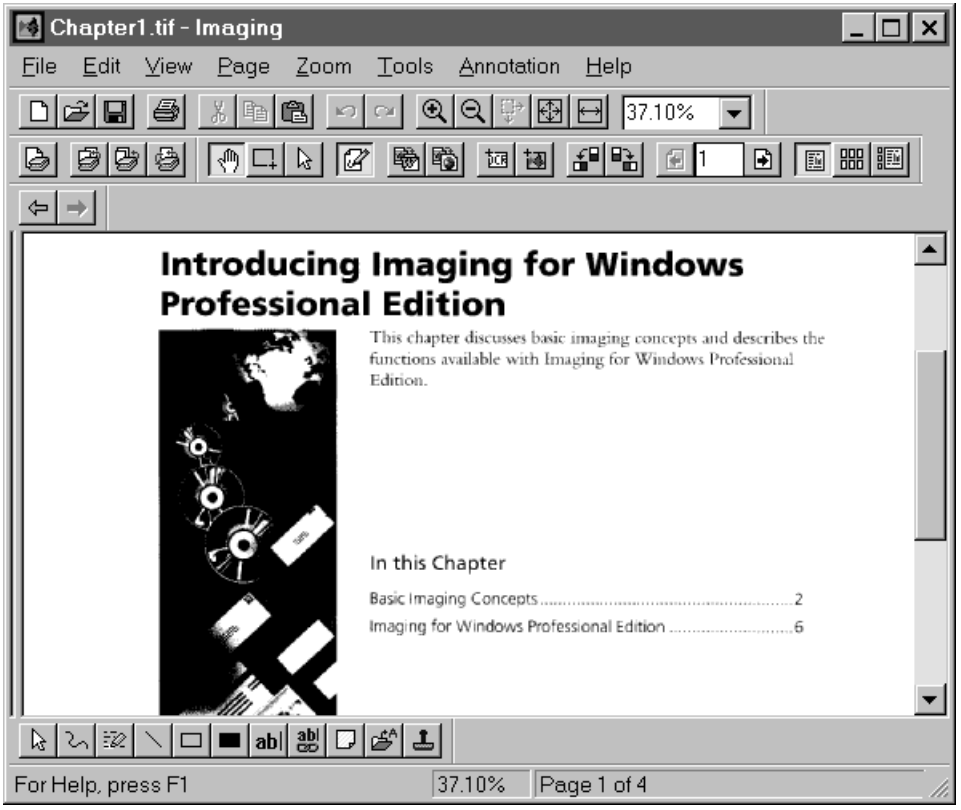
The Imaging application has three view modes that enable users to view and work with image files. Each view mode has its own set of advantages and capabilities.

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.



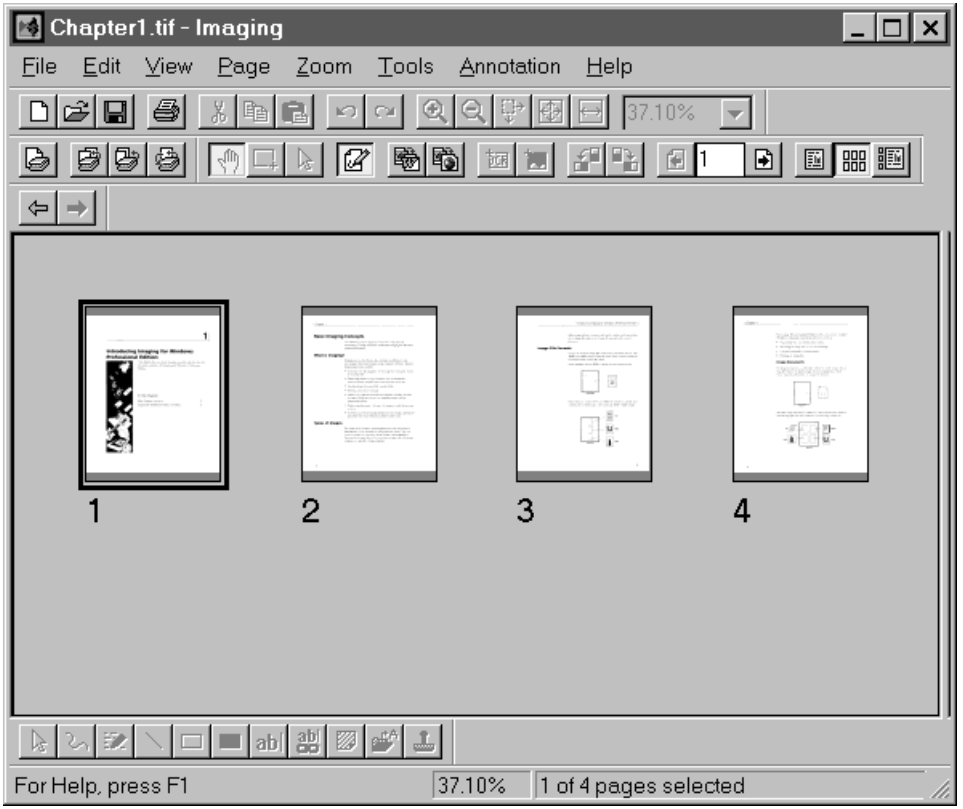
Thumbnail

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 — like annotation and zoom — are not available in this mode because they are not appropriate for use on such small images.

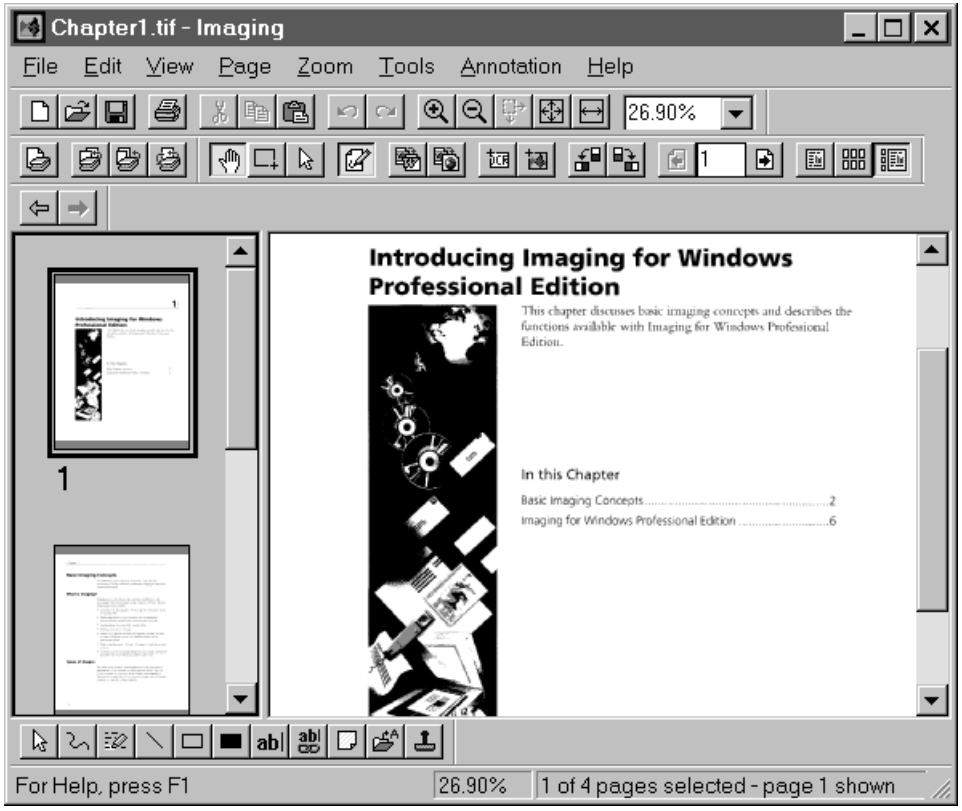
Rearranging image pages is available with Imaging for Windows Professional Edition only.



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 PCs 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 you if there was any way 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 `AutoFromExcel.bas`, `ImagingAutomation.xls`, and `Facc.tif`.

As stated previously, 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:

AutoFromExcel.bas — A Visual Basic for Applications (VBA) code module that contains macros that automate the Imaging application.

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.

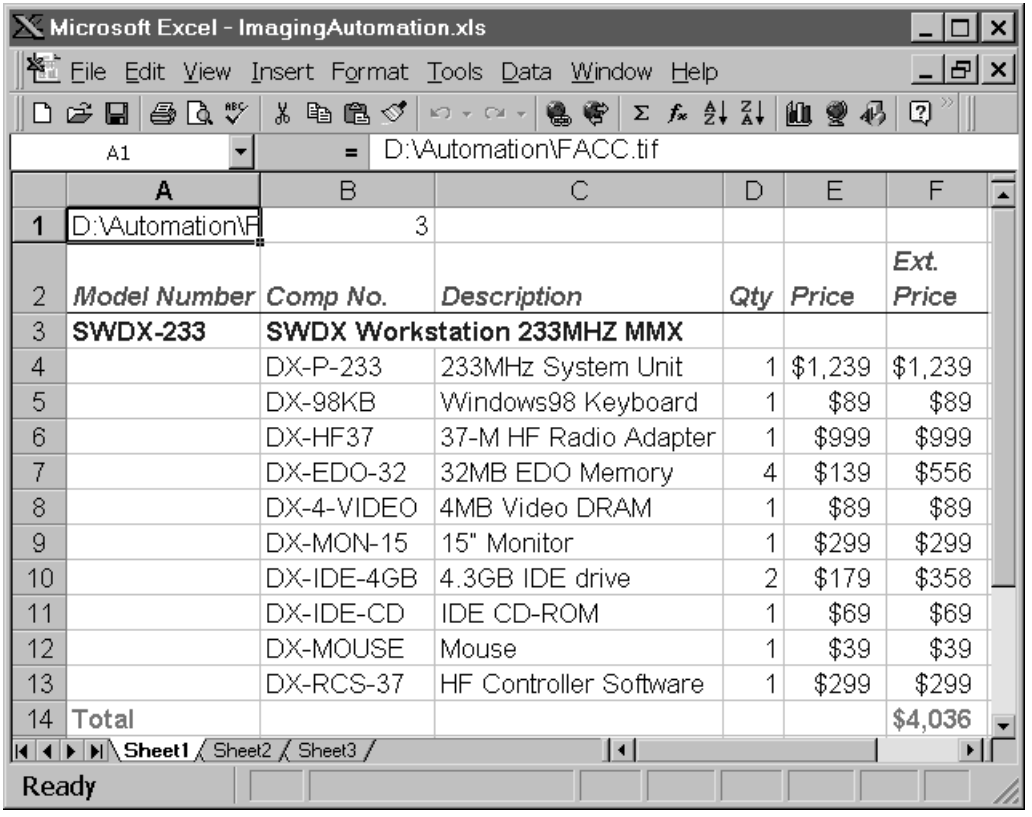
RotateLeft method (Page object) — Rotates the image 90 degrees counterclockwise.

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.

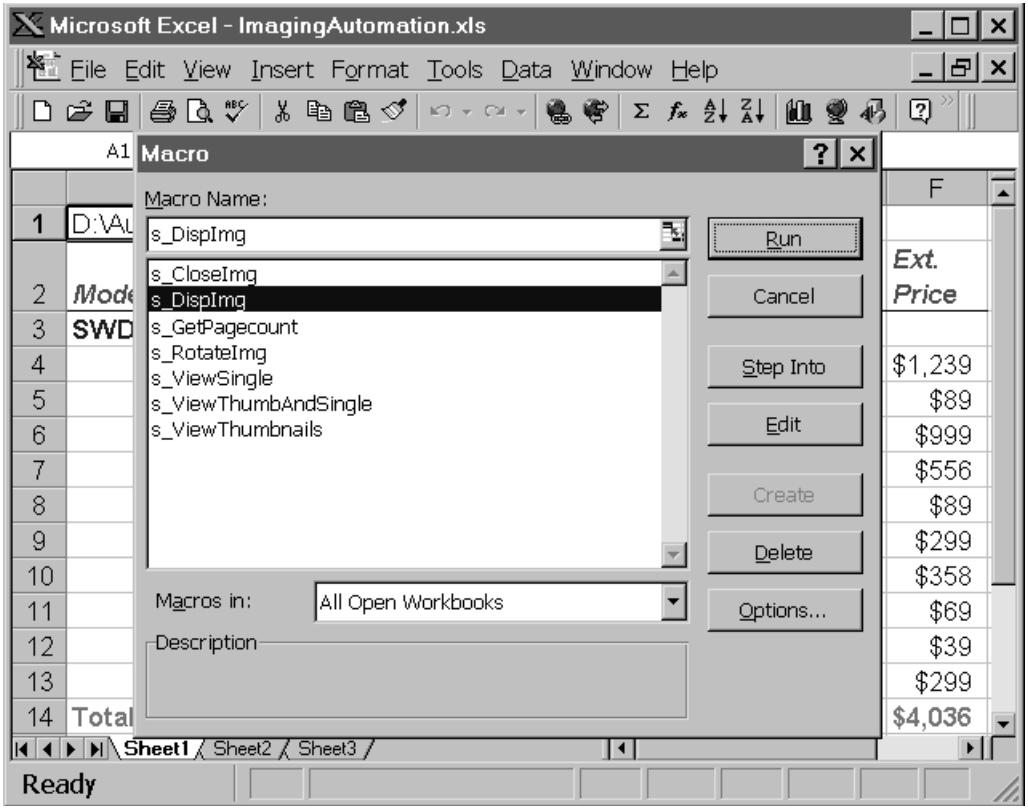


	A	B	C	D	E	F
1	D:\Automation\F	3				
2	*Model Number*	*Comp No.*	*Description*	*Qty*	*Price*	*Ext. Price*
3	**SWDX-233**	**SWDX Workstation 233MHZ MMX**				
4		DX-P-233	233MHz System Unit	1	$1,239	$1,239
5		DX-98KB	Windows98 Keyboard	1	$89	$89
6		DX-HF37	37-M HF Radio Adapter	1	$999	$999
7		DX-EDO-32	32MB EDO Memory	4	$139	$556
8		DX-4-VIDEO	4MB Video DRAM	1	$89	$89
9		DX-MON-15	15" Monitor	1	$299	$299
10		DX-IDE-4GB	4.3GB IDE drive	2	$179	$358
11		DX-IDE-CD	IDE CD-ROM	1	$69	$69
12		DX-MOUSE	Mouse	1	$39	$39
13		DX-RCS-37	HF Controller Software	1	$299	$299
14	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 constants that represent the early and recent versions of the Imaging application. The code also declares the object variables that contain references to the Application and Image File objects.

```
Const WANG_IMG = "wangimg.exe"      'Older versions of Imaging
Const KODAK_IMG = "kodakimg.exe"    'Newer versions of Imaging

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 if 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.

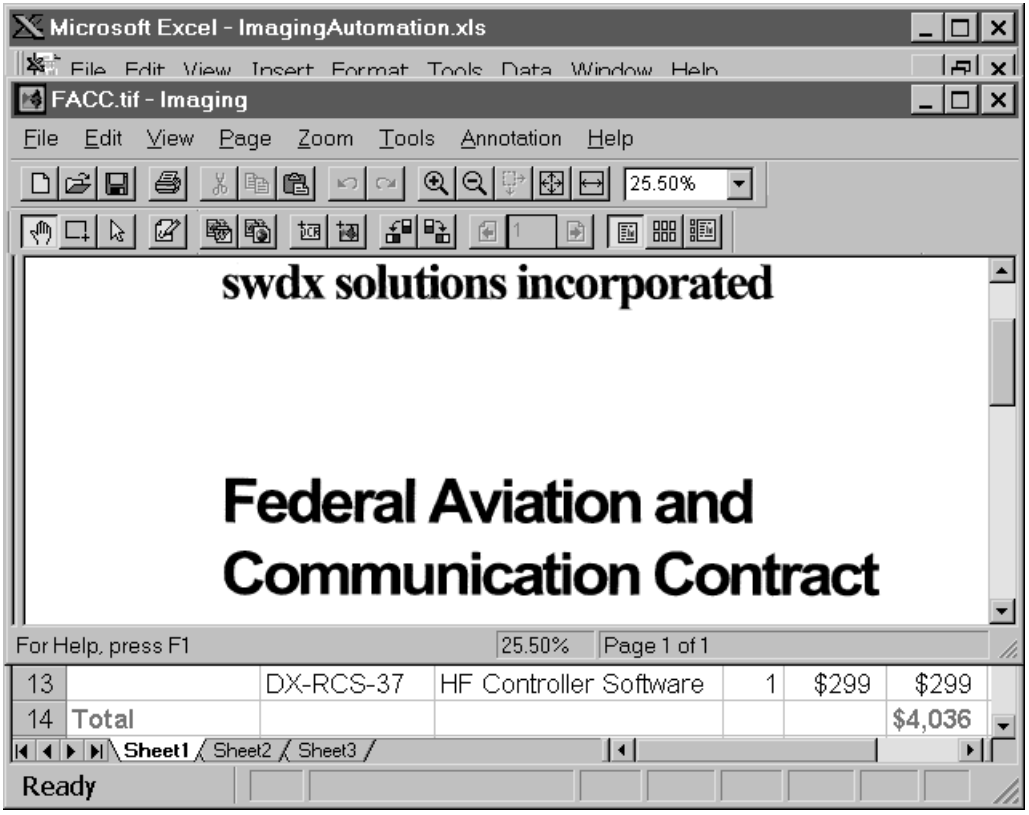
```
Function f_InitializeApp() As Boolean

    Dim strWinDir As String

    Set objApp = Nothing
    Set objImg = Nothing
    strWinDir = Environ("WinDir")  'does not apply to NT
    If LCase$(Dir(strWinDir & "\" & WANG_IMG)) = WANG_IMG Then
        Set objApp = CreateObject("WangImage.Application")
    Else
        If LCase$(Dir(strWinDir & "\" & KODAK_IMG)) = KODAK_IMG Then
            Set objApp = CreateObject("Imaging.Application")
        Else
            MsgBox ("The Imaging Application could not be found")
            f_InitializeApp = False
            Exit Function
        End If
    End If

    Set objImg = objApp.CreateImageViewerObject(1)
    f_InitializeApp = True

End Function
```

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

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

Then it checks to see if 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
```

3

Automation Lexicon

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

In This Chapter

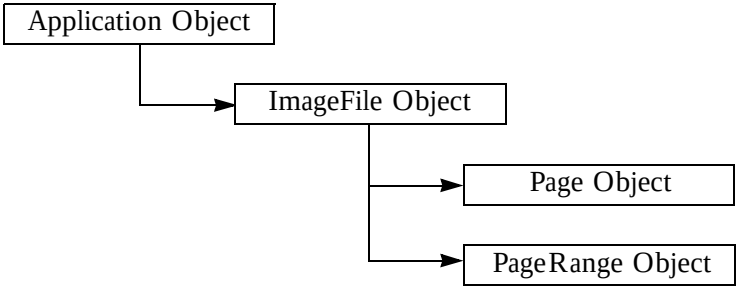
Overview

This chapter describes the properties and methods of each object in the Imaging application object hierarchy.

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.

The object model of the Imaging application consists of:

- 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.



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

Note: Refer to Chapter 2 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. Note that the properties that affect the displayed image (for example, **DisplayScaleAlgorithm**, **ImagePalette**, and **Zoom**) affect every image displayed in the Application object.

Application Object Properties

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

Application Object Properties (cont.)

Property	Description
ScanIsAvailable	Sets or returns the state of the scanner. Professional Edition only.
ScanToolBarVisible	Sets or returns the visibility of the application's imaging toolbar. Not available in all releases.
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. Professional edition only.
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).

Setting	Description
3	Stamp — Represents the image as a thumbnail.
4	Optimize — Changes the display scale algorithm based on the image type of the displayed image. Black and white images are scaled to gray. Palettized 4- and 8-bit, RGB, and BGR images remain color.

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

Edit Property

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

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

Data Type Integer (Boolean).

Remarks The **Edit** property settings are:

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

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

FullName Property

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

Usage *ApplicationObject*.**FullName**

Data Type String.

Height Property

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

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

Data Type Long.

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

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

ImagePalette Property

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

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

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

Data Type Short.

Remarks The **ImagePalette** property settings are:

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

ImageView Property

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

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

Data Type Short.

Remarks The **ImageView** property settings are:

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

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

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

See Also ImageFileObject.**ActivePage** property.

ImagingToolBarVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **ImagingToolBarVisible** property settings are:

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

Left Property

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

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

Data Type Long.

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

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

Name Property

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

Usage *ApplicationObject*.**Name**

Data Type String.

Parent Property

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

Usage *ApplicationObject*.**Parent**

Data Type Object.

Path Property

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

Usage *ApplicationObject*.**Path**

Data Type String.

ScannerIsAvailable Property

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

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

Data Type Integer (Boolean).

Remarks The **ScannerIsAvailable** property settings are:

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

ScanToolBarVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **ScanToolBarVisible** property settings are:

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

ScrollBarsVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **ScrollBarsVisible** property settings are:

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

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

StatusBarVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **StatusBarVisible** property settings are:

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

ToolBarVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **ToolBarVisible** property settings are:

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

Top Property

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

Usage *ApplicationObject*.**Top**

Data Type Long.

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

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

TopWindow Property

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

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

Data Type Integer (Boolean).

Remarks The **TopWindow** property settings are:

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

Example

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

Visible Property

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

Usage *ApplicationObject*.**Visible**

Data Type Integer (Boolean).

Remarks The **Visible** property settings are:

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

WebToolBarVisible Property

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

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

Data Type Integer (Boolean).

Remarks The **WebToolBarVisible** property settings are:

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

Width Property

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

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

Data Type Long.

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

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

Zoom Property

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

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

Data Type Float.

Remarks The zoom factor is a percent value.

Example

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

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

Application Object Methods

The following table lists the Application object methods.

Application Object Methods

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

CreateImageViewerObject Method

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

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

Data Type Object.

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

Example

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

FitTo Method

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

This method affects each view as follows:

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

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

Data Type Short.

Remarks ZoomOption settings are:

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

Help Method

Description Displays the Imaging online Help table of contents.

Usage *ApplicationObject*.**Help**

Quit Method

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

Usage *ApplicationObject*.**Quit**

ImageFile Object

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

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

ImageFile Object Properties

The following table lists the ImageFile object properties.

ImageFile Object Properties

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

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	AWD – Windows 95
3	BMP
4	PCX
5	DCX
6	JPEG
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. Professional Edition only.
FindOIServerDoc	Finds *Eastman Software* Imaging Server documents. Not available when the application is running as an embedded server. Professional Edition only.
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. Professional Edition only.
Open	Opens the ImageFile object. Not available when the application is running as an embedded server.
Pages	Returns a Page or PageRange object for the ImageFile object.
Print	Prints the ImageFile object.
RotateAll	Rotates all ImageFile object pages.
Save	Saves changes to the ImageFile object.
SaveAs	Saves the ImageFile object under another name.
SaveCopyAs	Saves a copy of the ImageFile object. The application must be running as an embedded server.
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 Imaging Server 3.x
'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 an *Eastman Software* Imaging server document. This method displays an Imaging server document **Find** dialog box, from which the user may search for *Eastman Software* Imaging 1.x server documents or *Eastman Software* Imaging 3.x server 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.

Note: This method is available only with the Imaging for Windows Professional Edition application.

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 Imaging Server 3.x 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 Imaging Server 3.x 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	AWD (not supported on NT)
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, AWD, 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.
Img.Pages(1).ScrollPositionY = 200

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

Width Property

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

Usage *PageObject*.**Width**

Data Type Long.

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

Page Object Methods

The following table lists the Page object methods.

Page Object Methods

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

Delete Method

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

Usage *PageObject*.**Delete**

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

Flip Method

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

Usage *PageObject*.**Flip**

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

Help Method

Description Displays the Imaging online Help table of contents.

Usage *PageObject*.**Help**

Ocr Method

Description OCRs the image page.

Usage *PageObject*.**Ocr**

Print Method

Description Prints the page.

Usage *PageObject*.**Print**

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

RotateLeft Method

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

Usage *PageObject*.**RotateLeft**

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

RotateRight Method

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

Usage *PageObject*.**RotateRight**

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

Scroll Method

Description Scrolls the page.

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

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

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

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

PageRange Object

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

PageRange Object Properties

The following table lists the PageRange object properties.

PageRange Object Properties

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

Application Property

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

Usage *PageRangeObject*.**Application**

Description Object.

Count Property

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

Usage *PageRangeObject*.**Count**

Data Type Long.

EndPage Property

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

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

Data Type Long.

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

Parent Property

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

Usage *PageRangeObject*.**Parent**

Data Type Object.

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

StartPage Property

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

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

Data Type Long.

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

PageRange Object Methods

The following table lists the PageRange object methods.

PageRange Object Methods

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

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

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

Delete Method

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

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

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

Ocr Method

Description OCRs the page range.

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

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

Print Method

Description Prints the page range.

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

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

4

Adding Imaging Using ActiveX Controls

This chapter demonstrates how to use the Imaging ActiveX controls to image-enable your applications.

It begins by explaining how to load the Imaging ActiveX controls into your development environment. Then it explains how to access the on-line help for the controls. It concludes by walking you through some sample applications to help you get started.

In This Chapter

Loading the Controls

This section explains how to load the Imaging ActiveX controls into three development environments: Microsoft Visual Basic, Visual C++, and Access.

Keep in mind that you can use the Imaging ActiveX controls with other programming environments, such as Powersoft PowerBuilder.

Before you can use the Imaging ActiveX controls, you must load them into your development environment.

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 list shows how the Imaging ActiveX controls appear on your system:

  – Kodak Image Admin Control
  – Kodak Image Edit Control
  – Kodak Image Ocr Control
  – Kodak Image Scan Control
  – Kodak Image Thumbnail Control

Note: The name Wang appears instead of Kodak when using Imaging for Windows 95, Imaging for Windows NT 4.0, and Imaging for Windows Professional Edition V1.0 and V1.1. (The Eastman Kodak Company acquired Wang Software, the software unit of Wang Laboratories, Inc. in 1997, creating Eastman Software, Inc.)

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	Notes
	Image Admin	Available with all versions of Imaging.
	Image Annotation Tool Button	Actually a member of the Image Edit control.
	Image Edit	Available with all versions of Imaging.
	Image OCR	Available with Imaging Professional only.
	Image Scan	Available with all versions of Imaging.
	Image Thumbnail	Available with all versions of Imaging.

Visual Basic

To add the Imaging ActiveX controls to Visual Basic

1. Start Visual Basic and create a new project.
2. On the **Project** menu, click **Components**.
3. On the **Components** dialog box, click the **Controls** tab.
4. Select the Imaging ActiveX controls from the controls listed. (Refer to "Loading the Controls" earlier in this chapter to see a list of Imaging ActiveX controls.)

A control is selected when a check mark appears next to it.

5. Click **OK**. Visual Basic adds the controls to your project and the control icons to your toolbox.
6. Work with the Imaging ActiveX controls as you would any other type of ActiveX control.

Visual C++

To add the Imaging ActiveX controls to Visual C++

1. Start Visual C++ and create a new project.
2. On the **Project** menu, point to **Add to Project**, and click **Components and Controls**. The **Component and Controls Gallery** dialog box appears.
3. In the **Look In** list box, click **SharedIDE** and then **Gallery**.
4. Below the **Look In** list box, double-click **Registered ActiveX Controls**. A list of registered ActiveX controls appears.
5. For *each* Imaging ActiveX control:
   a. Click the desired control among the list of registered controls and then click **Insert**. (Refer to "Loading the Controls" earlier in this chapter to see a list of Imaging ActiveX controls.)
   b. On the **Confirm Classes** dialog box, click **OK**. Visual C++ adds the control to your project and its icon to the Controls toolbox, which is visible when you edit a dialog box in Resource View.
6. When you have finished adding the Imaging ActiveX controls, click **Close** on the **Components and Controls Gallery** dialog box.
7. Work with the Imaging ActiveX controls as you would any other type of ActiveX control.

Note: If you are using Microsoft Foundation Classes (MFC), be sure to call **AfxEnableControlContainer** within `InitInstance`.

Access

If you selected the OLE Controls check box in the Setup program when you installed Microsoft Access, the Imaging ActiveX controls are available automatically.

To add the Imaging ActiveX controls to Access

1 Start Access and create a new database.

2 On the **Tools** menu, click **ActiveX Controls**.

3 If the **ActiveX Controls** dialog box lists all of the Imaging ActiveX controls as available, click **Close** and proceed to Step 4.

For each control the dialog box does *not* list as available:

a Click **Register**.

b On the **Add ActiveX Control** dialog box, navigate to your `Windows/System` folder.

c Click the file name of the control not listed as available. Then click **Open** to register the control.

The following table lists the file names of each Imaging ActiveX control.

Imaging ActiveX Control File Names

Click This File	To Register This Control
imgadmin.ocx	Image Admin
iimgedit.ocx	Image Annotation Tool Button
iimgedit.ocx	Image Edit
iimgocr.ocx	Image OCR
iimgscan.ocx	Image Scan
iimgthumb.ocx	Image Thumbnail

d When you finish registering the controls, click **Close** to exit the **ActiveX Controls** dialog box.

4 Enter Form or Report design view.

5 On the **View** menu:

a If necessary, click **Toolbox** to display the **Controls** toolbox.

b Point to **Toolbars** and click **Customize**.

6 On the **Customize** dialog box, click the **Commands** tab.

7 In the **Categories** list box:

a Click **ActiveX Controls**.

b From the **Commands** list box, drag each Imaging ActiveX control and drop it onto the **Controls** toolbox.

c When you finish dragging and dropping the Imaging ActiveX controls onto the toolbox, click **Close**.

8 Work with the Imaging ActiveX controls as you would any other type of ActiveX control.

Obtaining Help

This section explains how to access the on-line help system of the Imaging ActiveX controls.

How you access the Imaging ActiveX Controls on-line help system differs within each of the following programming environments:

- Visual Basic
- Visual C++
- Access

The following sections describe how to access help in each environment.

If desired, you can also open the help file in Windows Explorer. Refer to Chapter 6 for instructions.

Note: Several methods in the Imaging ActiveX controls present dialog boxes to the end user. Each dialog box provides its own context-sensitive help, which the user can invoke by clicking the question mark at the top of the dialog box and then the desired control.

Visual Basic

There are many ways to access the Imaging ActiveX Controls on-line help in Visual Basic. You can access help from the:

- **Object Browser**
- **Toolbox**
- **Form** window
- **Properties** window
- **Code** window

Before attempting to access help, make sure that the Imaging ActiveX controls have been added to your current project. (Refer to “Loading the Controls” for instructions.)

Object Browser

To access Imaging **ActiveX help from the Object Browser**

1. On the **View** menu, click **Object Browser**. The **Object Browser** appears.
2. In the **Project/Library** list box, click the library name of the desired Imaging ActiveX control.

   The following table lists the library and class names of each Imaging ActiveX control.

Imaging ActiveX Control Library and Class Names

Library Name	Class Name	Imaging Control
AdminLibCtl	ImgAdmin	Image Admin
ImgeditLibCtl	ImgAnnTool	Image Annotation Tool Button
ImgeditLibCtl	ImgEdit	Image Edit
IMGOCRLib	Imgocr	Image OCR
ScanLibCtl	ImgScan	Image Scan
ThumbnailLibCtl	ImgThumbnail	Image Thumbnail

3. In the **Classes** list box, click the class name of the control. (Refer to the preceding table for a list of class names.)
4. In the **Members** list box, click the desired property, method, or event, and then press F1. The help topic for the selected member appears.

Note: If the member you select is an extender property, method, or event, Visual Basic's on-line help appears.

Toolbox

To access Imaging **ActiveX help from the Toolbox**

- Click the desired Imaging ActiveX control in the toolbox, and then press F1. The overview topic for the selected control appears.

  From the overview topic, you can navigate to other topics that describe the properties, methods, and events of the selected control.

Form Window

To access Imaging **ActiveX help from the Form window**

1. Draw at least one Imaging ActiveX control on a form.
2. Select the Imaging ActiveX control on the form.
3. Press F1. The overview topic for the selected control appears.

   From the overview topic, you can navigate to other topics that describe the properties, methods, and events of the selected control.

Properties Window

To access Imaging **ActiveX help from the Properties window**

1. Select an Imaging ActiveX control that has been drawn on a form. Then, on the **View** menu, click **Properties Window**. The **Properties** window appears.
2. Click the desired property in the **Properties** window and then press F1. The help topic for the selected property appears.

Note: Keep in mind that only the design-time properties appear in the **Properties** window.
If the property you select is an extender property, Visual Basic's on-line help appears.

Code Window

To access Imaging **ActiveX help from the Code window**

1. Make sure that at least one Imaging ActiveX control has been drawn on a form.
2. Invoke the **Code** window.
3. Within your code, select the Imaging property, method, or event for which you want help. Then press F1. The help topic for the selected property, method, or event appears I.

Note: If the property, method, or event you select is an extender property, method, or event, Visual Basic's on-line help appears.

Visual C++

There are two ways to access the Imaging ActiveX Controls on-line help in Visual C++. You can access help from the:

- **Components and Controls Gallery** dialog box
- **Properties** window

Before attempting to access help, make sure that the Imaging ActiveX controls have been added to your current project. (Refer to "Loading the Controls" for instructions.)

Components and Controls Gallery Dialog Box

To access Imaging **ActiveX help from the Components and Controls Gallery dialog box**

1. On the **Project** menu, point to **Add To Project** and click **Components and Controls**. The **Components and Controls Gallery** dialog box appears.
2. Click the desired Imaging ActiveX control below the **Lookin** box, and then click **More Info**. The overview topic for the selected control appears.

From the overview topic, you can navigate to other topics that describe the properties, methods, and events of the selected control.

Properties Window

To access Imaging ActiveX help from the Properties window

1. Select an Imaging ActiveX control that has been drawn on a form.
2. On the **View** menu, click **Properties**.
3. Click the desired property in the **Properties** window, and then press F1. The help topic for the selected property appears.

Note: Keep in mind that only the design-time properties appear in the **Properties** window.
If the property you select is an extender property, the contents window for the Imaging ActiveX help system appears.

Access

There are three ways to access the Imaging ActiveX Controls on-line help in Access. You can access help from the:

- **Object Browser**
- **Properties** window
- **Module** window

Before attempting to access help, make sure that the Imaging ActiveX controls have been added to your current database. (Refer to "Loading the Controls" for instructions.)

Object Browser

To access Imaging **ActiveX help from the Object Browser**

1. Make sure the **Module** window is currently on display.
2. On the **View** menu, click **Object Browser**. The **Object Browser** appears.
3. In the **Project/Library** list box, select the library of the desired Imaging ActiveX control. (Refer to "Visual Basic" earlier in this section to see a list of library and class names for each Imaging ActiveX control.)
4. In the **Classes** list box, click the class name of the desired control. (Refer to the aforementioned list.)
5. In the **Members** list box, click the desired property, method, or event, and then press F1. The help topic for the selected member appears.

Note: If the member you select is an extender property, method, or event, Visual Basic's on-line help appears — provided Visual Basic is installed on your system. Access does not provide help for extender properties, methods, or events in the **Object Browser**.

Properties Window

To access Imaging ActiveX help from the Properties window

1. Select an Imaging ActiveX control that has been drawn on a form.
2. On the **View** menu, click **Properties**.
3. On the **Properties** window, click the **Other** tab or the **All** tab.

4 Click the desired property in the **Properties** window and then press F1. The help topic for the selected property appears.

Note: Keep in mind that only the design-time properties appear in the **Properties** window.
If the property you select is an extender property, Access' on-line help appears.

Module Window

To access Imaging ActiveX help from the Module window

1 Make sure that at least one Imaging ActiveX control has been drawn on a form.

2 Invoke the **Module** window.

3 Select the Imaging property, method, or event within your code for which you want help. And then press F1. The help topic for the selected property, method, or event appears I.

Note: If the property, method, or event you select is an extender property, method, or event, Visual Basic's on-line help appears — provided Visual Basic is installed on your system.

Demonstration Projects

This section demonstrates how to add a variety of Imaging functions to your applications.

A wide-ranging discussion of every Imaging function is beyond the scope of this chapter; however it does discuss many of the popular ones.

All demonstration projects were developed using Microsoft Visual Basic, Version 5.0.

To help you use the Imaging ActiveX controls, Eastman Software, Inc. has prepared seven demonstration projects that 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.
- Manage an image file using thumbnails.
- Unload a multipage image file.

Note: Some of the properties, methods, events, parameters, and constants discussed in this section may not be available if you use a version of Imaging that is earlier than Imaging for Windows Professional Edition Version 2.0.
The on-line help system and Chapters 7 through 11 of this guide identify the properties, methods, events, parameters, and constants that are available in each version of Imaging for Windows.

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. (Refer to Chapters 7 through 11 in this guide for more information.)

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, which is particularly important when users *read* scanned documents or faxes.

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.

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

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.

Inch To Inch — Scales the image page to its actual size, so that one inch on the scanned image equals one inch on your monitor.

Example

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

Scenario

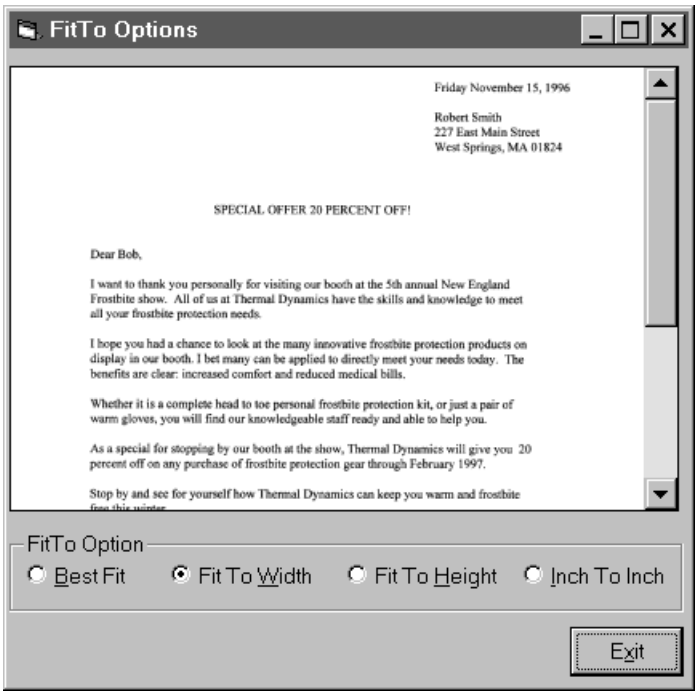
Assume Eileen receives several scanned business documents in her role as product manager for a major computer company. Because she receives these documents from others via e-mail, she has no control over how they are scanned, but she really needs to be able to read them.

Because you included all of Image Edit's 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
    kdkImgEdit1.FitTo BEST_FIT
    optFitTo(0).Value = True

    'Display the selected image file
    kdkImgEdit1.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).

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.

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

```
Private Sub optFitTo_Click(Index As Integer)

    Select Case Index

        Case 0 'Best Fit
            kdkImgEdit1.FitTo BEST_FIT, True

        Case 1 'Fit To Width
            kdkImgEdit1.FitTo FIT_TO_WIDTH, True

        Case 2 'Fit To Height
            kdkImgEdit1.FitTo FIT_TO_HEIGHT, True

        Case 3 'Inch To Inch (Actual Size)
            kdkImgEdit1.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. (Refer to Chapters 7 through 11 in this guide for more information.)

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 PC.

AWD — While it supports only the Black-and-White color type, AWD image files can contain multiple pages; and AWD is the format for Microsoft Windows 95 Fax documents.

Note: The AWD format is not available when your application runs on Windows NT.
Windows 98 supports AWD as a legacy file type.

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, like 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 Compression. The higher the Resolution setting, the greater the image quality. The higher the Compression setting, the lower the disk space requirement.

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

Note: If you make the LZW compression type available to your users, you need to negotiate a license with Unisys Corporation.

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

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 1 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 OCRd or 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 PC — 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.

Note: Only Imaging for Windows 98 and Imaging for Windows Professional Edition support changing the size of an image.

Example

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

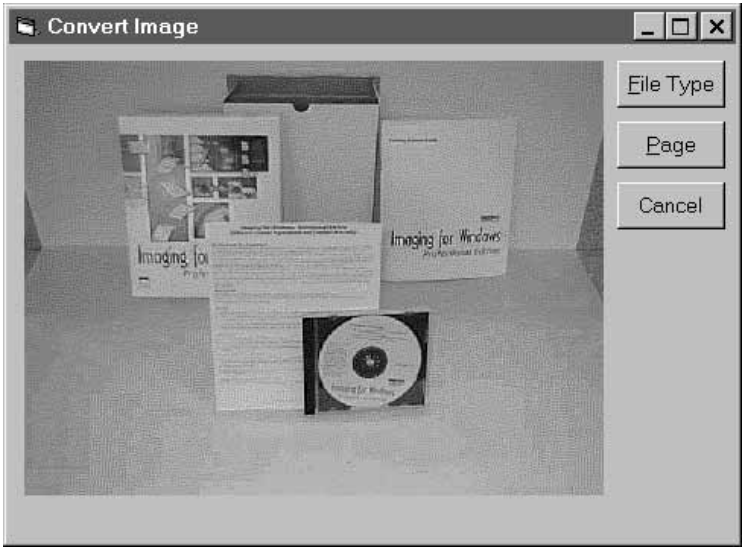
Scenario

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 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. Conveniently, the numeric value of the **FilterIndex** property equals the literal value of each file type. (Refer to the "**SaveAs Method**" of the Image Edit control in Chapter 7 for more information.)

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`.

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

    Select Case Index

        Case 0 'File Type button

            'Set the Filter property to file types that can be written.
            kdkImgAdmin1.Filter = "TIFF (*.tif)|*. tif|FAX (*.awd)|*. awd| _
                Bitmap (*.bmp)|*. bmp|"

            'Set the FilterIndex property to the file type of the displayed image
            'if it can be written; otherwise to TIFF.
            If kdkImgAdmin1.FileType < 4 Then
                kdkImgAdmin1.FilterIndex = kdkImgAdmin1.FileType
            Else
                kdkImgAdmin1.FilterIndex = 1
            End If

            'Invoke ShowFileDialog method.
            On Error GoTo CancelPressed_EH
            kdkImgAdmin1.ShowFileDialog SaveDlg, frmConvertImage.hWnd

            'Set Image Edit's Image property to the file name returned by the
            'Open dialog box.
            kdkImgEdit1.Image = kdkImgAdmin1.Image

            'Set the iFileType variable to the file type returned by the
            'Open dialog box.
            intFileType = kdkImgAdmin1.FilterIndex

            'Invoke the SaveAs method using the new file name and file type
            kdkImgEdit1.SaveAs kdkImgEdit1.Image, intFileType
.
.
.
End Select

CancelPressed_EH:

End Sub
```

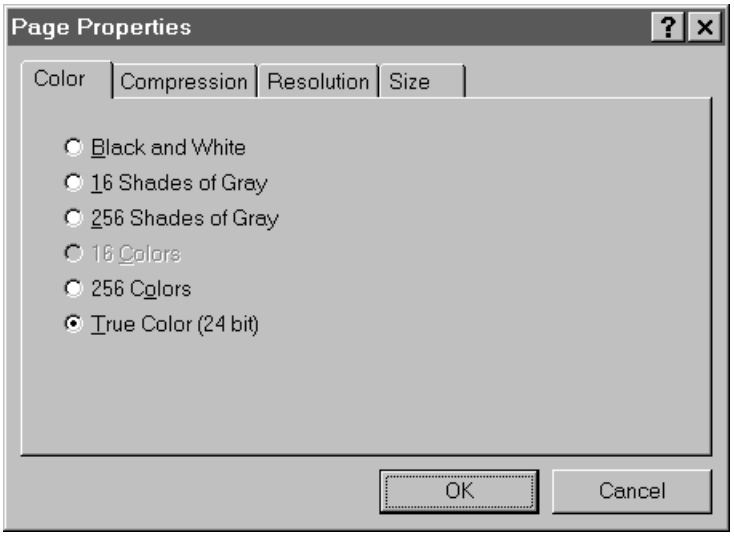
The **SaveAs** method saves the image file using the new file name and file type.

Changing the Color, Compression, Resolution, and Size

If necessary, start the Convert 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 **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 `intResponse` local variable.

The procedure evaluates the value of the `intResponse` 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 intResponse As Integer

    Select Case Index
.
.
.
       Case 1 'Page button

            'Display the ShowPageProperties dialog box to let users convert the
            'image
            intResponse = kdkImgEdit1.ShowPageProperties(False)
            If intResponse = vbOK Then
                'User clicked OK on the dialog box so save the converted image
                kdkImgEdit1.Save
            ElseIf intResponse = 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. (Refer to Chapters 7 through 11 in this guide for more information.)

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 if image or annotation data is present in the Clipboard. You can use this method to see if 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

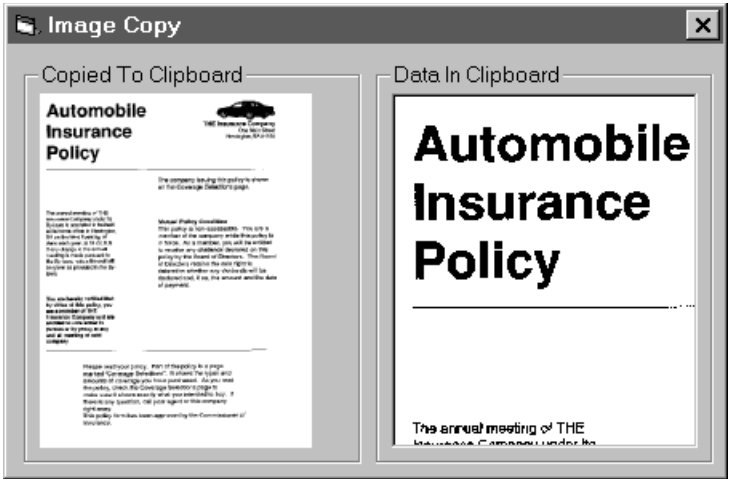
Assume Susan is writing a follow-up letter to her insurance company about a reimbursement claim she has yet to receive. Before she sent the original receipt, she scanned it and saved it to disk. Now she wants to include a copy of the receipt in her follow-up letter.

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 (0,0).
- The Width and Height of the image in pixels, as provided by the current values of the **ImageScaleWidth** and **ImageScaleHeight** 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) I.

```
Private Sub Form_Load()
    Dim vntTemp As Variant
    Dim lngRPosition As Long
    Dim sngLeftChar As Single, sngRightChar As Single
    .
    .
    .
    'Check for valid TIFF file.
    If kdkImgAdmin1.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.
        kdkImgEdit1.FitTo FIT_TO_WIDTH

        'Display the image.
        kdkImgEdit1.Display

        'Copy the whole image onto the Clipboard.
        kdkImgEdit1.ClipboardCopy 0, 0, kdkImgEdit1.ImageScaleWidth, _
             kdkImgEdit1.ImageScaleHeight

        'Get the image data from the the Clipboard and display it in
        'the PictureBox control to make show it was copied.
        picImage = Clipboard.GetData()
    End If

    Exit Sub

    File_EH:

    MsgBox "Quitting the program now. Please select a TIFF file to use the _
        program."
    'Quit the program
    End

End Sub
```

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. (Refer to Chapters 7 through 11 in this guide for more information.)

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 — only 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 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 Print Image project demonstrates printing an image file.

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

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

And 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.

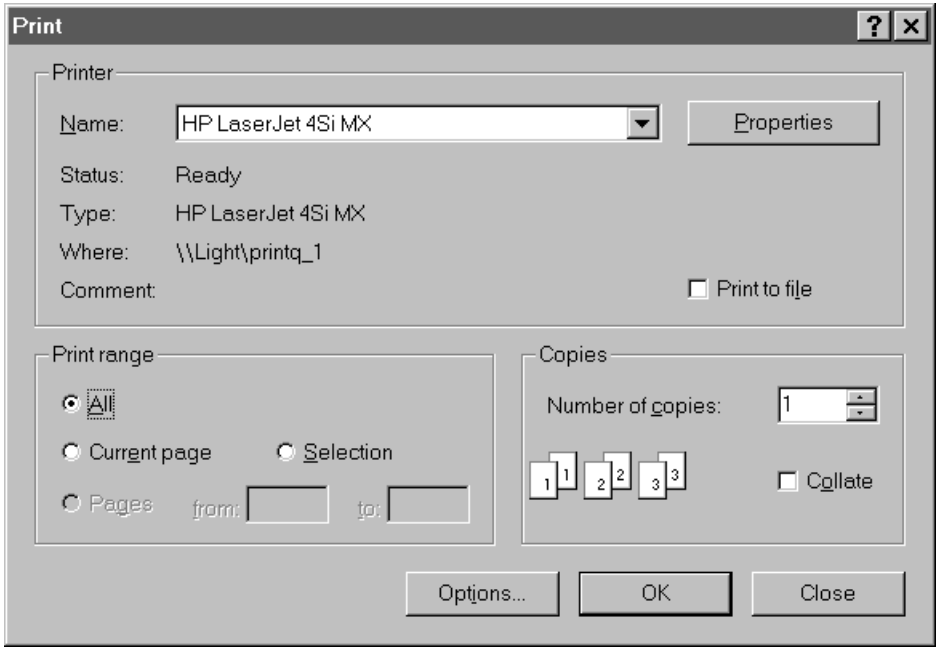
After you select the image file, the procedure displays it in the Image Edit control.

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[a]	**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[a]	**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
PrintOutputFormat[a]	**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[b]

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
PrintRangeOption	The following option buttons on the **Print** dialog box: ▪ **All pages** ▪ **Current page** ▪ **Selection** ▪ **Pages**	Whether to print: **0** — All pages **1** — Range of pages **2** — Current page **3** — Selection
PrintStartPage[a]	**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

a. You can include the values of these properties in your call to the **PrintImage** method to set the range of pages to print, determine whether to print annotations, and to set the output format. The remaining properties affect the print operation directly.

b. Available with Imaging for Windows Professional Edition Version 2.0 and greater.

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.
            kdkImgAdmin1.ShowPrintDialog frmPrintImage.hWnd

            'User pressed OK continue with print
            If kdkImgAdmin1.StatusCode = 0 Then

                'Check on which option the user selected then print
                'image using the Image Edit control.
                If kdkImgAdmin1.PrintRangeOption = PrintAll Then
                    kdkImgEdit1.PrintImage kdkImgAdmin1.PrintStartPage, _
                    kdkImgAdmin1.PrintEndPage, kdkImgAdmin1.PrintOutputFormat, _
                    kdkImgAdmin1.PrintAnnotations
                End If

                If kdkImgAdmin1.PrintRangeOption = PrintRange Then
                    kdkImgEdit1.PrintImage kdkImgAdmin1.PrintStartPage, _
                    kdkImgAdmin1.PrintEndPage, kdkImgAdmin1.PrintOutputFormat, _
                    kdkImgAdmin1.PrintAnnotations
                End If

                If kdkImgAdmin1.PrintRangeOption = PrintCurrent Then
                    kdkImgEdit1.PrintImage kdkImgEdit1.Page, kdkImgEdit1.Page, _
                    kdkImgAdmin1.PrintOutputFormat, kdkImgAdmin1.PrintAnnotations
                End If

            End If

        Case 1 'Exit
            'End the program
            End

    End Select
.
.
.
End Sub P
```

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. (Refer to Chapters 7 through 11 in this guide for more information.)

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 auto00001.tif, auto00002.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\auto00001.bmp
c:\claims\automobile\auto00002.bmp
c:\claims\automobile\auto00003.bmp
```

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

When scanning images using the TIFF or AWD file types — which do support 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 commences 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.

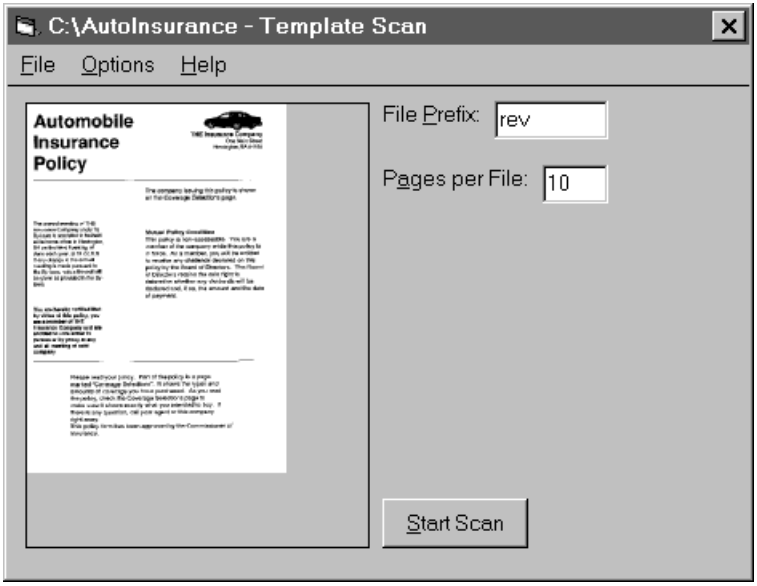
2. Creates the template by concatenating the path, a backslash, and the template prefix and by setting the **Image** property of the Image Admin control to the resulting string: *c:\employees\review*.
3. Sets the **FileType** property of the Image Admin control to *TIFF*.
4. Sets the **PageCount** property 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\review00001.tif
c:\employees\review00002.tif
c:\employees\review00003.tif
c:\employees\review00004.tif
c:\employees\review00005.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.

```
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!kdkImgScan1.FileType = TIFF
        frmMain!lblPages.Enabled = True
        frmMain!txtPages.Enabled = True
    ElseIf optBMP.Value Then
        frmMain!kdkImgScan1.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
    ElseIf optAWD.Value Then
        frmMain!kdkImgScan1.FileType = AWD_MicrosoftFax
        frmMain!lblPages.Enabled = True
        frmMain!txtPages.Enabled = True
    End If

    Unload Me

End Sub
```

- 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).

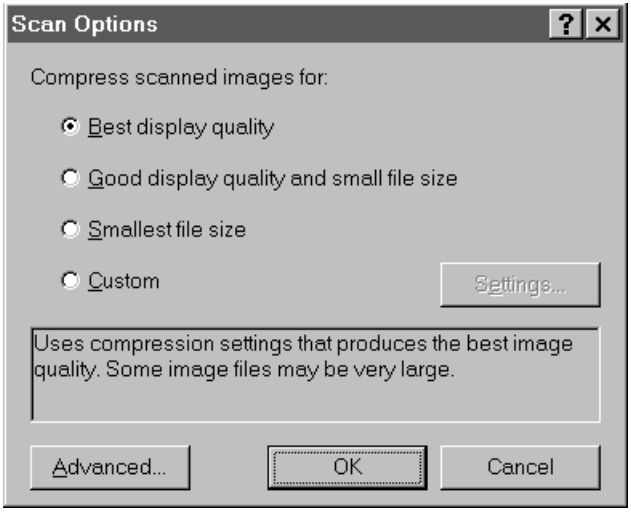
```
Private Sub cmdOK_Click()

    '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
```

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).

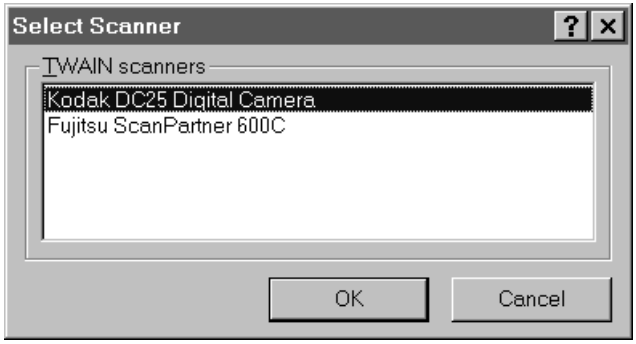
```
Private Sub cmdOK_Click()

    'Set the 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.



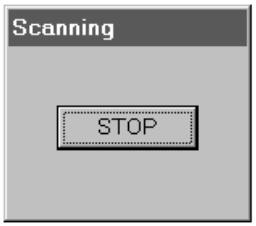

```
Private Sub mnuCompressionOptions_Click()

    'Invoke the ShowScanPreferences method which
    'displays the Scan Options dialog box
    kdkImgScan1.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 Scanner** 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
    kdkImgScan1.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
        kdkImgScan1.StopScanBox = False
    Else
        mnuStopButton.Checked = True
        kdkImgScan1.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 may 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
    kdkImgScan1.DestImageControl = "kdkImgEdit1"

    'Set the ScanTo property to enable template scanning
    kdkImgScan1.ScanTo = DisplayAndUseFileTemplate

    'Concatenate the path, backslash, and template prefix. Then assign the
    'string to the Image property
    kdkImgScan1.Image = gstrFolder + "\" + txtPrefix.Text

    'Assign the Pages per File value to the PageCount property
    kdkImgScan1.PageCount = txtPages

    'Set the MultiPage property to enable multipage scanning
    kdkImgScan1.MultiPage = True

    'Commence scanning
    kdkImgScan1.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. (Refer to Chapters 7 through 11 in this guide for more information.)

Thumbnails Defined

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.

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.

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.

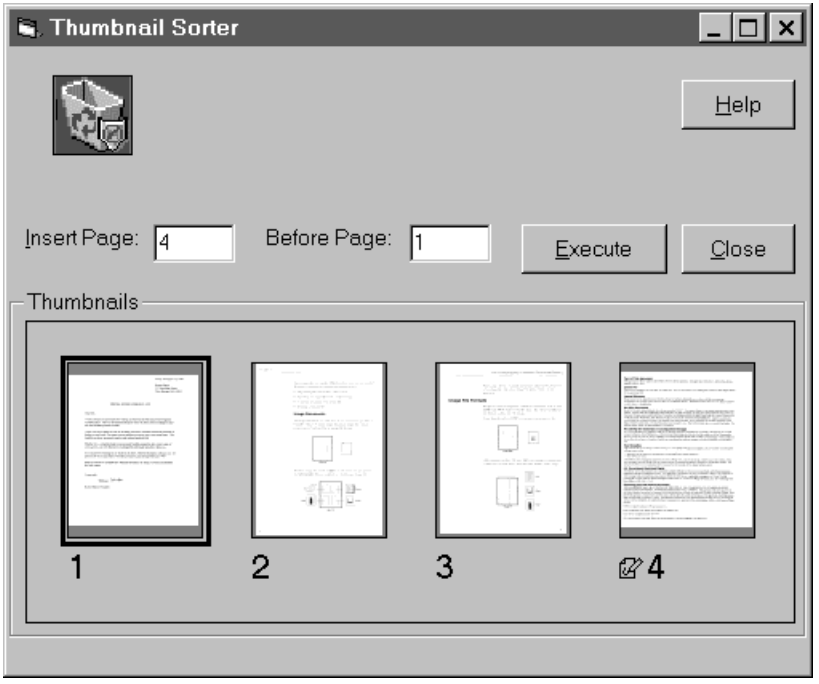
Thumbnail Sorter Project

The file name for the Thumbnail Sort project is Thumbnail.vbp.

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

Note: Only Imaging for Windows Professional Edition supports reorganizing image pages by dragging and dropping them *within* the 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.
- Sets the **AutoSelect** property of the Image Thumbnail control to `True` to have the control handle all thumbnail selections made using the mouse.

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

- 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
        kdkImgAdmin1.ShowFileDialog OpenDlg

    End If

    'Display an error message if the image file is not TIFF
    If kdkImgAdmin1.FileType <> 1 Then
        GoTo File_EH
    Else
        'Set the Image property of the Thumbnail control.
        kdkImgThumbnail.Image = kdkImgAdmin1.Image

        'Get the path of the application
        strPathName = App.Path

        'Display a thumbnail for each image page in the file
        kdkImgThumbnail.DisplayThumbs

        'Set AutoSelect to true to enable drag and drop, and
        'EnableDragDrop to DropFilesDropDragLeftRight (literal 15)
        kdkImgThumbnail.AutoSelect = True
        kdkImgThumbnail.EnableDragDrop = 15

        'Get the thumbnail page count
        mlngThumbCount = kdkImgThumbnail.ThumbCount

        'Set up an Image Thumbnail control as a trash bin
        'to provide a way to delete pages
        kdkImgThumbnailTrash.ThumbWidth = 50
        kdkImgThumbnailTrash.ThumbHeight = 50
        kdkImgThumbnailTrash.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 = kdkImgThumbnail.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
    kdkImgAdmin1.Insert strName, lngInsertPage, lngInsertBeforePage, 1

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

    'Delete the "leftover" page
    If strName = kdkImgThumbnail.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
        kdkImgAdmin1.DeletePages lngInsertPage, 1
        kdkImgThumbnail.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 kdkImgThumbnail_MouseDown(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single, _
ByVal ThumbNumberAs Long)

    'Invoke the Drag operation
    kdkImgThumbnail.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 kdkImgThumbnail_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 = kdkImgThumbnail.ThumbDropNames(X)
      'Get the Insert page
      lngInsertPage = kdkImgThumbnail.ThumbDropPages(X)

      'Check to see if the Insert page is to be inserted
      'before itself.  If it is, abort processing.
      If strName = kdkImgThumbnail.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
      kdkImgAdmin1.Insert strName, lngInsertPage, InsertBefore, 1

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

      'Delete the "leftover" page
      If strName = kdkImgThumbnail.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
          kdkImgAdmin1.DeletePages lngInsertPage, 1
          kdkImgThumbnail.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 kdkImgThumbnail_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
    kdkImgThumbnail.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 kdkImgThumbnailTrash_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 = kdkImgThumbnailTrash.ThumbDropNames(X)
        'Get the Delete page
        lngDeletePage = kdkImgThumbnailTrash.ThumbDropPages(X)

        'If page dropped from Explorer, exit the procedure
        If strName <> kdkImgThumbnail.Image Then
            Exit Sub
        Else
            'Delete the selected page from the image file and
            'the source Image Thumbnail control
            kdkImgAdmin1.DeletePages lngDeletePage - X, 1
            kdkImgThumbnail.DeleteThumbs lngDeletePage - X, 1
        End If
    Next X

End Sub
```

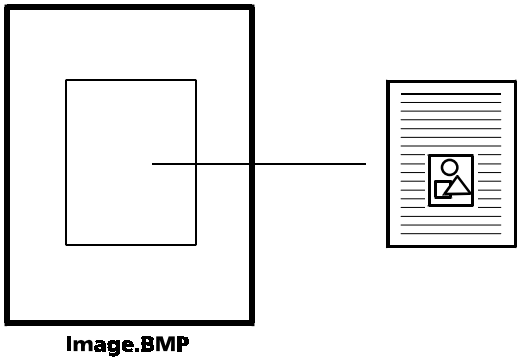
Unloading a Multipage Image File

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.

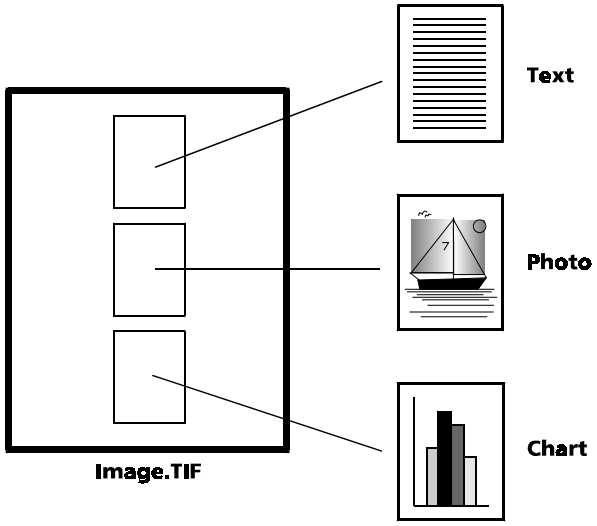
Refer to Chapters 7 through 11 in this guide for more information, particularly with respect to file-type support across the various versions of Imaging for Windows.

Multipage Image Files Defined

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



Other file types, such as AWD and 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
AWD[a]	Yes	Read/Write
BMP	No	Read/Write
DCX	Yes	Read Only
GIF	No	Read Only
JPG	No	Read Only
PCX	No	Read Only
TIFF	Yes	Read/Write
WIFF	Yes	Read Only

Multipage Support By File Type (continued)

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

a. Not available on Windows NT systems

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. (Refer to Chapters 7 through 11 in this guide for more specific information.)

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 name(s) of image pages dropped on the Thumbnail control.

ThumbDropPages property — Returns a list of pages for the file name(s) 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.

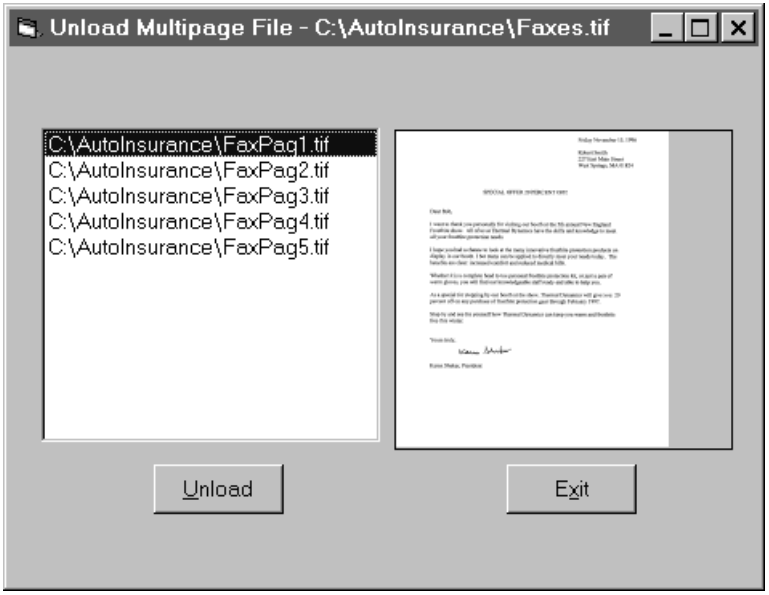
Sometimes Chris encounters pages that contain memos, letters, or forms that are of interest to different people but have no relationship with the remainder of the file.

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 if 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.

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.

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 **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 = kdkImgEdit1.Image
    strPrefix = kdkImgEdit1.Image

    'Get the number of pages in the source file
    intPageCount = kdkImgEdit1.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
        kdkImgAdmin1.Image = strUnloadedFileName
        intVerifyExistence = 0
        'If the Unloaded file exists, delete it
        If kdkImgAdmin1.VerifyImage(intVerifyExistence) = True Then
            kdkImgAdmin1.Delete strUnloadedFileName
        End If

        'Populate the Unloaded image file
        kdkImgAdmin1.Append strSourceFile, intPageNumber, 1

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

    Next intPageNumber

End Sub
```

5

Developing Client-Server Applications

This chapter explains how to use the Imaging ActiveX controls to develop applications that can access and interact with *Eastman Software* Imaging 1.x and Imaging 3.x 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

This section explains the basic concepts of Imaging 1.x and Imaging 3.x server access.

Developing applications that can access and interact with *Eastman Software* Imaging 1.x and Imaging 3.x servers is possible with the Imaging ActiveX controls. 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 server documents to Imaging 1.x servers.
- Read and display server documents from Imaging 3.x servers.
- Save 1.x image files and server documents and 3.x server documents to local and network drives.

Note: Imaging Server Access is available with Imaging for Windows Professional Edition V1.1 and greater.
To use the Imaging ActiveX controls with Imaging servers, you and your users must install and configure Imaging Server Access when installing Imaging for Windows Professional Edition.

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, Eastman Software, Inc. recommends that you also review the documentation that came with the server you and your users use.

File Type Support

The Imaging 1.x and Imaging 3.x 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 (read only)
	PCX (read only)
	WIFF (read only)
	XIF (read only)
Imaging 3.x	TIFF 4.0 (black and white only)

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 onto the desired server — either Imaging 1.x or Imaging 3.x.
- 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 Imaging 3.x servers by document name or field values to locate documents to open.
- Save 1.x image files and server documents and 3.x server documents to an Imaging 1.x file volume, an Imaging 1.x document volume, or a local or network drive.

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 Imaging 3.x Servers

When interacting with Imaging 3.x servers, your program can read documents from Imaging 3.x 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.

Imaging 3.x 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

This section describes the Imaging 1.x functions provided by the Imaging ActiveX controls.

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:

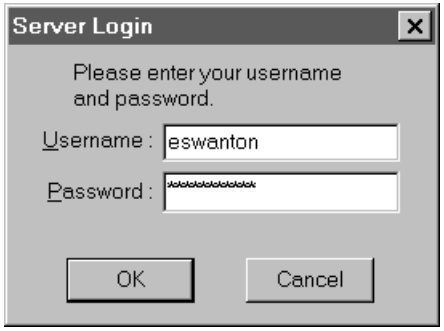
- Log onto the server.
- Set server options.
- Browse for Imaging 1.x file and document volumes.
- Browse the Imaging 1.x file and document volumes for files and documents to open.
- Query Imaging 1.x document volumes by document name, location, creation date, modification date, or keywords to locate 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 Onto the Server

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

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 onto the server.



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

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

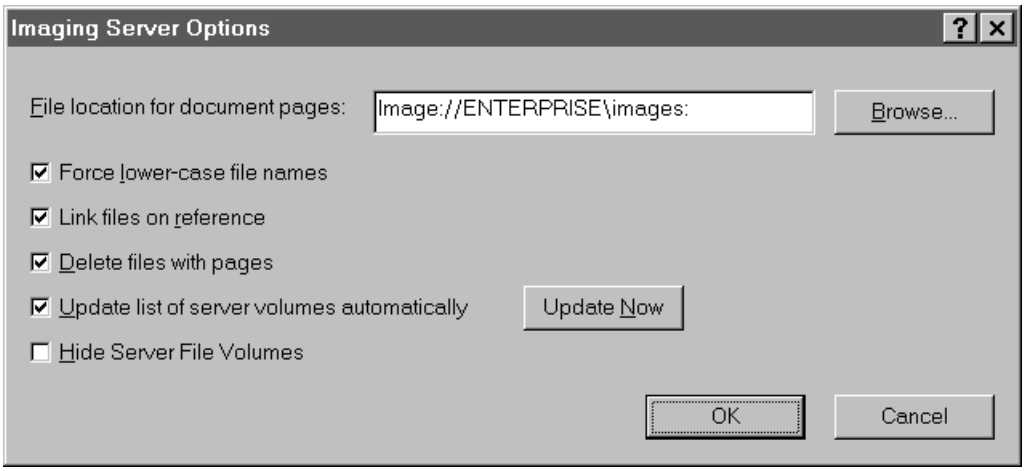
Setting Imaging Server Options

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 Adman 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 *new* or *copied* 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 server document *from an existing 1.x image file* are linked (True) or copied (False) by reference.
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.

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

As stated earlier, an Imaging 1.x server 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, allow your program to add pages to an Imaging 1.x server 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 server 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 a 1.x server 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 server 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 server 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 1x 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 server 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 **ForceFile Linking1x** 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

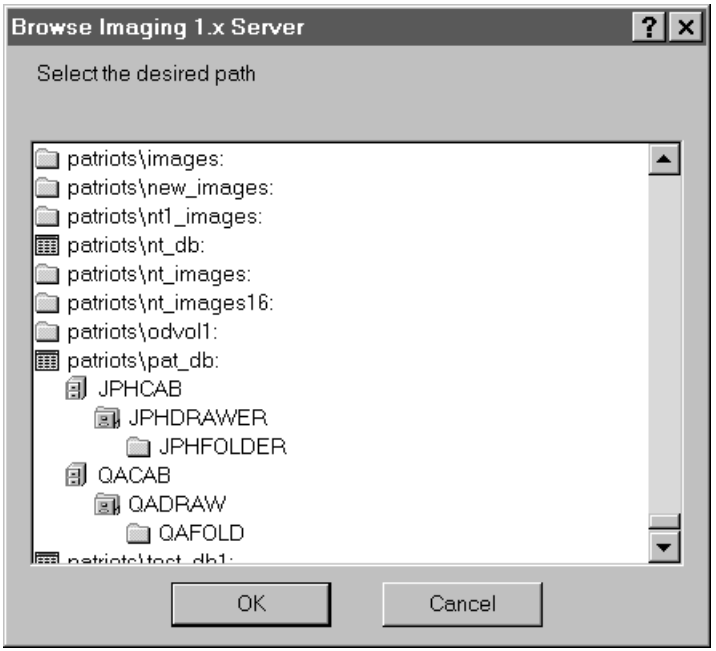
The Image Admin control provides two ways to browse Imaging 1.x servers. Using them, you can let your users:

- 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 Imaging 1.x Server" in the preceding figure).
- Another string that instructs the user to browse the server (for example, "Select the desired path" in the preceding figure).
- The handle to the parent window (optional).

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.

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.

Browsing for Files and Documents

As you know from Chapter 4, 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:

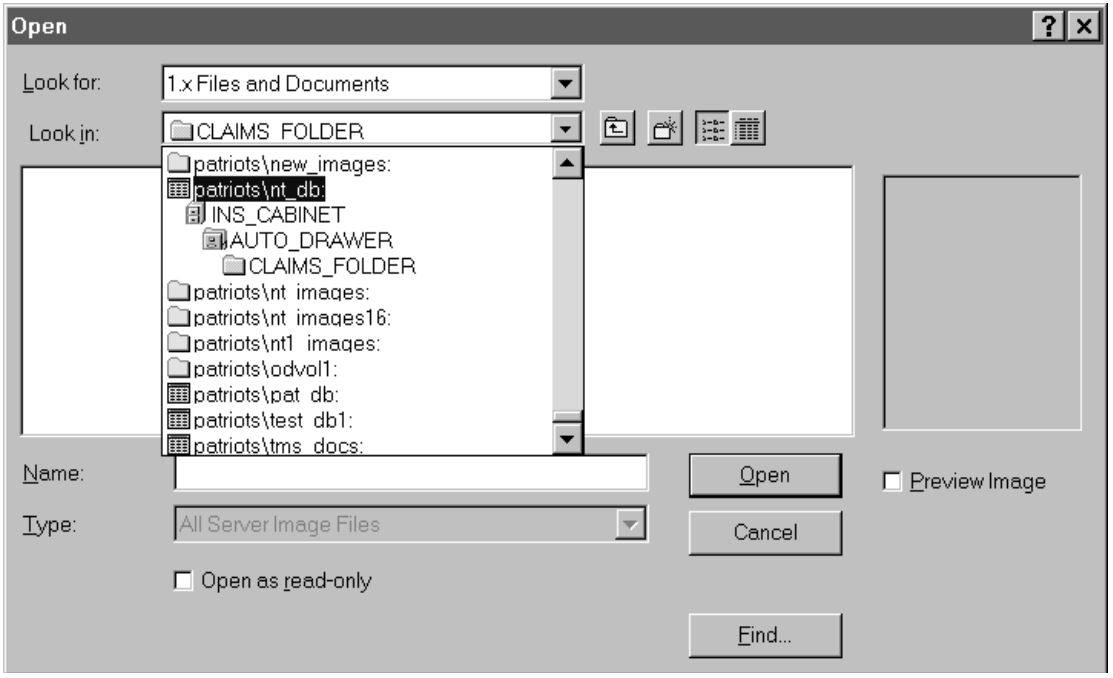
Desktop Files — The image files stored on local or network drives.

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

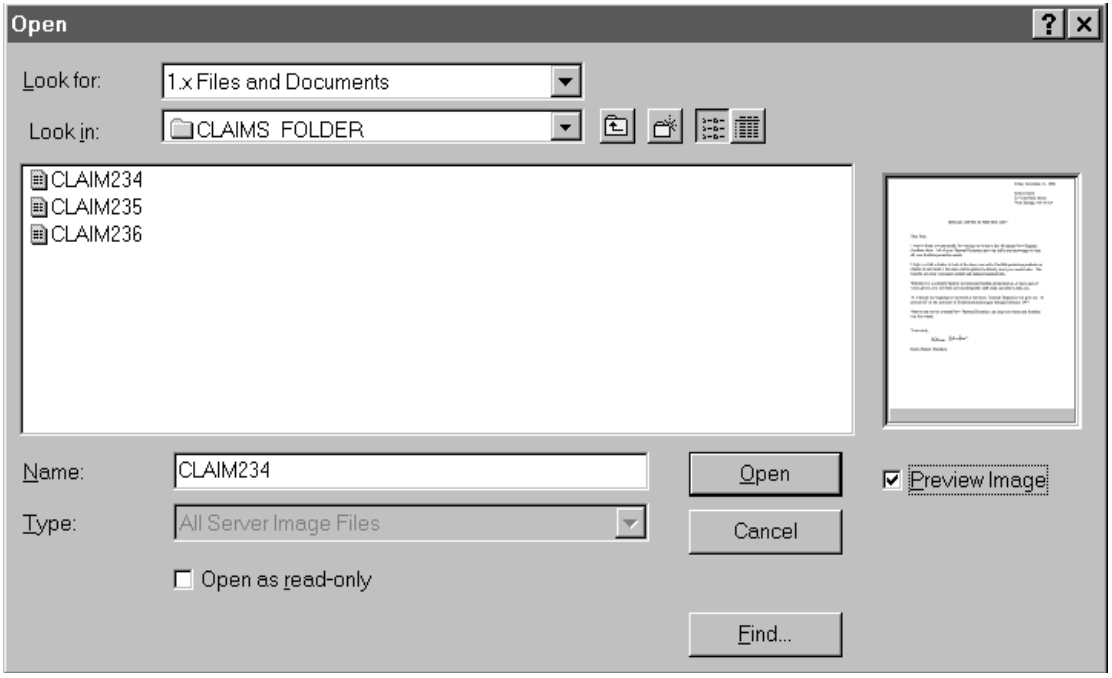
Invoke the **ShowFileDialog** method, therefore, whenever you want users to browse for and then select an Imaging 1.x file or server document they want to display.

The **Look for** list box also contains a 3.x 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 figure2



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.



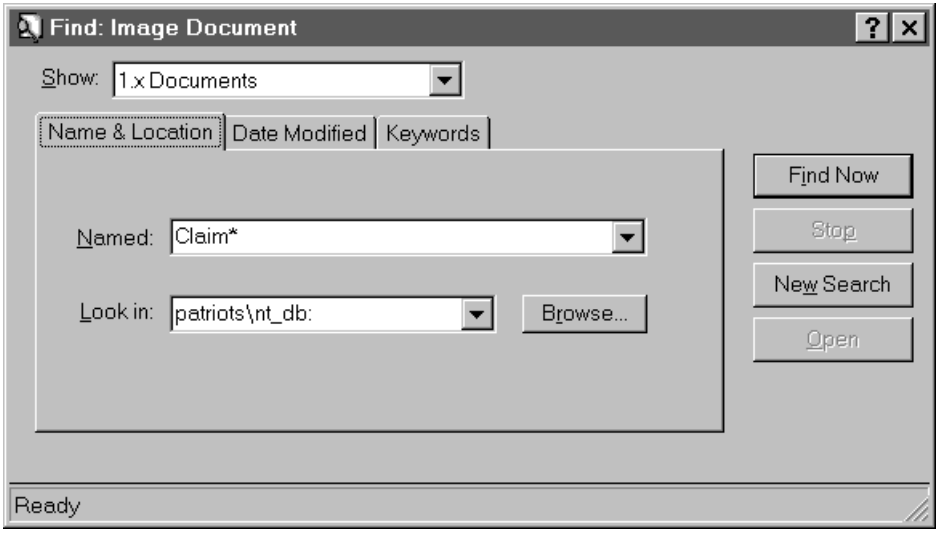
When users select the desired file or document and then click **Open**, the Imaging software writes the path and name of the file or document to the **Image** property of the Image Admin control.

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 Chapters 4, 8, and 11 for more information about displaying images and thumbnails as well as for more information about the **ShowFileDialog** method.

Querying for 1.x Server 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 **Look in** list box (as a default). If desired, 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*.

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

Note: Refer to the *Eastman Software Imaging for Windows Professional Edition Getting Started Guide* for instructions on how to use the **Find Image Document** dialog box to find Imaging 1.x server documents.

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.

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 server document.

Refer to Chapters 4, 8, and 11 for more information about displaying images and thumbnails and for more information about the **ShowFindDialog** method.

Saving 1.x Image Files and Server Documents

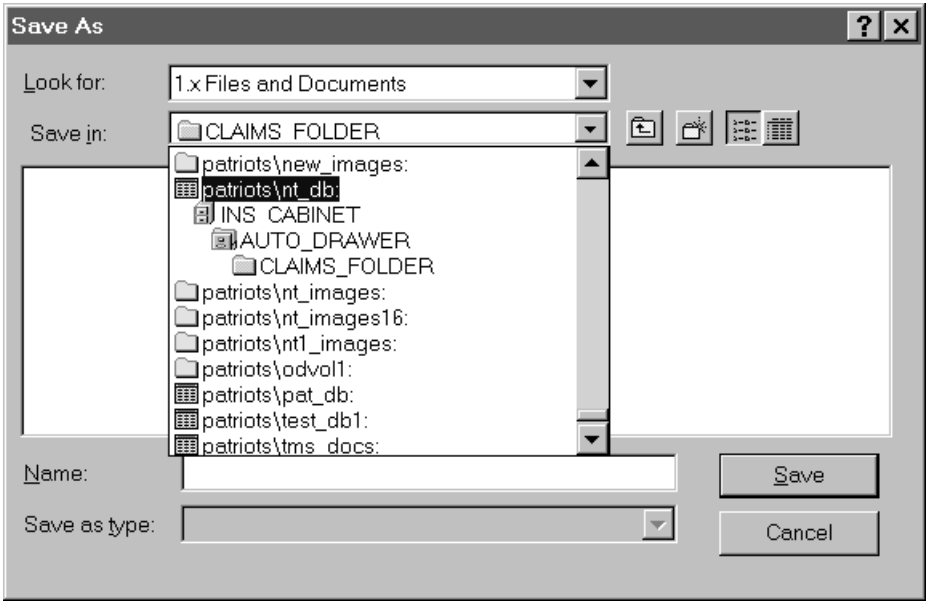
As you know, the **ShowFileDialog** method of the Image Admin control also displays a dialog box that lets users enter the path and name for the image files they want to save.

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:

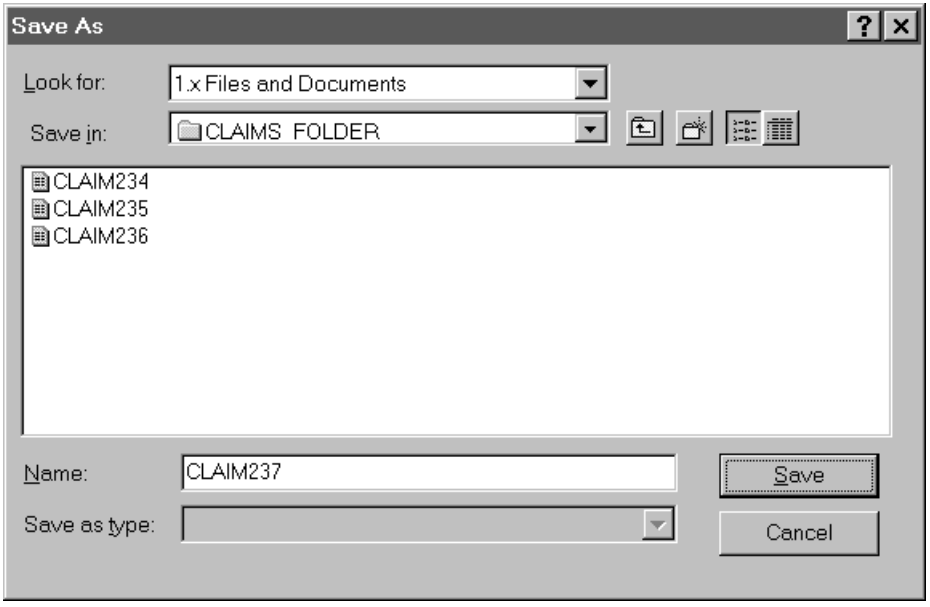
Desktop Files — Image files on their PCs.

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, 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 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.

Refer to Chapter 8 for more information about saving images and the **ShowFileDialog** method.

Imaging 3.x Server Programming Considerations

This section describes the Imaging 3.x functions that are provided by the Imaging ActiveX controls.

Several properties and methods in the Imaging ActiveX controls let you provide Imaging 3.x server access functions to your end users. Specifically, they permit your users to:

- Log onto the server.
- Query Imaging 3.x servers by document name as well as by Class and Index field values.

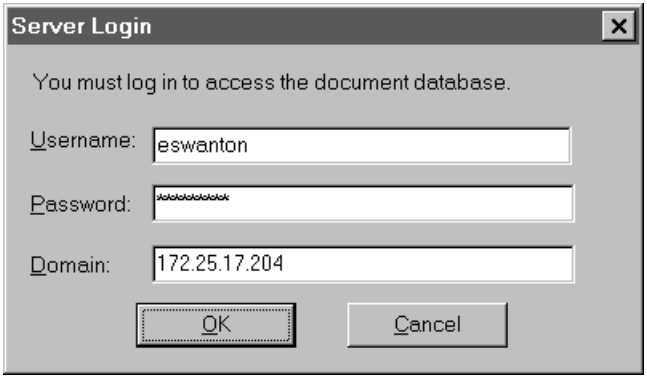
The following sections describe each function in detail by pointing out the properties and methods you can use.

Logging Onto the Server

The **Domain** property is initialized to either an empty string or the name of the Imaging 3.x server last accessed.

Before users can interact with an Imaging 3.x server, they must log onto it. The Image Admin control provides a **LoginToServer** method that lets you programmatically log your users onto the Imaging 3.x 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 onto the server.



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

The Image Admin control also provides a **LogOffServer** method that lets you programmatically log your users off an Imaging 3.x server.

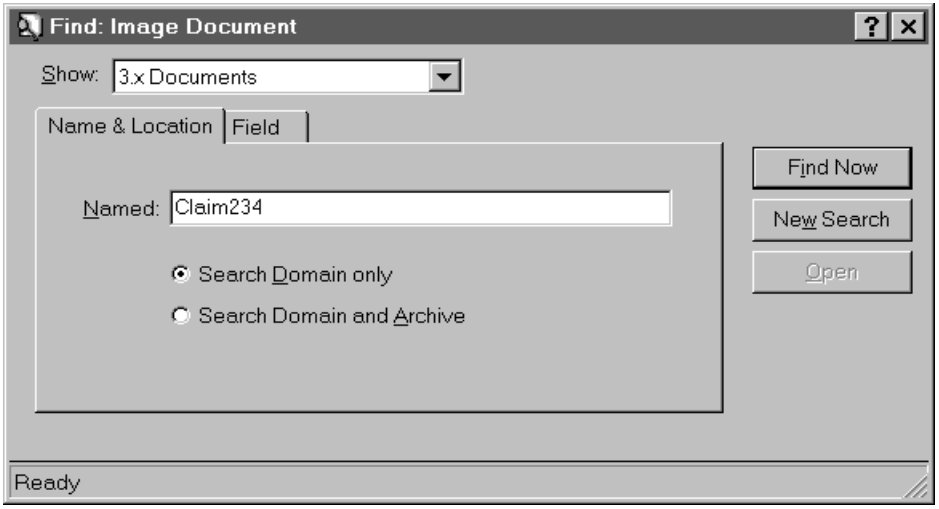
You can also bypass the dialog box and simply pass the user name and password as parameters to the **LoginToServer** method, most likely in response to a user completing and closing a logon dialog box of your own design.

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 onto the server.

Note: If the unified logon facility is enabled on the Imaging 3.x 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 3.x Server documents

The **ShowFindDialog** method of the Image Admin control displays a dialog box that enables users to query an Imaging 3.x server for the documents they want.



If users attempt to access Imaging 3.x 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*.

Note: Refer to the *Eastman Software Imaging for Windows Professional Edition Getting Started Guide* for instructions on how to use the **Find Image Document** dialog box to find Imaging 3.x server documents.

The **ImgQuery** and **ImgQueryEnd** methods of the Image Admin control enable you to query a 3.x server programmatically. The "Demonstration Project" section of this chapter describes and demonstrates these methods.

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 Imaging 3.x server document.

Refer to Chapters 4, 8, and 11 for more information about displaying images and thumbnails and for more information about the **ShowFindDialog** method.

Demonstration Project

This section demonstrates how to add Imaging 1.x server access functions to your image-enabled applications.

While a wide-ranging discussion of Imaging server access functions 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, Version 5.0.

Even if you are not going to include Imaging server access in your applications, you'll 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, Eastman Software, Inc. has prepared a demonstration project — called Image Server — that 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 on-line help system and Chapters 7 through 11 of this guide identify the properties, methods, events, parameters, and constants that are available in each version of Imaging for Windows.
Keep in mind that Imaging 1.x server access is available to users of Imaging for Windows Professional Edition Version 1.1 and greater.

Before walking through the demonstration project, read the following sections, which explain the concepts of zooming an image and of working with annotations. Chapter 4 of this guide explains the concepts of displaying an image in the Image Edit control and of multipage image files.

Zooming an Image Defined

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 4 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. The property supports a range of zoom factors from 2 to 6500 percent.

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 it can be seen more clearly, which 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

As described in Chapter 4, Eileen receives several scanned business documents in her role as product manager for a major computer company.

Because the documents contain important information, she really 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 Defined

The Image Edit and Image Annotation Tool Button controls provide several ways to add annotation functions to your 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.

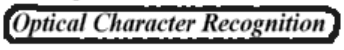
Annotations are digitized versions of the marks or items commonly applied to paper-based documents; for example, highlighting, rubber stamps, lines, and post-it notes. People typically use annotations to emphasize important portions of documents or to add their comments to documents being circulated 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 their attributes — such as color, size, text, and visibility.
- Use them to 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.

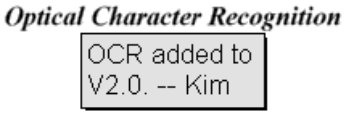
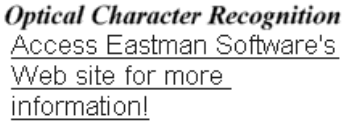
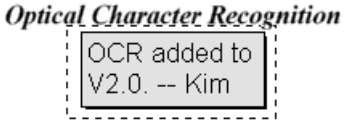
Imaging for Windows Annotation Types

Example	Annotation Type
	**Freehand Line** — Draws a free-hand line on a portion of an image for emphasis
	**Straight Line** — Underlines text, demarcates a section of a page, or draws callout lines on an image

Imaging for Windows Annotation Types (continued)

Example	Annotation Type
Optical [redacted] *Recognition*	**Filled Rectangle** — Covers a portion of an image (may be used to redact an image; that is, to permanently cover the parts of an image that you don't want other people to see)
Optical Character Recognition	**Hollow Rectangle** — Places a border around areas of an image for emphasis
Optical Character Recognition 	**Image Embedded** — Embeds a *copy* of another image (the tool button in the example) onto the displayed image
	Image Reference — Places another image onto the displayed image *by reference* (the image annotation is not a copy, it is a linked reference to an external file)
Optical Character Recognition Include OCR in next release! -- Geoff	**Text** — Enters text directly onto an image (the directive from Geoff in the example)
	Text From File — Enters text from a file onto an image
Optical Character Recognition Include OCR in next release! -- Geoff Approved 7/22/98	**Text Stamp** — Places a rubber stamp text annotation directly onto an image (the "Approved 7/22/98" stamp in the example)

Imaging for Windows Annotation Types (continued)

Example	Annotation Type
	**Attach-a-Note** — Enters text onto a Filled Rectangle annotation and then places it onto an image (the note from Kim in the example)
	**Hyperlink** — Enters a hypertext link directly onto an image; invokes the **Link To** dialog box, which lets users specify the desired destination (an image page or Web page as in the example)
	**OCR Zone** — Draws an OCR Text or Picture zone on an image. When drawn, the OCR engine recognizes only the area indicated by Text zones (1 in the example) and retains only those graphics indicated by Picture zones (2 in the example).
	**Select Annotations** — Selects existing annotation marks for copying, deleting, modifying, moving, or resizing (the dotted line surrounding the Attach-a-Note annotation in the example)

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

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 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.

Note: Hyperlink and OCR Zone annotations are available with Imaging for Windows Professional Edition.

Image Annotation Tool Button Control

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

Your custom annotation tool bar 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.

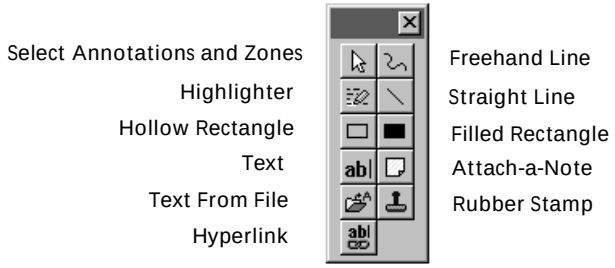
The Image Edit control shares several properties and one method with the Image Annotation Tool Button control. Together they let you manage annotation functions when creating a tool bar or palette of your own design. Refer to the next section for more information.

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 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**.

The remainder of this section briefly describes the properties, methods, and events of the Image Edit and Image Annotation Tool Button controls you'll find useful when adding annotation functions to your applications. (Refer to Chapters 7 through 11 in this guide for more information — particularly with respect to the properties, methods, events, and annotation types that are available with the version of Imaging for Windows you and your users are running.)

Properties, Methods, and Events of Both the Image Edit Control and the Image Annotation Tool Button Control

AnnotationBackColor property — Returns or sets the background color of an Attach-a-Note annotation.

AnnotationFillColor property — Returns or sets the color used to fill a Filled Rectangle annotation.

AnnotationFillStyle property — Returns or sets the pattern used to fill Image Embedded, Image Reference, and Filled Rectangle annotations.

AnnotationFont property — Returns or sets font object properties for all text-related annotation types.

AnnotationFontColor property — Returns or sets the font color for all text-related annotation types.

AnnotationImage property — Returns or sets the fully-qualified file name of the image file used in Image Embedded and Image Reference annotations.

AnnotationLineColor property — Returns or sets the line color for Straight Line, Freehand Line, and Hollow Rectangle annotations.

AnnotationLineStyle property — Returns or sets the line style for Straight Line, Freehand Line, and Hollow Rectangle annotations.

AnnotationLineWidth property — Returns or sets the line width for Straight Line, Freehand Line, and Hollow Rectangle annotations.

AnnotationStampText property — Returns or sets the stamp text to be placed on an image by the Text Stamp annotation type. The stamp text can consist of text, text macros (like the current date and time), or a combination of both.

AnnotationTextFile property — Returns or sets the fully-qualified file name of the text file to be placed on an image by the Text From File annotation type.

AnnotationType property — Returns or sets the type of annotation to draw.

Draw method — Draws the annotation.

Remaining Properties, Methods, and Events of the Image Edit Control

AnnotationGroupCount property — Returns the number of annotation groups that are on an image page.

AnnotationOcrType property — Returns or sets the type of OCR zone to be drawn on an image page.

OcrZoneVisibility property — Determines the visibility of OCR zones on the image page.

AddAnnotationGroup method — Adds a new annotation group to an image page.

BurnInAnnotations method — Burns annotations onto an image page, permanently incorporating them into the image.

Use the **BurnInAnnotations** method with care because once annotations are burned-in, they cannot be removed or modified as annotation data.

DeleteAnnotationGroup method — Deletes an annotation group and its associated annotations, and then redisplays the image.

DeleteSelectedAnnotations method — Deletes selected annotations from an image page.

EditSelectedAnnotationText method — Displays a dialog box that lets end users modify Text, Attach-a-Note, and Hyperlink annotations.

The **EditSelected AnnotationText** method is very useful. It enables users to modify text annotations. You should consider using it whether users draw annotations from the **standard annotation tool palette**, a tool palette of your own design, or from functions you provide programmatically.

ExecuteTextEditCommand method — Executes commands on the **Text Edit** dialog box when the Image Edit control is operating in the Text Edit mode (which is invoked using the **EditSelected AnnotationText** method).

EditingTextAnnotation event — Fires immediately after the Image Edit control enters or exits the Text Edit mode.

GetAnnotationGroup method — Returns the name of the annotation group based on the index specified.

GetAnnotationMarkCount method — Returns the number of annotation marks on an image page or in an annotation group.

For non-text annotations, use the **ShowAttribs Dialog** method. It enables users to modify non-text annotations.

GetCurrentAnnotationGroup method — Returns the name of the annotation group to which subsequent annotations will belong.

GetRubberStampItem method — Returns the item number of the currently selected rubber stamp according to its position on the **Rubber Stamp Properties** dialog box and in the shortcut menu of the **standard annotation tool palette**.

SetRubberStampItem method — Sets the item number for a rubber stamp annotation according to its position on the **Rubber Stamp Properties** dialog box and in the shortcut menu of the **standard annotation tool palette**. It also activates the Rubber Stamp annotation type.

GetRubberStampMenuItems method — Returns the menu items of the Rubber Stamp tool on the **standard annotation tool palette**.

ShowRubberStampDialog method — Shows the **Rubber Stamp Properties** dialog box, which permits end users to create, delete, and edit rubber stamp annotation properties.

GetSelectedAnnotationBackColor method — Returns the background color of a selected Attach-a-Note annotation.

SetSelectedAnnotationBackColor method — Sets the background color of a selected Attach-a-Note annotation.

GetSelectedAnnotationFillColor method — Returns the color used to fill a selected Filled Rectangle annotation.

SetSelectedAnnotationFillColor method — Sets the color used to fill a selected Filled Rectangle annotation.

GetSelectedAnnotationFillStyle method — Returns the style used to fill selected Image Embedded, Image Reference, and Filled Rectangle annotations.

SetSelectedAnnotationFillStyle method — Sets the style used to fill selected Image Embedded, Image Reference, and Filled Rectangle annotations.

GetSelectedAnnotationFont method — Returns font object properties for selected text-related annotation types.

SetSelectedAnnotationFont method — Sets font object properties for selected text-related annotation types.

GetSelectedAnnotationFontColor method — Returns the font color used in selected text-related annotation types.

SetSelectedAnnotationFontColor method — Sets the font color to use in selected text-related annotation types.

GetSelectedAnnotationImage method — Returns the fully-qualified file name of the image being used in selected Image Embedded and Image Reference annotations.

GetSelectedAnnotationLineColor method — Returns the line color used in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

SetSelectedAnnotationLineColor method — Sets the line color to use in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

GetSelectedAnnotationLineStyle method — Returns the line style used in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

SetSelectedAnnotationLineStyle method — Sets the line style to use in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

GetSelectedAnnotationLineWidth method — Returns the line width (in pixels) used in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

SetSelectedAnnotationLineWidth method — Sets the line width to use in selected Straight Line, Freehand Line, and Hollow Rectangle annotations.

GetSelectedAnnotationOcrType method — Returns the OCR type of a selected OCR zone annotation.

SetSelectedAnnotationOcrType method — Changes the OCR type of a selected OCR Zone annotation.

HideAnnotationGroup method — Hides the specified annotation group.

ShowAnnotationGroup method — Shows the specified annotation group.

HideAnnotationToolPalette method — Hides the standard annotation tool palette.

ToolPaletteHidden event — Fires immediately after the standard annotation tool palette is hidden.

ShowAnnotationToolPalette method — Shows the standard annotation tool palette.

SelectTool method — Selects an annotation tool from the standard annotation tool palette.

ToolSelected event — Fires immediately after the user selects a tool from the standard annotation tool palette.

LoadAnnotations method — Loads annotations from a file and places them on the displayed image page.

SaveAnnotations method — Saves annotations on the displayed image page to a file.

RemoveAllOCRMarks method — Removes OCR zones from all image pages in the displayed image document.

SelectAnnotationGroup method — Selects all annotation marks within a specific annotation group on an image page.

SelectFirstOcrZone method — Selects the first OCR zone on the displayed image page.

SelectNextOcrZone method — Selects the next OCR zone on the displayed image page.

SetCurrentAnnotationGroup method — Sets the annotation group to which subsequent annotations will belong.

ShowAttribsDialog method — Shows an annotation attributes dialog box, which lets end users change the properties of a selected annotation mark.

The **ShowAttribsDialog** method is very useful. It enables users to edit existing non-text annotations. You should consider using it whether users draw annotations from the **standard annotation tool palette**, a tool palette of your own design, or from functions you provide programmatically.

For text annotations, use the **EditSelectedAnnotationText** method. It enables users to modify text annotations.

HyperlinkGoToDoc event — Fires when users click a Hyperlink annotation that is linked to a page in an external image file.

HyperlinkGoToPage event — Fires when users click a Hyperlink annotation that is linked to a page within the current image file.

MarkEnd event — Fires immediately after the user or the program completes the drawing of an annotation mark.

MarkMove event — Fires immediately after the user or the program moves or resizes an annotation mark.

MarkSelect event — Fires immediately after the user or the program selects an annotation mark.

ToolTip event — Fires immediately after a tool tip is displayed on 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, 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.

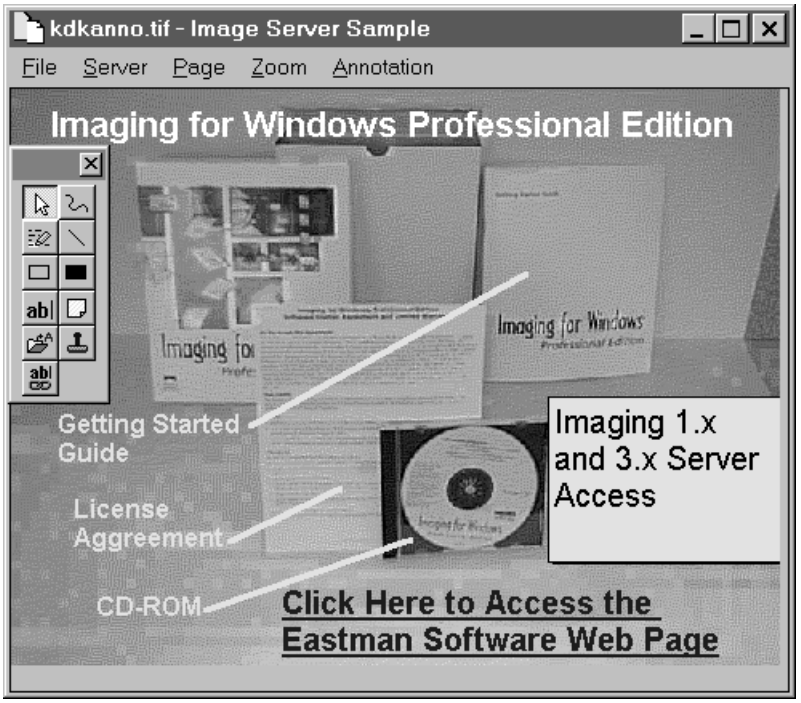
Because you included e-mail and image annotation functions in your program, Kim and her staff can use it to retrieve, annotate, and send the image documents. Analysts use its text-related annotation types to enter their review comments directly on the documents. And they use its Hyperlink annotation 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 `ImgServr.vbp`.

As stated previously, 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 PCs 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 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

            kdkImgAdmin1.Show1xServerOptDlg
    .
    .
    .
    End Select

End Sub
```

Browsing for Imaging 1.x File and/or 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.

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

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

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
            kdkImgAdmin1.Show1xServerOptDlg

        Case 2, 3, 4 'Browse 1.x File, Document, or File and Document Volumes
            If Index = 2 Then
                kdkImgAdmin1.Browse1x BrowseFiles, "Browse 1.x File Volumes", _
                    "Select the Desired Path", frmMain.hWnd
            ElseIf Index = 3 Then
                kdkImgAdmin1.Browse1x BrowseDocuments, _
                    "Browse 1.x Document Volumes", _
                    "Select the Desired Path", frmMain.hWnd
            Else
                kdkImgAdmin1.Browse1x 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 kdkImgAdmin1.Browse1xReturnedPath = "" Then  'Not installed.
                Exit Sub
            ElseIf kdkImgAdmin1.StatusCode <> 0 Then
                MsgBox Err.Description & " (ImgAdmin error " & _
                    Hex(kdkImgAdmin1.StatusCode) & ")", vbCritical
                Exit Sub
            End If

            strPath = kdkImgAdmin1.Browse1xReturnedPath
            strPathType = kdkImgAdmin1.Browse1xReturnedType

    .
    .
    .
    End Select

End Sub
```

Opening 1.x Files and Documents

To simply browse for and then open an Imaging 1.x image file or server 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(kdkImgAdmin1.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.
    '-------------------------------------------------------
    kdkImgAdmin1.Flags = 0
    kdkImgAdmin1.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 kdkImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(kdkImgAdmin1.StatusCode) & ")", vbCritical
        Exit Sub
    End If

    '-------------------------------------------------------
    ' Display the image.
    '-------------------------------------------------------
    Call PerformFileOpen(kdkImgAdmin1.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 on-line help as well as Chapter 8 of this guide 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
```

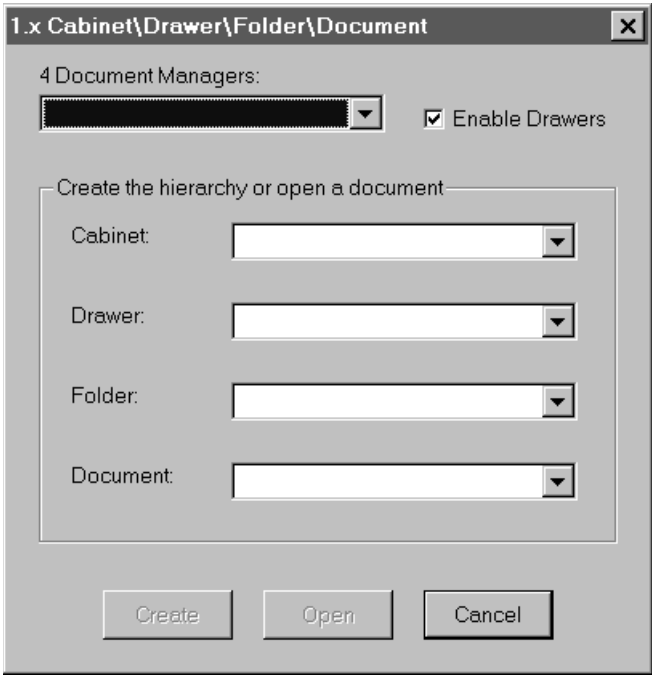
The query finds the available Imaging 1.x databases. Then the `Form_Load()` event procedure loads them from the `objResults`

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.
    '-------------------------------------------------------------------------
    kdkImgAdmin1.ImgQueryEnd
    kdkImgAdmin1.ImgQuery "DMVOLUMES", "", objResults
    '-------------------------------------------------------------------------
    ' If an error occurred, display a message box and exit.
    '-------------------------------------------------------------------------
    If kdkImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(kdkImgAdmin1.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
    kdkImgAdmin1.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"
    kdkImgAdmin1.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
    kdkImgAdmin1.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
```

Make 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
        kdkImgAdmin1.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
    kdkImgAdmin1.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
    kdkImgAdmin1.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
    kdkImgAdmin1.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
    kdkImgAdmin1.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
    kdkImgAdmin1.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.

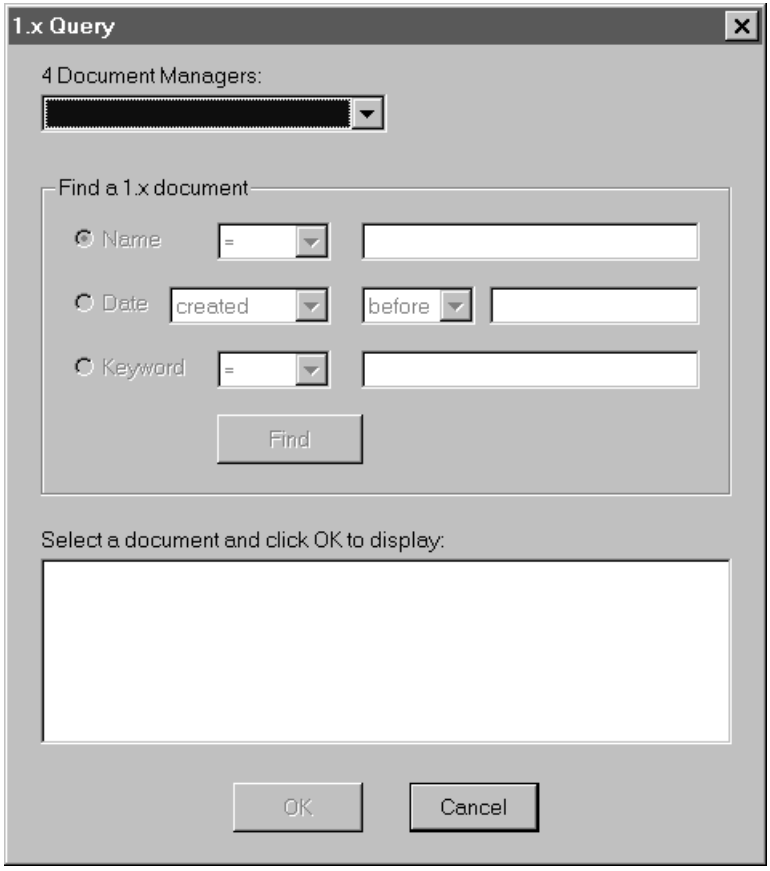
iDispatch parameter — The object variable, `objResults`, which represents the collection object that contains the results of the query.

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.



```
Private Sub Form_Load()
    Dim objResults As Object
    Dim strSinglePlural As String
    Dim vntItem As Variant
    .
    .
    .
    '-------------------------------------------------------------------------
    ' Perform an ImgQuery for all Document Manager databases.
    '-------------------------------------------------------------------------
    kdkImgAdmin1.ImgQueryEnd
    kdkImgAdmin1.ImgQuery "DMVOLUMES", "", objResults
    '-------------------------------------------------------------------------
    ' If an error occurred, display a message box and exit.
    '-------------------------------------------------------------------------
    If kdkImgAdmin1.StatusCode <> 0 Then
        MsgBox Err.Description & " (ImgAdmin error " & _
            Hex(kdkImgAdmin1.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
    kdkImgAdmin1.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:

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.

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.

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 the Regional Settings applet of Windows 95, 98, or NT.

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 server 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.)

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:

When building your own **QueryTerms** strings, be sure to include a space between each element.

- 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
```

The 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 = kdkImgAdmin1.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

    kdkImgAdmin1.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
    kdkImgAdmin1.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 (kdkImgAdmin1.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 redisplays 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%
            kdkImgEdit1.Zoom = 25
        Case 1  '50%
            kdkImgEdit1.Zoom = 50
        Case 2  '75%
            kdkImgEdit1.Zoom = 75
        Case 3  '100%
            kdkImgEdit1.Zoom = 100
        Case 4  '200%
            kdkImgEdit1.Zoom = 200
        Case 5  '400%
            kdkImgEdit1.Zoom = 400
    End Select
    '---------------------------------------------------------
    ' Check the menu item that was clicked.
    '---------------------------------------------------------
    mnuZoomFactorItem(Index).Checked = True
    '---------------------------------------------------------
    ' Refresh the image at the new zoom factor.
    '---------------------------------------------------------
    kdkImgEdit1.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**.

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 Defined” 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).

If desired, you can include parameters in your call to the **ShowAnnotationTool Palette** method. The parameters control:

- Whether users can set annotation properties.
- Where the tool palette will appear on the screen.
- The tool tip text the appears when the mouse pointer hovers over a button on the tool palette.

```
Private Sub mnuAnnotationItem_Click(Index As Integer)

    Select Case Index

        Case 0  'Show Annotations
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                kdkImgEdit1.HideAnnotationGroup
            Else
                mnuAnnotationItem(Index).Checked = True
                kdkImgEdit1.ShowAnnotationGroup
            End If

        Case 1  'Show Annotation Toolbar
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                kdkImgEdit1.HideAnnotationToolPalette
            Else
                mnuAnnotationItem(Index).Checked = True
                kdkImgEdit1.ShowAnnotationToolPalette
            End If

    End Select

End Sub
```

Showing and Hiding Annotations

Open an image file or server document. Then show the annotation tool palette and draw some annotations on the image.

On the **Annotations** menu, click **Show Annotations**.

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.

If the corresponding menu item is checked, the `Case 0` expression invokes the **HideAnnotationGroup** method of the Image Edit control, which hides all of the annotations on the image. (Refer to the "Annotations Defined" section earlier in this chapter for more information about annotations.)

If the corresponding menu item is not checked, the `Case 0` expression invokes the **ShowAnnotationGroup** method to show all of the annotations on the image I.

If desired, you can include the name of the specific annotation group to show or hide when calling the **ShowAnnotationGroup** or **HideAnnotationGroup** methods.

```
Private Sub mnuAnnotationItem_Click(Index As Integer)

    Select Case Index

        Case 0  'Show Annotations
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                kdkImgEdit1.HideAnnotationGroup
            Else
                mnuAnnotationItem(Index).Checked = True
                kdkImgEdit1.ShowAnnotationGroup
            End If

        Case 1  'Show Annotation Toolbar
            If mnuAnnotationItem(Index).Checked = True Then
                mnuAnnotationItem(Index).Checked = False
                kdkImgEdit1.HideAnnotationToolPalette
            Else
                mnuAnnotationItem(Index).Checked = True
                kdkImgEdit1.ShowAnnotationToolPalette
            End If

    End Select

End Sub
```

6

Image-Enabling Web Pages

This chapter explains how to use the Imaging ActiveX controls to image-enable your Web pages.

The chapter begins by showing you how to declare and define the Imaging ActiveX controls in a Java applet and HTML source file. It continues by explaining how to access the on-line help for the controls within Microsoft Visual J++. The chapter concludes by walking you through a sample demonstration project to help you get started.

In This Chapter

Image-Enabling a Web Page

This section explains how to use a Java applet and some HTML source code to image-enable a Web page. It assumes that you are using Microsoft Visual J++ to develop the applet.

Image-enabling a Web page involves a series of programming steps within two source files, specifically a:

- Java applet source file, and a
- HyperText Markup Language (HTML) source file.

These programming steps establish two-way communication between the code in the Java applet and the Imaging ActiveX controls on the Web page.

The code in the Java applet manipulates the controls on the Web page. Within your Java source code, you can set properties, invoke methods, and respond to events to provide a variety of Imaging functions to your Web page audience.

The code in the HTML file defines the controls as well as the Java applet on the Web page. Within your HTML source code, you pass control object references to the Java applet for processing.

Note: You can image-enable Web pages using Imaging for Windows Professional Edition and Imaging for Windows 98 only.

Java Applet

Before you can manipulate the Imaging ActiveX controls in Java, you must import their type libraries as Java classes and declare an object reference for each control. Then you must connect the logic in the Java applet to the controls on the Web page.

Importing Type Libraries

The process of importing type libraries creates Java classes that represent each Imaging ActiveX control within the applet.

You can use the Java Type Library Wizard within Visual J++ to search for the Imaging ActiveX control type libraries and create the Java interfaces for them. The interfaces allow you to access the properties, methods, and events of the controls.

Type libraries store information about COM objects like the Imaging ActiveX controls. This information can include classes, interfaces, dispinterfaces, etc.

A library is selected when a check mark appears next to it.

To import the Imaging ActiveX control type libraries

1. Start Visual J++ and create a new applet project.
2. On the **Tools** menu, click **Java Type Library Wizard**.

3 On the **Java Type Library Wizard** dialog box, select the type libraries that correspond to the Imaging ActiveX controls registered on your system.

The following list shows how the Imaging ActiveX controls appear on the wizard dialog box:

- Kodak Image Admin Control
- Kodak Image Edit Control
- Kodak Image Ocr Control
- Kodak Image Scan Control
- Kodak Image Thumbnail Control

Note: The name Wang appears instead of Kodak when using Imaging for Windows Professional Edition V1.0 and V1.1. (The Eastman Kodak Company acquired Wang Software, the software unit of Wang Laboratories, Inc. in 1997, thereby creating Eastman Software, Inc.)
The Image OCR control is available with Imaging for Windows Professional Edition only.

4 Click **OK** and look at the Visual J++ **Output** window. The window displays the Import statements for each Imaging ActiveX control interface the wizard has created.

The following table lists the import statements of each Imaging ActiveX control.

Imaging ActiveX Import Statements

Import Statement	Imaging ActiveX Control
import imgadmin.*;	Image Admin
import imgedit.*;	Image Annotation Tool Button
import imgedit.*;	Image Edit
import imgocr.*;	Image OCR
import imgscan.*;	Image Scan
import imgthumb.*;	Image Thumbnail
import olepro32.*;	(Include once for all controls)

The Imaging ActiveX controls use Variants as parameters. Importing the com.ms.com.Variant package creates Variant parameters.

5 Copy the Imaging ActiveX import statements into the Imports area of your Java source code.

6 In addition, include the following import statement in the Imports area:

```
Import com.ms.com.Variant
```

As an example, the following code snippet shows the required import statements for two Imaging ActiveX controls: Image Edit and Image Thumbnail.

```
//*****************************************
// myapplet.java: Applet
//*****************************************
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import imgedit.*;
import imgthumb.*;
import olepro32.*;
import com.ms.com.Variant;
```

7 To see a list of the properties, methods, events, and parameter types supported by each Imaging ActiveX control, click the appropriate `summary.txt` file inside the **Output** window.

The summary file you click appears within the Developer Studio. You can set the properties, call the methods, and respond to the events listed.

Note: The Java Type Library Wizard creates one subfolder within the Java trusted library subfolder — typically c:\windows\java\trustlib — for each Imaging ActiveX type library imported.
The wizard populates each subfolder with `.class` files, one for each COM class and/or interface described in the type library. All of the generated classes and interfaces are part of the Java package for the respective Imaging ActiveX type library.

Declaring Object References

Declaring an object reference for each control enables you to use that reference when accessing the properties, methods, or events of the control.

To declare object references for the Imaging ActiveX controls

- Within the Main class of your Java applet, declare an object reference for each Imaging ActiveX control you are going to use.

  The following table lists the class names of each Imaging ActiveX control.

Imaging ActiveX Class Names

Class Name	Imaging ActiveX Control
_DImgAdmin	Image Admin
__DImgAnnTool	Image Annotation Tool Button
__DImgEdit	Image Edit
__DImgocr	Image OCR
__DImgScan	Image Scan
__DImgThumbnail	Image Thumbnail

As an example, the following code snippet shows the proper object reference declarations for two Imaging ActiveX controls: Image Edit and Image Thumbnail.

```
//=====================================================
// Main Class for applet myapplet
//=====================================================
public class myapplet extends Applet implements Runnable
{

   ButtonbLoadImage;
   ButtonbRotate;
   ButtonbNextPage;
   ButtonbPrevPage;

   Component cLoadImage;
   Component cRotate;
   Component cNextPage;
   Component cPrevPage;

   intPageCount;

   _DImgEdit theImage;
   _DImgThumbnail theThumb;
   .
   .
   .
```

Connecting the Applet to the Controls

Java's **setControl** method enables you to save an object reference to each Imaging ActiveX control on the Web page. Saving references establishes a connection between the Java applet code and the controls on the Web page, making it possible for the controls to be manipulated by the Java applet.

Scripting code in the HTML file invokes the **setControl** method when the Web page loads. The invocation includes arguments that pass control object references to the Java applet.

To connect the Imaging ActiveX controls

1. Include the **setControl** method in the applet source code.
2. Include code that saves each control reference passed.

   The following code snippet shows saving a reference to the Image Thumbnail control.

```
public void setControl(Object thumbctl)
{
   // save the thumb nail control passed in
   theThumb = (_DImgThumbnail) thumbctl;
}
```

HTML File

To complete the process of image-enabling your Web page, you must define the Java applet and the Imaging ActiveX controls in an HTML file. Then you must invoke the **setControl** method in the Java applet to complete the connection between the Java applet and the controls on the Web page.

Defining the Applet and the Controls

The process of defining the applet and the Imaging ActiveX controls enables you to include them on your Web page.

To define an applet and 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>, etc.
3. Define the Java applet by including the <applet> command element and setting its properties.

   As a minimum, the applet definition must contain the name of the applet class (code attribute) as well as its initial dimensions in pixels (width, height attributes). It can also contain other attributes, such as the name of the applet (name attribute) and the alignment of the applet window (align attribute).

An applet 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 applet.

```
<applet
   code=myapplet.class
   name=myapplet
   width=425
   height=40
   align = top>
   <param name=image value="myimage.tif">
</applet>
```

You can use the Visual J++ OLE/COM Object Viewer to see registry information for each Imaging ActiveX control. You can also use the viewer to copy the class identifier and the HTML object definition to the Clipboard.

4 Define each Imaging ActiveX control you want to use by including the <object> command element and setting its properties.

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.

```
<object
   classid="clsid:E1A6B8A0-3603-101C-AC6E-040224009C02"
   id=imgthumbctrl
   width=140
   height=256
   align=left
>
<param name="BackColor" value="#ffffff">
</object>
```

The following table lists the class identifiers of each Imaging ActiveX control.

Imaging ActiveX Class Identifiers

Class 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
{84926CA0-2941-101C-816F-0E6013114B7F}	Image Scan
{E1A6B8A0-3603-101C-AC6E-040224009C02}	Image Thumbnail

Completing the Connection

Earlier, you included the **setControl** method in your Java applet to connect the logic in the Java applet to the controls on the Web page. Now you must invoke the **setControl** method with the appropriate parameters in order for it to perform its work.

You need to include some scripting code within your HTML file that performs this task. The scripting code invokes the **setControl** method in the Java applet, passing to it an object reference to each Imaging ActiveX control that has been defined in the HTML file.

Once invoked, the **setControl** method saves each object reference passed, thereby completing the connection between the Java applet and the controls on the Web page. You can now manipulate each control from within your Java applet.

To complete the connection

1. Use VBScript (or some other scripting language) to invoke the **setControl** method when the _onLoad event fires.
2. Within the call to **setControl**, pass an object reference to each Imaging ActiveX control defined in the HTML file.

   The following code snippet shows passing an object reference to the Image Thumbnail control.

The window_onLoad event fires immediately after the browser finishes loading a window or all of the frames within a <FRAMESET> tag.

```
<script language=VBScript>
<!--
sub Window_OnLoad
   document.imgctls.setControl imgthumbctrl
end sub
-->
</script>
```

Obtaining Help

This section explains how to access the on-line help system of the Imaging ActiveX controls.

You can access the Imaging ActiveX Controls on-line help system within Visual J++ or directly from Windows Explorer.

Note: Several methods in the Imaging ActiveX controls present dialog boxes to the end user. Each dialog box provides its own context-sensitive help, which the user can invoke by clicking the question mark at the top of the dialog box and then the desired control.

Visual J++

Imaging ActiveX help is accessible from the **Properties** window when you are editing a dialog box resource.

To access Imaging ActiveX help from the Properties window

1. Insert an Imaging ActiveX control onto a dialog box as you would any other type of ActiveX control. Make sure the control has the focus.
2. On the **View** menu, click **Properties**. The **Properties** window appears.
3. Click the desired property in the **Properties** window and then press F1. The help topic for the selected property appears.

Note: You can navigate to the other topics of the help system by clicking the **Help Topics** button.
Keep in mind that it is only the design-time properties that appear in the **Properties** window.
If the property you select is an extender property, the contents window for the Imaging ActiveX help system appears.

Windows Explorer

Access the Imaging ActiveX help system from Explorer when you cannot access it within Visual J++.

To access Imaging ActiveX help from Windows Explorer

1. Navigate to the folder where you installed Windows.
2. Open the System subfolder.
3. Click the `imgocxd.hlp` file. The contents window for the Imaging ActiveX help system appears.

   From the contents window, you can navigate to the topics that describe the properties, methods, and events of the controls.

Note: When using Imaging for Windows Professional Edition V1.0 and V1.1, the name of the help file is `wangocxd.hlp`.

Demonstration Project

This section demonstrates how to use the Imaging ActiveX controls within a Java applet and HTML source file to image-enable a Web page.

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

The demonstration applet — along with its companion HTML code — was developed using Microsoft Visual J++, Version 1.1.

To help you use the Imaging ActiveX controls to image-enable a Web page, Eastman Software, Inc. has prepared a demonstration project — called Imgctls — that shows you how to:

- Display an image file in an Image Edit control and in an Image Thumbnail control.
- Rotate an image page.
- Navigate the pages of a multipage image file.

Note: The on-line help system and Chapters 7 through 11 of this guide identify the properties, methods, events, parameters, and constants that are available in each version of Imaging for Windows.
Keep in mind that image-enabling a Web page is available to users of Imaging for Windows Professional Edition and Imaging for Windows 98 only.

Before walking through the demonstration project, read the following sections, which explain the concepts of rotating an image and working with thumbnail captions. Chapter 4 in this guide explains the concepts of displaying an image in the Image Edit and Image Thumbnail controls and of multipage image files.

Rotating an Image Defined

The Image Edit control provides several methods that let you add image rotation functions to your image-enabled applet or application.

Most image application developers make rotation functions available to their end users. Doing so lets users rotate one or more image pages to the right or to the left so the pages may be viewed in their proper orientation.

You can also use the **Flip** method to rotate an image page by 180 degrees.

The following list briefly describes the methods you'll find useful when including rotation functions in your applet or application (refer to Chapters 7 through 11 in this guide for more specific information).

Rotate — Displays a dialog box that lets end users rotate one or all of the pages in an image document file. End users can specify the degree of rotation applied as well as the direction.

Rotate All — Rotates all of the pages in an image document file to the degree and in the direction specified and then saves the file.

Note: The **Rotate** method is available with all versions of Imaging for Windows Professional Edition. The **RotateAll** method is available with Imaging for Windows Professional Edition Version 2.0 only.

Rotate Right — Rotates an image page 90 degrees to the right, or to the degree specified.

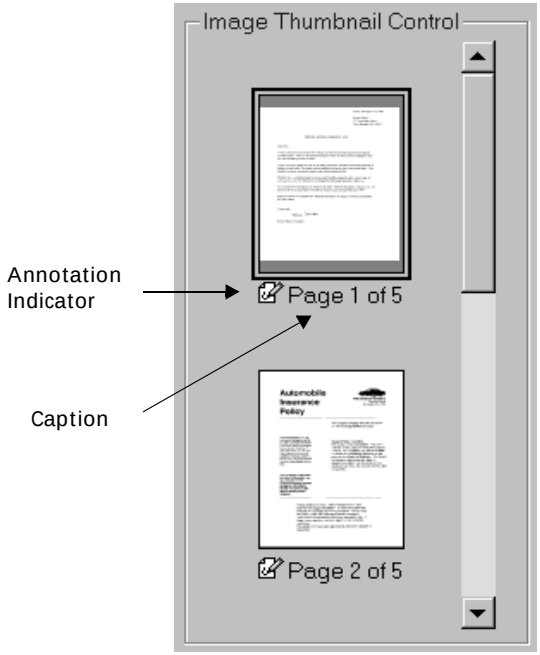
Rotate Left — Rotates an image page 90 degrees to the left, or to the degree specified.

Note: Specifying a rotation other than 90 degrees is available with Imaging for Windows Professional Edition and Imaging for Windows 98 only.

Thumbnail Captions Defined

As you know from reading Chapter 4, the Image Thumbnail control lets users view each page of an image document file in miniature boxes called thumbnails — there is one thumbnail image for each page in the file.

You may not know that each thumbnail image can optionally have a caption beneath it that indicates its page position within the image file as well as an annotation indicator that shows whether one or more annotation marks exist on a corresponding image page.



The caption can be customized if desired (as was done in the preceding figure). You can use the Image Thumbnail control's reserved symbols — such as the number sign (#) and the asterisk (*) — to represent each page number as well as the total number of pages in the image file.

For example, you can display captions such as:

- Page 1 of 20
- Page 2 of 20
- Page 3 of 20

by specifying the following custom string:

```
Page # of *
```

As you might guess, the Image Thumbnail control provides several properties that let you manage captions. Customizing and formatting captions can add a nice, professional appearance to your image-enabled applets and applications.

The following list briefly describes the properties you'll find useful when working with thumbnail captions (refer to Chapters 7 through 11 in this guide for more specific information).

ThumbCaptionColor — Sets the text color of thumbnail captions.

ThumbCaptionFont — Returns a font object, or sets the font properties of thumbnail captions.

ThumbCaption — Sets an optional custom caption string.

ThumbCaptionStyle — Determines the type of caption that appears.

Example

Users of your applet (or application for that matter) may want to rotate an image page to apply the proper orientation.

They may also want to draw attention to the pages of an image file that have been annotated with important comments so their readers won't miss important information.

Scenario

Assume Amy uses one of your applications to scan several technical documents in her role as a mechanical engineer for a major aeronautics company. The letter-size, technical documents contain descriptions and drawings of flight surface components.

The descriptions appear in portrait orientation, while the drawings appear in landscape orientation. Because all of the pages in the document are the same size, Amy scans them together using her automatic document feeder (ADF)-equipped scanner.

After each document has been scanned, Amy annotates several image pages with her technical comments. Then she publishes the documents to the company's intranet page so employees in the field can view them using Internet Explorer and your image-enabled Java applet.

Because you included Image Edit's **Rotate** method in your applet, users can rotate the drawings by 90 degrees to make them appear right-side up. And they can rotate a description page by 180 degrees to correct those rare occasions when Amy inadvertently scanned a page upside down.

Further, because you included an Image Thumbnail control in your applet and you set it to indicate when annotations are present, readers can easily spot the pages that contain Amy's annotated comments.

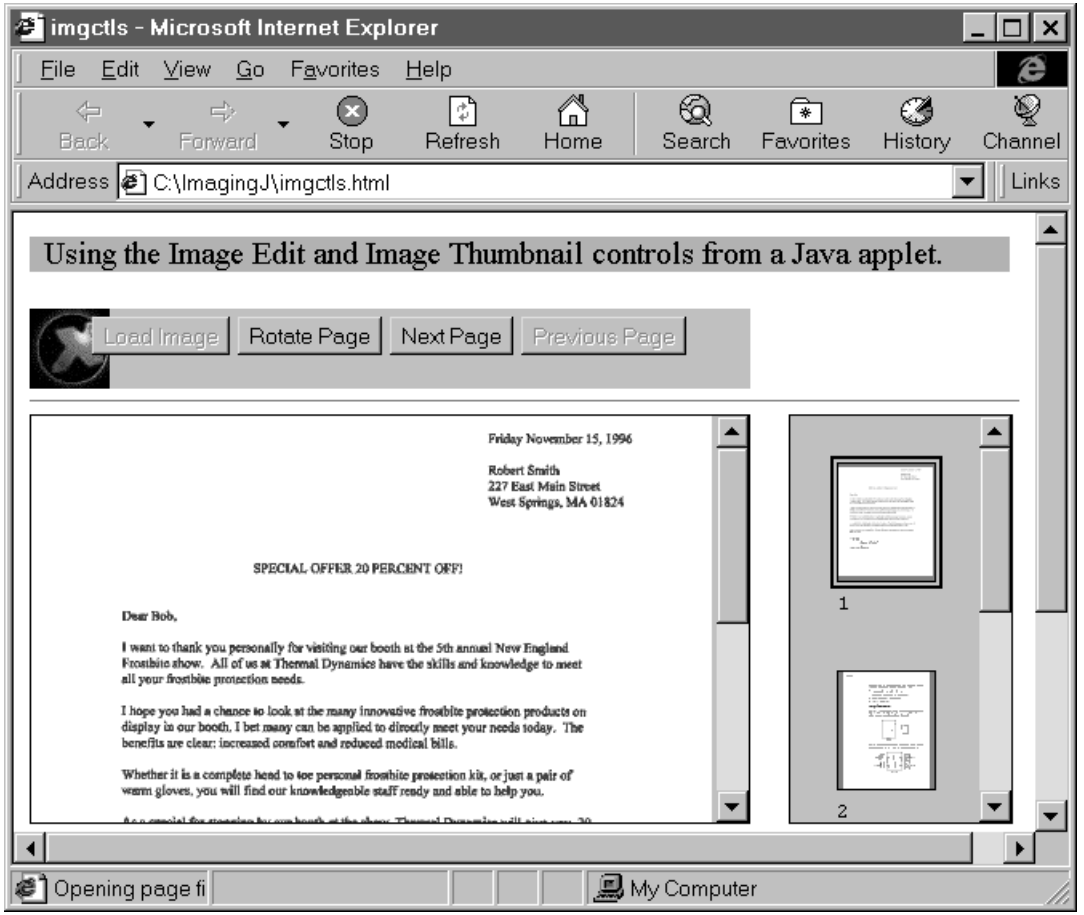
The Imgctrls Project

The file name for the demonstration project is `Imgctls.dsw`.

As mentioned earlier, the Imgctrls project demonstrates:

- Displaying a multipage image document file on a Web page.
- Rotating an image page 90 degrees to the right.
- Navigating the image document file.

The following figure shows the Imgctls project running in Microsoft Internet Explorer.



Project Overview

The Imgctrl project consists of the following source files:

Imgctls.html — Contains the HTML and VBScript code.

Imgctls.java — Contains the Java applet code.

And it consists of the following components:

- One Image Edit control
- One Image Thumbnail control
- Four Button objects

The project uses the following methods in the Image Edit and Image Thumbnail controls to provide the display, rotate, and navigation functions:

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

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

RotateRight (Image Edit control) — Rotates the image 90 degrees to the right.

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

Note: The Imgctls Java applet is multithreaded. Its executes a marquee in addition to its Imaging functions. Because it is beyond the scope of this book to discuss multithreading, any code associated with it is ignored. If necessary, refer to your Java documentation for information about multithreading.

Starting Imgctls

Start the Imgctls project by opening the `Imgctls.html` file in Internet Explorer. The browser begins loading the code.

In the Browser

First, the code within `Imgctls.html` directs the browser to define the Imgctls applet. Once the browser defines the applet, it begins executing. Then the code directs the browser to define the Image Edit control and the Image Thumbnail control.

The applet definition contains the code, name, width, height, and align attributes for the Imgctls Java applet, as shown in the following code snippet. (Note that the code passes no values to the applet via the param element.)

```
<applet
    code=imgctls.class
    name=imgctls
    width=450
    height=50
    align = top>
    <param name=image value="">
</applet>
```

Each Imaging ActiveX control object definition contains the classid, id, width, height, align, and hspace attributes for the Image Edit and Image Thumbnail controls, as shown in the following code snippet.

```
<object
   classid="clsid:6D940280-9F11-11CE-83FD-02608C3EC08A"
   id=imgeditctrl
   width=450
   height=256
   align=left
>
</object>

<object
   classid="clsid:E1A6B8A0-3603-101C-AC6E-040224009C02"
   id=imgthumbctrl
   width=140
   height=256
   align=right
   hspace=24
>
</object>
```

When the browser finishes loading the HTML code, its **_onLoad** event fires. VBScript within the **_onLoad** event invokes the **setControl** method in the now-running applet, passing to it references to the Image Edit and Image Thumbnail controls that were defined earlier.

```
<script language=VBScript>
<!--
sub Window_OnLoad
   document.imgctls.setControl imgeditctrl, imgthumbctrl
end sub
-->
</script>
```

In the Applet

Once the browser defined the Imgctls applet, it began executing. By convention, whenever an applet executes, it invokes several methods in series: its constructor, **init()**, **paint()**, and **start()**.

First, the Imgctls applet invokes the **imgctls()** method, which is its constructor. Since there are no initialization tasks to perform there, nothing happens.

Next, the applet invokes the **Init()** method, which starts the execution of the marquee's thread; and then the **paint()** method, which draws the contents of the applet window on the Web page and displays an animation.

```
public void paint(Graphics g)
{
   // ANIMATION SUPPORT:
   // The following code displays a status message until
   // all the images are loaded. Then it calls displayImage
   // to display the current image.
   //-----------------------------------------------------
   if (m_fAllLoaded)
   {
      Rectangle r = g.getClipRect();
      g.clearRect(r.x, r.y, r.width, r.height);
      displayImageAnimation(g);
   }
   // TODO: Place additional applet Paint code here
}
```

As soon as the applet window becomes visible on the Web page, the applet invokes the **start()** method. Code within the **start()** method starts the execution of the Imgctls applet's thread.

```
public void start()
{
   if (m_imgctls == null)
   {
      m_imgctls = new Thread(this);
      m_imgctls.start();
   }
}
```

When the browser finishes loading the HTML source code, its **_onLoad** event fires. VBScript within the **_onLoad** event invokes the **setControl** method in the applet, passing to it references to the Image Edit and Image Thumbnail controls defined in the HTML source code.

The **setControl** method saves these references. This action completes the connection between the Imgctls applet and the HTML source code, enabling the Imgctls applet to manipulate the Image Edit and Image Thumbnail controls on the Web page.

```
// Accept the image and thumbnail controls passed from
// Window_OnLoad
public void setControl(Object imageCtl, Object thumbctl)
{

   // save the image control passed in
   theImage = (_DImgEdit) imageCtl;

   // save the thumb nail control passed in
   theThumb = (_DImgThumbnail) thumbctl;

}
```

With the functioning Imgctls applet now on your browser, you can begin using it.

Loading an Image

To load an image, click the **Load Image** button on the applet window.

The **action** method evaluates the `target` argument to determine which button you clicked. When you click the **Load Image** button, the code invokes the **LoadImage()** method.

The **action** method of the applet class is overridden to intercept the action events for all of the buttons attached to the applet. One of the arguments passed to **action**, called `target`, contains a reference to the button clicked.

```
public boolean action(Event event, Object objtype)
{
.
.
.
   if (event.target == bLoadImage)
   {
      LoadImage();
      myContext.showStatus("Done");

   }

   return true;
}
```

The **LoadImage()** method (shown in the following code snippet) uses several properties and methods to display the image document file in the Image Edit and Image Thumbnail controls.

Specifically, it performs the following tasks:

- Sets the **Image** property of the Image Edit control to the value of the `sImage` variable. (You can assign the complete path and file name of a multipage TIFF image file on your PC to the `sImage` variable.)
- Invokes the **FitTo** method of the Image Edit control, passing to it a:
  - Literal value of `1`, which sets the initial fit-to setting of the Image Edit control to the Fit to Width option.
  - Variant containing the boolean value of `False`, which prevents the method from refreshing the Image Edit control.
- Sets the **ImagePalette** property of the Image Edit control to the Common palette (literal value of `1` assigned to `BackGround`) to prevent flicker.
- Sets the **DisplayScaleAlgorithm** property of the Image Edit control to the value of the `wiScaleOptimize` constant (literal 4), which displays black-and-white images in gray scale for greater clarity.
- Invokes the **Display** method of the Image Edit control, which displays the image specified in the **Image** property at the fit-to, image palette, and display-scale options specified.
- Sets the `PageCount` variable to the value of the **PageCount** property of the Image Edit control, which contains the total number of image pages in the image document file. (The applet will use the `PageCount` value later when navigating a multipage image document.)
- Sets the height and width of the individual thumbnail images by setting the **ThumbHeight** and **ThumbWidth** properties of the Image Thumbnail control.
- Sets the **Image** property of the Image Thumbnail control to the value of the `sImage` variable, so the thumbnail control displays the same image file as the Image Edit control.
- Sets the **ThumbCaptionStyle** property of the Image Thumbnail control to the value of the `SimpleWithAnno` constant (literal 2), which displays a left-justified page number caption beneath each thumbnail image along with an annotation indicator for those image pages that have annotations.
- Sets the **ScrollDirection** property of the Image Thumbnail control to the value of the `Vertical` constant (literal 1), which enables the end user to scroll the individual thumbnail images vertically.

Setting the **DisplayScale Algorithm** property to Optimize is the recommended way to make black-and-white images appear in gray scale. Color images continue to appear in color using this setting.

- Uses the **ThumbCaptionFont** property of the Image Thumbnail control to return a thumbnail caption font object. The applet uses the font object to change two of its properties. Specifically, it changes the thumbnail caption font size to 4 points, and the thumbnail caption font name to Courier New.
- Invokes the **DisplayThumbs** method of the Image Thumbnail control to display the thumbnail images, passing to it:
  - A ThumbNumber parameter value of 1 (from `vPage`), which sets thumbnail image number 1 as the first thumbnail image to which the Option parameter setting (described next) applies.
  - An Option parameter value of 1 (from `Pos`), which sets the position of the thumbnails to the middle row within the control.
- For page 1, sets the **ThumbSelected** property of the Image Thumbnail control to `True` which automatically selects — or gives focus to — the first thumbnail image within the control.

```
private void LoadImage()
{
  try
  {
  // retrieve image and save as temp file"

  myContext.showStatus("Loading Image " + sImage);

  theImage.putImage(sImage);

  // set the image scale
  Variant vOption = new Variant();
  vOption.putBoolean(false);
  short type = 1;
  theImage.FitTo(type, vOption);

  // set the image palette to use the background
  // to prevent flicker
  short BackGround = 1;
  theImage.SetImagePalette(BackGround);

  // display the image using scale to gray
  int ScaleToGray = DisplayScaleConstants.wiScaleOptimize;
  theImage.putDisplayScaleAlgorithm(ScaleToGray);

  // display the image
  theImage.Display();

  // get the number of pages
  PageCount = theImage.getPageCount();

  cRotate.enable();
  cLoadImage.disable();

(continued on the next page)
```

```
  if (PageCount > 1)
   cNextPage.enable();

  // set the thumbnail size
  int Height = 75;
  int Width = 62;
  theThumb.putThumbHeight(Height);
  theThumb.putThumbWidth(Width);

  // use the same image as displayed in the Image Control
  theThumb.putImage(sImage);

  // use the SimpleWithAnno caption to display page number
  // under thumbnail
  short Caption = CaptionStyleConstants.SimpleWithAnno;
  theThumb.putThumbCaptionStyle(Caption);


  // scroll the thumbnails vertically
  short ScrollVertical = ScrollConstants.Vertical;
  theThumb.putScrollDirection(ScrollVertical);

  // display the first thumbnail page
  Variant vPage = new Variant();
  Variant vPos = new Variant();

  int Page = 1;
  short Pos = 1;
  vPage.putInt(Page);

  // set the font and size of the caption initializing it from the
  //default thumbnail font
  olepro32.Font theFont;
  theFont = theThumb.getThumbCaptionFont();
  long FontSize = 4;
  theFont.putSize(FontSize);
  String FontName = "Courier New";
  theFont.putName(FontName);

  vPos.putShort(Pos);
  theThumb.DisplayThumbs(vPage, vPos);
  theThumb.putThumbSelected(1, true);
  }
  catch (Throwable e)
}
```

Rotating the Image

To rotate the image, click the **Rotate Page** button on the applet window.

The **action** method evaluates the `target` argument to determine which button you clicked. When you click the **Rotate Page** button, the method invokes the **RotateRight** method of the Image Edit control without passing a parameter. By default, the **RotateRight** method rotates the image 90 degrees to the right.

```
public boolean action(Event event, Object objtype)
{
// rotate page
  if (event.target == bRotate)
  {
    try
    {
      Variant vParam = new Variant();
      // set variant to no parameter passed
      vParam.noParam();
      theImage.RotateRight(vParam);
    }
    catch (Throwable e)
    {
    }

  }
.
.
.
}
```

Navigating the Image Document File

To navigate the image, click the **Next Page** or **Previous Page** button on the applet window. The **action** method (shown in the following code snippets) evaluates the `target` argument to determine which button you clicked.

When you click the **Next Page** button, the method assigns the current page number — provided by the value of the **Page** property of the Image Edit control — to the `CurrentPage` local variable.

As long as the value of `CurrentPage` is less than the maximum number of pages in the file (as provided by the value of the `PageCount` variable), the method sets the **Page** property of the Image Edit control to the value of `CurrentPage + 1`. Then it invokes the **Display** method of the Image Edit control to display the next page. (Note that the `CurrentPage` variable now represents the *previous* page.)

After enabling or disabling the **Next Page** and **Previous Page** buttons as you would expect, the method deals with the Image Thumbnail control.

As the user navigates the multipage image file, having the Image Thumbnail control synchronize automatically with the Image Edit control would be nice. Using the **ThumbSelected** property of the Image Thumbnail control, the method performs this task by selecting the thumbnail image that corresponds to the image page currently being displayed.

Synchronizing the Image Thumbnail control with the Image Edit control is actually a three step process:

1. The method sets the **ThumbSelected** property to the value of the `CurrentPage` variable with a Selected value of `False`. This action removes the selected highlight from the thumbnail image that corresponds to the *previous* page.
2. The method sets the **ThumbSelected** property to the value of `CurrentPage + 1` with a Selected value of `True`. This action applies the selected highlight to the thumbnail image that corresponds to the *currently displayed* page.
3. The method invokes the **DisplayThumbs** method of the Image Thumbnail control, passing to it:
   - A ThumbNumber parameter value of `CurrentPage + 1` (from `vPage`), which displays the thumbnail image that corresponds to the page currently being displayed in the Image Edit control.
   - An Option parameter value of 1 (from `vPos`), which sets the position of the individual thumbnail images to the middle row within the control.

```
public boolean action(Event event, Object objtype)
{
   // display next page
   if (event.target == bNextPage)
   {
     try
     {
       // get the current page
       int CurrentPage = theImage.getPage();
       // position to next page
       if (CurrentPage < PageCount)
       {
         // set the page to the next page
         theImage.putPage(CurrentPage + 1);
         // display the page
         theImage.Display();
       }
       if(CurrentPage +  1 == PageCount)
         cNextPage.disable();
       else
          cNextPage.enable();
       cPrevPage.enable();

       theThumb.putThumbSelected(CurrentPage , false);
       theThumb.putThumbSelected(CurrentPage + 1, true);

       Variant vPage = new Variant();
       Variant vPos = new Variant();
       vPage.putInt(CurrentPage+1);
       vPos.putShort((short)1);
       theThumb.DisplayThumbs(vPage, vPos);
       }
     catch (Throwable e)
     {
     }

   }
.
.
.
}
```

When you click the **Previous Page** button, the method assigns the current page number — provided by the value of the **Page** property of the Image Edit control — to the `CurrentPage` local variable.

As long as the value of `CurrentPage` is greater than the first page in the file, the method sets the **Page** property of the Image Edit control to the value of `CurrentPage - 1`. Then it invokes the **Display** method of the Image Edit control to display the previous page. (Note that the `CurrentPage` variable now represents the *next* page.)

After enabling or disabling the **Next Page** and **Previous Page** buttons as you would expect, the method deals with synchronizing the Image Thumbnail control:

1. The method sets the **ThumbSelected** property to the value of the `CurrentPage` variable with a Selected value of `False`. This action removes the selected highlight from the thumbnail image that corresponds to the *next* page.
2. The method sets the **ThumbSelected** property to the value of `CurrentPage - 1` with a Selected value of `True`. This action applies the selected highlight to the thumbnail image that corresponds to the *currently displayed* page.
3. The method invokes the **DisplayThumbs** method of the Image Thumbnail control, passing to it:
   - A ThumbNumber parameter value of `CurrentPage - 1` (from `vPage`), which displays the thumbnail image that corresponds to the page displayed in the Image Edit control.
   - An Option parameter value of 1 (from `vPos`), which sets the position of the individual thumbnail images to the middle row within the control.

```
public boolean action(Event event, Object objtype)
{
   if (event.target == bPrevPage)
   {
     try
     {
       // get the current page displayed
       int CurrentPage = theImage.getPage();
       if (CurrentPage > 1)
       {
         // set to previous page
         theImage.putPage(CurrentPage -  1);
         // display the page
         theImage.Display();
       }
       if(CurrentPage -  1== 1)
         cPrevPage.disable();
       else
         cPrevPage.enable();
       cNextPage.enable();

       theThumb.putThumbSelected(CurrentPage , false);
       theThumb.putThumbSelected(CurrentPage - 1, true);

       Variant vPage = new Variant();
       Variant vPos = new Variant();
       vPage.putInt(CurrentPage -1);
       vPos.putShort((short)1);
       theThumb.DisplayThumbs(vPage, vPos);
     }
     catch (Throwable e)
     {
     }
   }
.
.
.
}
```

7

Image Edit/Image Annotation Controls

This chapter describes the properties, methods, and events for the Image Edit and Image Annotation controls. This chapter also describes properties, methods, and events which are common to the Image OCR, Image Scan, Image Admin, and Image Thumbnail controls. See the overview in this chapter for more information.

In This Chapter

Overview

Image Edit Control

All properties, methods, and events in this chapter apply to the Image Edit control, with the exception of the following five properties, which apply only to the Image Annotation control:

- DestImageControl property
- PictureDisabled property
- PictureDown property
- PictureUp property
- Value property

Image Annotation Tool Button Control

The following properties described in this chapter apply to the Image Annotation Tool Button control:

- AnnotationBackColor
- AnnotationFillColor
- AnnotationFillStyle
- AnnotationFont
- AnnotationFontColor
- AnnotationImage
- AnnotationLineColor
- AnnotationLineStyle
- AnnotationLineWidth
- AnnotationOcrType
- AnnotationStampText
- AnnotationTextFile
- AnnotationType
- DestImageControl
- Enabled
- hWnd
- PictureDisabled
- PictureDown
- PictureUp
- ReadyState
- StatusCode
- Value

The following methods and events described in this chapter apply to the Annotation Tool Button control:

- AboutBox method
- Draw method
- ReadyStateChange event

Image Admin Control

The following properties and methods described in this chapter also apply to the Image Admin control:

- StatusCode property
- AboutBox method
- GetVersion method

Image OCR Control

The following properties, methods, and events described in this chapter also apply to the Image OCR control:

- AboutBox method
- GetVersion method
- ReadyStateChange event

Image Scan Control

The following properties and methods described in this chapter also apply to the Image Scan control:

- StatusCode property
- GetVersion method
- AboutBox method

Image Thumbnail Control

The following properties, methods, and events described in this chapter also apply to the Image Thumbnail control:

- Enabled property
- hWnd property
- StatusCode property
- AboutBox method
- GetVersion method
- ReadyStateChange event

Image Edit Control Overview

What the Image Edit Control Lets You Do

The Image Edit control lets you — the application developer — add image display, annotation, manipulation, and management functions to applications that support 32-bit ActiveX/OCX controls.

Note: The Image Edit control is an ActiveX control in:

- Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- Imaging for Windows 98

And an OCX control in:

- Imaging for Windows 95
- Imaging for Windows NT 4.0

What the Image Edit Control Lets Your Users Do

Depending on how you design and code your application, the Image Edit control lets your users display, annotate, and manipulate image document files and managed image documents.

Image Display

The Image Edit control allows end users to display image documents of various types. Specifically, the control supports the following file types:

- AWD
- BMP
- DCX
- GIF
- JPG
- PCX
- TIFF
- WIFF
- XIF

Note: AWD is not available with Imaging for Windows NT 4.0. GIF and WIFF are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0; and Imaging for Windows 98.

Image Annotation

The control includes several properties, methods, and events that enable you to add annotation functions to application programs. The annotation functions permit your end users to annotate displayed images.

You can set the annotation types (see page 529) you want to include in your application programmatically, or you can invoke the standard Annotation Tool Palette (see page 533) to include a complete set of annotation tools. Invoking the Annotation Tool Palette frees you from writing several lines of code.

Image Manipulation

The Image Edit control has properties, methods, and events that enable you to add image manipulation functions to your application programs. These functions permit end users to perform the following tasks on displayed images:

- Convert to text (when used in conjunction with the Image OCR control)
- Copy, cut, and paste to the Clipboard
- Crop
- Deskew
- Despeckle
- Flip
- Invert
- Rotate
- Scale
- Scroll
- Zoom

Image Document Retrieval and Storage

The Image Edit control also includes methods that enable you to add image document retrieval and storage functions to application programs. These functions permit end users to retrieve, save, print, and delete image and annotation data.

Linking Other Imaging Controls to the Image Edit Control

The Image Edit control is the main Imaging control. The Image Annotation Tool Button and Image Scan controls link to the Image Edit control to perform several functions.

Image Annotation Tool Button Control

The Image Annotation Tool Button control links to the Image Edit control to permit the annotation of displayed images. The Image Annotation Tool Button control sends messages to the Image Edit control that set annotation attributes, thereby changing all of the Image Edit annotation attributes to the values set within the Image Annotation Tool Button control. When the Draw method is invoked, the Image Edit control draws the annotation.

Image Scan Control

The Image Scan control links to the Image Edit control to display images while they are being scanned.

Establishing the Link

To link the Image Annotation Tool Button control to the Image Edit control:

1. Set the DestImageControl property of the Image Annotation Tool Button control to the name of the desired Image Edit control.
2. Set the ImageControl property of the Image Edit control to the same Image Edit control name.

To link the Image Scan control to the Image Edit control:

1. Set the DestImageControl property of the Image Scan control to the name of the desired Image Edit control.
2. Set the ImageControl property of the Image Edit control to the same Image Edit control name.

If your program has only one Image Annotation Tool Button or Image Scan control and one Image Edit control, VB 5.0 sets these properties automatically.

Image Annotation Tool Button Overview

What the Image Annotation Tool Button Control Lets You Do

The Image Annotation Tool Button (IATB) control lets you, the application developer, add image annotation functions to applications that support 32-bit ActiveX controls.

Specifically, you include one or more IATB controls in your application to create a custom annotation tool bar or palette. Each control is actually a button that invokes an annotation type when the end user clicks on it.

Your custom annotation tool bar or palette can contain buttons that provide:

- Discrete annotation types (see page 542)
- The same annotation type with different annotation styles (see page 541)
- A combination of both

Note: The IATB control is an ActiveX control in:

- Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- Imaging for Windows 98

And an OCX control in:

- Imaging for Windows 95
- Imaging for Windows NT 4.0

What the IATB Control Lets Your Users Do

Each IATB control lets your users draw text or graphical annotations on displayed image documents.

Prerequisites

You must include at least one Image Admin control and one Image Edit control in your application.

Further, you must link each IATB control to the Image Edit control that displays images to be annotated by your end users.

The link permits the IATB control to send messages to the Image Edit control. These messages change the annotation properties of the Image Edit control to the values set within the IATB control. It is the Image Edit control that actually draws the annotations.

To link an Image Annotation Tool Button control to an Image Edit control:

1. Set the DestImageControl property of the Image Annotation Tool Button control to the name of the desired Image Edit control.
2. Set the ImageControl property of the Image Edit control to the same Image Edit control name.

If your program has only one Image Annotation Tool Button control and one Image Edit control, VB 5.0 sets these properties automatically.

AnnotationBackColor Property

Description Returns or sets the background color of an Attach-a-Note annotation object.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationBackColor**[=*color*]

Applies To OLE_COLOR.

Remarks Use the RGB format (see page 540). The default color is yellow (255, 255, 0).

To ensure that the text in an Attach-a-Note annotation object will be visible against the background, specify a background color that is significantly different from the font color set using the **AnnotationFontColor** property.

See Also AnnotationFontColor property, AnnotationType property, GetSelectedAnnotationBackColor method, SetSelectedAnnotationBackColor method.

AnnotationBackColor Example — VB

This example adds an annotation group, sets the annotation type, sets the background color of the annotation, and then draws the mark. (Annotation groups are not actually created until a mark is drawn in the group.)

```
Private Sub cmdAddGroup_Click()
    ImgEdit1.AddAnnotationGroup "Accounting Dept"
    ImgEdit1.AnnotationType = wiTextAttachment  '10
    ImgEdit1.AnnotationBackColor = vbYellow
    ImgEdit1.Draw 10, 10, 150, 100
    'The user must manually enter the text on the attachment
End Sub
```

AnnotationBackColor Example — VC++

This example adds an annotation group, sets the annotation type, sets the background color of the annotation, and then draws the mark. (Annotation groups are not actually created until a mark is drawn in the group.)

```
void CFrmGroup::OnAddGroup()
{
    // Adds a group and draws a mark in the group. Annotation groups
    // are not actually created until a mark is drawn in the group.
    CString szCurGroup;
    szCurGroup.Format("Accounting Dept %i",rand());
    // Get the name of the group the user clicked on in the listbox
```

```
    m_GroupList.AddString(szCurGroup);
    if(pParentDlg)
    {
      pParentDlg->ImgEdit1.AddAnnotationGroup(szCurGroup);
      pParentDlg->ImgEdit1.SetAnnotationType(10); // wiTextAttachment // 10
      pParentDlg->ImgEdit1.SetAnnotationBackColor(0xFFFF); // vbYellow
      VARIANT vWidth; V_VT(&vWidth) = VT_I2; V_I2(&vWidth) = 150;
      VARIANT vHeight; V_VT(&vHeight) = VT_I2; V_I2(&vHeight) = 100;
      pParentDlg->ImgEdit1.Draw(10, 10, vWidth, vHeight);
    }
    // The user must manually enter the text on the attachment
}
```

AnnotationFillColor Property

Description Returns or sets the color used to fill a Filled Rectangle annotation object.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationFillColor**[=*color*]

Data Type OLE_COLOR.

Remarks Use the RGB format (see page 540). The default color is red (255, 0, 0).

See Also AnnotationFontColor property, AnnotationType property, GetSelectedAnnotationFillColor method, SetSelectedAnnotationFillColor method.

AnnotationFillColor Example — VB

This example uses the AnnotationFillColor property to set the color of a Filled Rectangle annotation to blue. The annotation is then drawn on the Image Edit control specified by the DestImageControl property.

```
Private Sub cmdDestImage_Click()
    'If there are multiple ImgEdit controls within an application, use the
    'DestImage control to define which image window to place annotation
    'marks on. ImageControl names will default to ImgEdit1, ImgEdit2, etc.
    'at the time the controls are drawn. The control name could be changed
    'at design time if desired. Draw a straight line on the Image Edit
    'control whose ImageControl name is ImgEdit2
    ImgAnnTool1.DestImageControl = "ImgEdit2"
    ImgAnnTool1.AnnotationType = wiStraightLine
    ImgAnnTool1.Draw 10, 10, 100, 100
```

```
    'Sets the AnnotationType to a filled rectangle and the fill color of the
    'rectangle to blue using Visual Basic color constants.
    ImgAnnTool1.AnnotationType = wiFilledRect '4
    ImgAnnTool1.AnnotationFillColor = vbBlue
    'Draw the blue rectangle on the Image Edit control whose ImageControl
    'name is ImgEdit1
    ImgAnnTool1.DestImageControl = "ImgEdit1"
    ImgAnnTool1.Draw 10, 10, 100, 100
End Sub
```

AnnotationFillStyle Property

Description Returns or sets the pattern used to fill image and Filled Rectangle annotation objects.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationFillStyle**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiTransparent	0 (default)	Transparent — The underlying image data is visible.
wiOpaque	1	Opaque — The underlying image data is obscured.

Remarks The AnnotationFillStyle property is used with the following annotation types:

- Filled Rectangle
- Image Embedded
- Image Reference

See Also AnnotationType property, GetSelectedAnnotationFillStyle method, SetSelectedAnnotationFillStyle method.

AnnotationFillStyle Example — VB

This example places a company logo (image) annotation at the top of a new page. The AnnotationFillStyle property is set to opaque so none of the underlying image data is visible through the logo.

```
Private Sub cmdDrawLogo_Click()
    'Create a new 8 1/2 X 11" black and white image at 100 dpi
    ImgEdit1.DisplayBlankImage 850, 1100, 100, 100, wiPageTypeBW
    ImgEdit1.AnnotationImage = "D:\image2\corplogo.bmp"
```

```
    'Logo would obscure any text under it if Opaque
    ImgEdit1.AnnotationFillStyle = wiOpaque   '1
    'If image is embedded rather than referenced, the source
    'image need not be present when the image is displayed
    ImgEdit1.AnnotationType = wiImageEmbedded   '5
    'Image mark only requires left and top parameters to draw
    'Start not quite halfway across page, 25 pixels from top (1/4")
    ImgEdit1.Draw 350, 25
End Sub
```

AnnotationFillStyle Example — VC++

This example places a company logo (image) annotation at the top of a new page. The AnnotationFillStyle property is set to opaque so none of the underlying image data is visible through the logo.

```
void CImgEdit1Dlg::OnImageStamp()
{
    // This places a company logo at the top center of a new page
    // Create a new 8 1/2 X 11" black and white image at 100 dpi
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vPageType) = 1; // wiPageTypeBW
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 100;
    ImgEdit1.DisplayBlankImage (850, 1100, vRes, vRes, vPageType);
    ImgEdit1.SetAnnotationImage("D:\\image2\\corplogo.bmp");
    // Logo would obscure any text under it if Opaque
    ImgEdit1.SetAnnotationFillStyle(1);  // wiOpaque   // 1
    // If image is embedded rather than referenced, the source
    // image need not be present when the image is displayed
    ImgEdit1.SetAnnotationType(5);  // wiImageEmbedded   // 5
    // Image mark only requires left and top parameters to draw
    // Start not quite halfway across page, 25 pixels from top (1/4")
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(350, 25, evt, evt);
}
```

AnnotationFont Property

Description Returns or sets a font object's properties. Applies to all text annotation types.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationFont**

Data Type IFontDisp.

Remarks Use the AnnotationFont property of an object to identify the specific font object whose properties you want to use or set. The following list shows the properties you can use or set, along with their data types, descriptions, and defaults:

Property	Data Type	Description	Default
Bold	Boolean	True — Selects bold attribute False — Deselects bold attribute	False
Italic	Boolean	True — Selects italic attribute False — Deselects italic attribute	False
Name	String	The desired font name	MS San Serif
Size	Single	The desired font size in points (0 to 2048)	12
Underline	Boolean	True — Selects underline attribute False — Deselects underline attribute	False
StrikeThrough	Boolean	True — Selects ~~strikethrough~~ attribute False — Deselects strikethrough attribute	False

The AnnotationFont property is used with the following annotation types:

- Attach-a-Note
- Hyperlink
- Text
- Text From File
- Text Stamp

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

See Also AnnotationType property, GetSelectedAnnotationFont method, SetSelectedAnnotationFont method.

AnnotationFont Example — VB

Example 1 places a customized rubber stamp at a predetermined location on the image. The AnnotationFont and AnnotationFontColor properties are set to determine the stamp's font and font color.

Example 2 shows how to change the properties of the font object identified by the AnnotationFont property.

Example 1

```
Private Sub cmdStamp_Click()
    ImgEdit1.AnnotationFont = "Courier"
    ImgEdit1.AnnotationFontColor = vbRed
    'This stamp uses embedded text macros for date and time
    ImgEdit1.AnnotationStampText = "Application received on %B %d, %Y
    ➥  at %H"
    ImgEdit1.AnnotationType = wiTextStamp  '8
    ImgEdit1.Draw 10, 10
End Sub
```

Example 2

```
Private Sub cmdAnnoFont_Click()
    'This example shows how to change the properties of the font object
    'identified by the AnnotationFont property.
    'This would define the font to be used for all text annotations.
    Dim AnnoFont As IFontDisp
    Set AnnoFont = ImgEdit1.AnnotationFont
    With AnnoFont
        .Bold = True
        .Italic = True
        .Name = "Arial"
        .Size = 16
        .Underline = True
        .Strikethrough = True
    End With
End Sub
```

AnnotationFont Example — VC++

This example places a customized rubber stamp at a predetermined location on the image. The AnnotationFont and AnnotationFontColor properties are set to determine the stamp's font and font color.

```
void CImgEdit1Dlg::OnRubberStamp()
{
    // Places a customized rubber stamp at a predetermined location on the
    // image. Font used is Courier and color is red.
    COleFont szFontType;
    szFontType = ImgEdit1.GetAnnotationFont();
    szFontType.SetName("Courier New");
    ImgEdit1.SetAnnotationFont(szFontType);
    ImgEdit1.SetAnnotationFontColor(0xFF);  // vbRed
    // This stamp uses embedded text macros for date and time
    ImgEdit1.SetAnnotationStampText("Application received on %B %d, %Y at
    ➥  %H");
    ImgEdit1.SetAnnotationType(8); // wiTextStamp  // 8
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

AnnotationFontColor Property

Description Returns or sets the font color of text annotation objects.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationFontColor**[=*color*]

Data Type OLE_COLOR.

Remarks Use the RGB format (see page 540). The default color is black (0, 0, 0).

The AnnotationFontColor property is used with the following annotation types:

- Attach-a-Note
- Hyperlink
- Text
- Text From File
- Text Stamp

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

See Also AnnotationBackColor property, AnnotationFillColor property, AnnotationType property, GetSelectedAnnotationFontColor method, SetSelectedAnnotationFontColor method.

AnnotationFontColor Example — VB

This example places a customized rubber stamp at a predetermined location on the image. The AnnotationFont and AnnotationFontColor properties are set to determine the stamp's font and font color.

```
Private Sub cmdStamp_Click()
    ImgEdit1.AnnotationFont = "Courier"
    ImgEdit1.AnnotationFontColor = vbRed
    'This stamp uses embedded text macros for date and time
    ImgEdit1.AnnotationStampText = "Application received on %B %d, %Y
    ➥  at %H"
    ImgEdit1.AnnotationType = wiTextStamp  '8
    ImgEdit1.Draw 10, 10
End Sub
```

AnnotationFontColor Example — VC++

This example places a customized rubber stamp at a predetermined location on the image. The AnnotationFont and AnnotationFontColor properties are set to determine the stamp's font and font color.

```
void CImgEdit1Dlg::OnRubberStamp()
{
    // Places a customized rubber stamp at a predetermined location on the
    // image. Font used is Courier and color is red.
    COleFont szFontType;
    szFontType = ImgEdit1.GetAnnotationFont();
    szFontType.SetName("Courier New");
    ImgEdit1.SetAnnotationFont(szFontType);
    ImgEdit1.SetAnnotationFontColor(0xFF);  // vbRed
    // This stamp uses embedded text macros for date and time
    ImgEdit1.SetAnnotationStampText("Application received on %B %d, %Y at
    ➥   %H");
    ImgEdit1.SetAnnotationType(8); // wiTextStamp  // 8
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

AnnotationGroupCount Property

Description Returns the number of annotation groups that are on an image page.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**AnnotationGroupCount**

Data Type Integer.

Remarks The AnnotationGroupCount is used as the upper bound value when indexing through all of the annotation groups.

When a new annotation group is created, the AnnotationGroupCount is not incremented until at least one annotation mark is added to the group.

Available at run-time as read-only.

The **Display** method must be invoked prior to reading this property.

See Also Display method, GetAnnotationGroup method, GetAnnotationMarkCount method, SetCurrentAnnotationGroup method.

AnnotationGroupCount Example — VB

This example uses the AnnotationGroupCount property to determine the number of groups on an image. Then it uses the GetAnnotationGroup method to get the group names.

```
Private Sub cmdListGroups_Click()
    Dim intNumGroups As Integer
    Dim strGroupName As String
    Dim intCount As Integer
    'find out how many groups are on the page
    intNumGroups = ImgEdit1.AnnotationGroupCount
    Load frmGroupList
    'Since index values of GetAnnotationGroup start with
    '0 have to loop one less than the number of groups
    For intCount = 0 To intNumGroups - 1
        strGroupName = ImgEdit1.GetAnnotationGroup(intCount)
        'write names of groups to a listbox
        frmGroupList.AnnoGroups.AddItem (strGroupName)
    Next intCount
    frmGroupList.Show
End Sub
```

AnnotationGroupCount Example — VC++

This example uses the AnnotationGroupCount property to determine the number of groups on an image. Then it uses the GetAnnotationGroup method to get the group names.

```
BOOL CFrmGroup::OnInitDialog()
{
    CDialog::OnInitDialog();
    // This routine uses AnnotationGroupCount to determine the number of
    // groups on an image and then gets the group names using the
    // GetAnnotationGroup method.
    int iNumGroups;
    CString szGroupName;
    int iCount;
    // find out how many groups are on the page
    iNumGroups = pParentDlg->ImgEdit1.GetAnnotationGroupCount();
    // Since index values of GetAnnotationGroup start with
    // 0 have to loop one less than the number of groups
    for (iCount = 0;iCount < iNumGroups ; iCount++)
    {
        szGroupName = pParentDlg->ImgEdit1.GetAnnotationGroup(iCount);
        // write names of groups to a listbox
        m_GroupList.AddString (szGroupName);
    }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

AnnotationImage Property

Description Returns or sets the fully-qualified file name of an image file. When the annotation is applied, the image file specified will be placed on the image currently displayed by the Image Edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationImage**[=*name*]

Data Type String.

Remarks The name must be a fully-qualified file name. File extensions are required as part of the file name.

The AnnotationImage property is used with the following annotation types:

- Image Embedded
- Image Reference

See Also AnnotationType property, Display method, GetSelectedAnnotationImage method.

AnnotationImage Example — VB

This example places a company logo annotation at the top center of a new page.

```
Private Sub cmdDrawLogo_Click()
    'Create a new 8 1/2 X 11" black and white image at 100 dpi
    ImgEdit1.DisplayBlankImage 850, 1100, 100, 100, wiPageTypeBW
        ImgEdit1.AnnotationImage = "D:\image2\corplogo.bmp"
    'Logo would obscure any text under it if Opaque
    ImgEdit1.AnnotationFillStyle = wiOpaque   '1
    'If image is embedded rather than referenced, the source
    'image need not be present when the image is displayed
    ImgEdit1.AnnotationType = wiImageEmbedded   '5
    'Image mark only requires left and top parameters to draw
    'Start not quite halfway across page, 25 pixels from top (1/4")
    ImgEdit1.Draw 350, 25
End Sub
```

AnnotationImage Example — VC++

This example places a company logo (image) annotation at the top, center of a new page.

```
void CImgEdit1Dlg::OnImageStamp()
{
    // This places a company logo at the top center of a new page
    // Create a new 8 1/2 X 11" black and white image at 100 dpi
    VARIANT vPageType; V_VT(&vPageType) = VT_I2;
    // set to PageType for saveas
    V_I2(&vPageType) = 1; / wiPageTypeBW
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 100;
    ImgEdit1.DisplayBlankImage (850, 1100, vRes, vRes, vPageType);
        ImgEdit1.SetAnnotationImage("D:\\image2\\corplogo.bmp");
    // Logo would obscure any text under it if Opaque
    ImgEdit1.SetAnnotationFillStyle(1);  // wiOpaque   // 1
    // If image is embedded rather than referenced, the source
    // image need not be present when the image is displayed
    ImgEdit1.SetAnnotationType(5);  // wiImageEmbedded   // 5
    // Image mark only requires left and top parameters to draw
    // Start not quite halfway across page, 25 pixels from top (1/4")
    VARIANT evt; V_VT(&evt) = VT_ERROR; / set to default
    ImgEdit1.Draw(350, 25, evt, evt);
}
```

AnnotationLineColor Property

Description Returns or sets the line color for line and Hollow Rectangle annotation objects.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationLineColor**[=*color*]

Data Type OLE_COLOR.

Remarks Use the RGB format (see page 540). The default color is red (255, 0, 0).

The AnnotationLineColor property is used with the following annotation types:

- Freehand Line
- Hollow Rectangle
- Straight Line

See Also AnnotationType property, GetSelectedAnnotationLineColor method, SetSelectedAnnotationLineColor method.

AnnotationLineColor Example — VB

This example draws a green line across the width of an image.

```
Private Sub cmdDrawLine_Click()
    Dim XWidth As OLE_XSIZE_PIXELS
    'Determine the width of the image -- this will
    'work even if the image is scaled up or down
    XWidth = ImgEdit1.ImageScaleWidth
    'line will be green, transparent, 10 pixels wide
    ImgEdit1.AnnotationLineColor = vbGreen
    ImgEdit1.AnnotationLineStyle = wiTransparent  '0
    ImgEdit1.AnnotationLineWidth = 10
    ImgEdit1.AnnotationType = wiStraightLine  '1
    'Draw the line
    'left position = 0 (start at left side)
    'top position = 100 pixels from top of page
    'width of line = width of image as it is displayed
    'height of line = 0 (line will be horizontal)
    ImgEdit1.Draw 0, 100, XWidth, 0
End Sub
```

AnnotationLineColor Example — VC++

This example draws a green line across the width of an image.

```
void CImgEdit1Dlg::OnDrawline()
{
    // This would draw a line across the width of an image
    long XWidth;
    // Determine the width of the image -- this will work even if the image
    // is scaled up or down
    XWidth = ImgEdit1.GetImageScaleWidth();
    // line will be green, transparent, 10 pixels wide
    ImgEdit1.SetAnnotationLineColor(0xFF00);    // vbGreen
    ImgEdit1.SetAnnotationLineStyle(0);  // 0
    ImgEdit1.SetAnnotationLineWidth(10); // 10
    ImgEdit1.SetAnnotationType(1); // wiStraightLine 1
    // Draw the line
    // left position = 0 (start at left side)
    // top position = 100 pixels from top of page
    // width of line = width of image as it is displayed
    // height of line = 0 (line will be horizontal)
    VARIANT vWidth; V_VT(&vWidth) = VT_I4; V_I4(&vWidth) = XWidth;
    VARIANT vHeight; V_VT(&vHeight) = VT_I4; V_I4(&vHeight) = 0;
    ImgEdit1.Draw(0, 100, vWidth, vHeight);
}
```

AnnotationLineStyle Property

Description Returns or sets the line style for line and Hollow Rectangle annotation objects.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
√ Imaging for Windows 95 and 98
√ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationLineStyle**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiTransparent	0 (default)	Transparent — The underlying image data is visible.
wiOpaque	1	Opaque — The underlying image data is obscured.

Remarks The AnnotationLineStyle property is used with the following annotation types:

- Freehand Line
- Hollow Rectangle
- Straight Line

See Also AnnotationLineWidth property, AnnotationType property, GetSelectedAnnotationLineStyle method, SetSelectedAnnotationLineStyle method.

AnnotationLineStyle Example — VB

This example draws a transparent line across the width of an image.

```
Private Sub cmdDrawLine_Click()
    Dim XWidth As OLE_XSIZE_PIXELS
    'Determine the width of the image -- this will work even if the image
    'is scaled up or down
    XWidth = ImgEdit1.ImageScaleWidth
    'line will be green, transparent, 10 pixels wide
    ImgEdit1.AnnotationLineColor = vbGreen
    ImgEdit1.AnnotationLineStyle = wiTransparent  '0
    ImgEdit1.AnnotationLineWidth = 10
    ImgEdit1.AnnotationType = wiStraightLine  '1
    'Draw the line: left position = 0 (start at left side)
    'top position = 100 pixels from top of page
    'width of line = width of image as it is displayed
    'height of line = 0 (line will be horizontal)
    ImgEdit1.Draw 0, 100, XWidth, 0
End Sub
```

AnnotationLineStyle Example — VC++

This example draws a transparent line across the width of an image.

```
void CImgEdit1Dlg::OnDrawline()
{
    // This would draw a line across the width of an image
    long XWidth;
    // Determine the width of the image -- this will work even if the image
    // is scaled up or down
    XWidth = ImgEdit1.GetImageScaleWidth();
    // line will be green, transparent, 10 pixels wide
    ImgEdit1.SetAnnotationLineColor(0xFF00);    // vbGreen
    ImgEdit1.SetAnnotationLineStyle(0);  // 0
    ImgEdit1.SetAnnotationLineWidth(10); // 10
    ImgEdit1.SetAnnotationType(1); // wiStraightLine 1
    // Draw the line
    // left position = 0 (start at left side)
    // top position = 100 pixels from top of page
    // width of line = width of image as it is displayed
    // height of line = 0 (line will be horizontal)
    VARIANT vWidth; V_VT(&vWidth) = VT_I4; V_I4(&vWidth) = XWidth;
    VARIANT vHeight; V_VT(&vHeight) = VT_I4; V_I4(&vHeight) = 0;
    ImgEdit1.Draw(0, 100, vWidth, vHeight);
}
```

AnnotationLineWidth Property

Description Returns or sets the line width for line and Hollow Rectangle annotation objects.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationLineWidth**[=*value*]

Data Type Integer.

Remarks The integer value specifies the line width in pixels. The valid range is from 1 to 999 pixels; the default value is 2 pixels.

The AnnotationLineWidth property is used with the following annotation types:

- Freehand Line
- Hollow Rectangle
- Straight Line

See Also AnnotationLineStyle property, AnnotationType property, GetSelectedAnnotationLineWidth method, SetSelectedAnnotationLineWidth method.

AnnotationLineWidth Example — VB

This example draws a 10-pixel-wide line across the width of an image.

```
Private Sub cmdDrawLine_Click()
    Dim XWidth As OLE_XSIZE_PIXELS
    'Determine the width of the image -- this will work even if the image
    'is scaled up or down
    XWidth = ImgEdit1.ImageScaleWidth
    'line will be green, transparent, 10 pixels wide
    ImgEdit1.AnnotationLineColor = vbGreen
    ImgEdit1.AnnotationLineStyle = wiTransparent  '0
    ImgEdit1.AnnotationLineWidth = 10
    ImgEdit1.AnnotationType = wiStraightLine  '1
    'Draw the line
    'left position = 0 (start at left side)
    'top position = 100 pixels from top of page
    'width of line = width of image as it is displayed
    'height of line = 0 (line will be horizontal)
    ImgEdit1.Draw 0, 100, XWidth, 0
End Sub
```

AnnotationLineWidth Example — VC++

This example draws a 10-pixel-wide line across the width of an image.

```
void CImgEdit1Dlg::OnDrawline()
{
    // This would draw a line across the width of an image
    long XWidth;
    // Determine the width of the image -- this will work even if the image
    // is scaled up or down
    XWidth = ImgEdit1.GetImageScaleWidth();
    // line will be green, transparent, 10 pixels wide
    ImgEdit1.SetAnnotationLineColor(0xFF00);    // vbGreen
    ImgEdit1.SetAnnotationLineStyle(0);  // 0
    ImgEdit1.SetAnnotationLineWidth(10); // 10
    ImgEdit1.SetAnnotationType(1); // wiStraightLine 1
    // Draw the line
    // left position = 0 (start at left side)
    // top position = 100 pixels from top of page
    // width of line = width of image as it is displayed
    // height of line = 0 (line will be horizontal)
    VARIANT vWidth; V_VT(&vWidth) = VT_I4; V_I4(&vWidth) = XWidth;
    VARIANT vHeight; V_VT(&vHeight) = VT_I4; V_I4(&vHeight) = 0;
    ImgEdit1.Draw(0, 100, vWidth, vHeight);
}
```

AnnotationOcrType Property

Description Returns or sets the type of OCR zone to be drawn on an image page.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationOcrType**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiOcrTypeText	0 (default)	OCR Text zone — Defines an area of an image containing text to be processed by the OCR engine. When the OCR engine encounters a text zone, it converts the underlying image to text. When drawn, an OCR Text zone is dark green in color.
wiOcrTypePicture	1	OCR Picture zone — Defines an area of an image containing one or more pictures that *should not* be processed by the OCR engine. When the OCR engine encounters a picture zone, it does not convert the underlying image to text, it retains the image as a graphic instead. When drawn, an OCR Picture zone is dark magenta in color.

Remarks Applies to the OCR Zones annotation type only.

While users can draw OCR Zone annotations on any type of image, OCR processing functions only with zones drawn on read/write TIFF images.

See Also AnnotationType property, GetSelectedAnnotationOcrType method, Image OCR control, OcrZoneVisibility property, SelectFirstOcrZone method, SelectNextOcrZone method, SetSelectedAnnotationOcrType method.

AnnotationOcrType Example — VB

This example lets the user draw OCR text regions on an image for zoned recognition. The AnnotationType property is set to the OCR Zone annotation type, and the AnnotationOcrType property is set so a Text OCR zone can be drawn.

```
Private Sub cmdDrawOcrZone_Click()
    ImgEdit1.AnnotationType = wiOcrRegion
    ImgEdit1.AnnotationOcrType = wiOcrTypeText
End Sub
```

AnnotationOcrType Example — VC++

This example lets the user draw OCR text regions on an image for zoned recognition. The AnnotationType property is set to the OCR Zone annotation type, and the AnnotationOcrType property is set so a Text OCR zone can be drawn.

```
void CImgEdit1Dlg::OnDrawOcrZone()
{
    // This would enable the user to draw OCR text regions on an image for
    // zone recognition.
    ImgEdit1.SetAnnotationType(13);    // wiOcrRegion
    ImgEdit1.SetAnnotationOcrType(0);  // wiOcrTypeText
}
```

AnnotationStampText Property

Description Returns or sets the format of the stamp text to be placed on an image by a Text Stamp annotation. The stamp text can consist of text, text macro(s), or a combination of both.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationStampText**[=*stamp*]

Data Type String.

Remarks Text is placed on the image when one of the following events occurs:

- The end user releases the left mouse button on a displayed image
- The end user releases the left mouse button on a displayed image
- The **Draw** method is invoked

The stamp text can include the following text macros:

Macro	Description
%b	Alphabetic abbreviation of the month; for example, Jan
%B	Alphabetic full name of the month; for example, January
%d	ASCII numeric day of the month, ranging from 1 to 31
%H	ASCII time in a 24-hour, hh:mm format, where: hh = hour mm = minute
%I	ASCII time in a 12-hour, hh:mm Xm format, where: hh = hour mm = minute Xm = AM or PM
%m	ASCII numeric designation of the month, ranging from 1 to 12 for January through December respectively
%x	Default Windows date format
%X	Default Windows time format
%y	ASCII numeric last two digits of the year; for example, 97 for 1997
%Y	ASCII numeric full year; for example, 1997

See Also AnnotationType property, Draw method.

Example

To create an annotation text stamp that displays the current time and date in a 12-hour format, enter:

```
Time: %I Date: %d-%B-%y
```

The resulting annotation text stamp is:

```
Time: 10:38 pm Date: 07-November-97
```

To create an annotation text stamp that displays the current time and date in a 24-hour format, enter:

```
Time: %H Date: %d-%B-%y
```

The resulting annotation text stamp is:

```
Time: 22:38 Date: 07-November-97
```

AnnotationStampText Example — VB

This example places a customized rubber stamp at a predetermined location on the image. TheAnnotationStampText property is set to the desired stamp content, which includes a combination of text and date macros.

```
Private Sub cmdStamp_Click()
    ImgEdit1.AnnotationFont = "Courier"
    ImgEdit1.AnnotationFontColor = vbRed
    'This stamp uses embedded text macros for date and time
    ImgEdit1.AnnotationStampText = "Application received on %B %d, %Y at %H"
    ImgEdit1.AnnotationType = wiTextStamp  '8
    ImgEdit1.Draw 10, 10
End Sub
```

AnnotationStampText Example — VC++

This example places a customized rubber stamp at a predetermined location on the image. TheAnnotationStampText property is set to the desired stamp content, which includes a combination of text and date macros.

```
void CImgEdit1Dlg::OnRubberStamp()
{
    // Places a customized rubber stamp at a predetermined location on the
    // image. Font used is Courier and color is red.
    COleFont szFontType;
    szFontType = ImgEdit1.GetAnnotationFont();
    szFontType.SetName("Courier New");
    ImgEdit1.SetAnnotationFont(szFontType);
    ImgEdit1.SetAnnotationFontColor(0xFF);  // vbRed
    // This stamp uses embedded text macros for date and time
    ImgEdit1.SetAnnotationStampText("Application received on %B %d, %Y at
    ➥  %H");
    ImgEdit1.SetAnnotationType(8); // wiTextStamp  // 8
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

AnnotationTextFile Property

Description Returns or sets the fully-qualified file name of a text file to embed into an image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationTextFile**[=*filename*]

Data Type String.

Remarks The name must be a fully-qualified file name. A file extension is required for all files.

The AnnotationTextFile property is used with the Text From File annotation type.

See Also AnnotationType property.

AnnotationTextFile Example — VB

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
Private Sub cmdSelectTool_Click()
    'If the annotation tool palette is utilized within an application,
    'SelectTool should be used rather than setting the
    'AnnotationType property for drawing marks programatically.
    ImgEdit1.SelectTool 9 'Text from file
    ImgEdit1.AnnotationTextFile = "C:\text\disclaim.txt"
    ImgEdit1.Draw 20, 20
End Sub
```

AnnotationTextFile Example — VC++

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
void CImgEdit1Dlg::OnSelectTool()
{
    // Uses the SelectTool method to select the Text from file annotation
    // type. Overlays text from a text file onto image. If the annotation
    // tool palette is utilized within an application, SelectTool
    // should be used rather thansetting the AnnotationType property for
    // drawing marks programatically.
    ImgEdit1.SelectTool(9); // Text from file
    ImgEdit1.SetAnnotationTextFile("C:\\text\\disclaim.txt");
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

AnnotationType Property

Description Returns or sets the type of annotation to be drawn.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Note: Hyperlink and OCR Zone annotation types are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

Applies To Image Edit and Image Annotation controls.

Usage *object*.**AnnotationType**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiNone	0 (default)	None
wiStraightLine	1	Straight Line
wiFreehandLine	2	Freehand Line
wiHollowRect	3	Hollow Rectangle
wiFilledRect	4	Filled Rectangle
wiImageEmbedded	5	Image Embedded
wiImageReference	6	Image Reference
wiText	7	Text
wiTextStamp	8	Text Stamp
wiTextFromFile	9	Text From File
wiTextAttachment	10	Attach-a-Note
wiAnnotation Selection	11	Select Annotations
wiHyperlink	12	Hyperlink
wiOcrRegion	13	OCR Zone

Prerequisite Properties

Some annotation types require that other properties be set to valid values before they can be used in an annotation. The following list shows the properties that must be set before the corresponding annotation type can be drawn:

Prerequisite Property	Annotation Type
AnnotationImage	Image Embedded
AnnotationImage	Image Reference
AnnotationStampText	Text Stamp
AnnotationTextFile	Text From File

The **Draw** method generates an error if it is invoked without the required property being set.

If an end user attempts to draw an annotation using the Image Annotation Tool Button control without the required property being set, an error event is sent to the target Image Edit control window. It is up to you to decide whether to present this error to the end user in your application.

Do not attempt to change the AnnotationType property when the standard Annotation Tool Palette is displayed. Use the **SelectTool** method instead.

Tab Characters

The Image Annotation Tool Button control and the Image Edit control do not support the use of Tab characters in any of the text-related annotation types.

Image Embedded and Image Reference Annotation Remarks

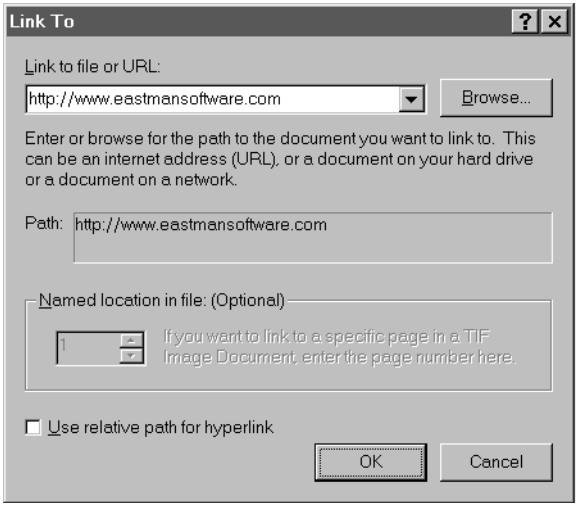
The Image Embedded annotation type (setting 5) stores a copy of the actual image data. The Image Reference annotation type (setting 6) stores the image data by reference (file name only).

Hyperlink Annotation Remarks

The Hyperlink annotation type (setting 12) lets end users enter hypertext jumps on an image. (The hypertext jumps are similar to those encountered on the Word Wide Web.)

When end users draw the text box for a Hyperlink annotation at run time, Imaging invokes the Link To dialog box (shown here), which enables them to specify the item they want the annotation to be linked to. Hyperlink annotations can be linked to the following items:

- An image page in a TIFF image document stored on a local or network drive
- Universal Resource Locator (URL) on the World Wide Web



While users can draw a Hyperlink annotation on any type of image, a Hyperlink annotation can be saved to disk only when the image upon which it is drawn is saved as a TIFF file. If end users burn-in a Hyperlink annotation, it no longer functions as a hypertext jump.

Keep in mind that *any* annotation type can be a Hyperlink annotation. You can use the **ShowAttribsDialog** method of the Image Edit control to invoke the Link To dialog box (shown below) at run time. The Link To dialog box lets end users specify the item they want a selected annotation to be linked to.

OCR Zone Annotation Remarks

The OCR Zones annotation type (setting 13) lets end users draw OCR zones on an image. OCR zones control the process of recognizing image documents (otherwise known as performing OCR). Depending on the setting of the **AnnotationOcrType** property, OCR zones can be text zones or picture zones.

OCR Text zones — Define areas of an image containing text to be processed by the OCR engine. When the OCR engine encounters a text zone, it converts the underlying image to text. When drawn, OCR Text zones are dark green in color.

OCR Picture zones — Define areas of an image containing one or more pictures that *should not* be processed by the OCR engine. When the OCR engine encounters a picture zone, it does not convert the underlying image to text, it retains the image as a graphic instead. When drawn, OCR Picture zones are dark magenta in color.

While users can draw OCR Zone annotations on any type of image, OCR processing functions only with zones drawn on read/write TIFF images. OCR Zone annotations cannot be burned into TIFF images.

See Also AnnotationBackColor property, AnnotationFillColor property, AnnotationFillStyle property, AnnotationFont property, AnnotationFontColor property, AnnotationImage property, AnnotationLineColor property, AnnotationLineStyle property, AnnotationLineWidth property, AnnotationStampText property, AnnotationTextFile property, Value property, Draw method, AnnotationOcrType property, MousePointer property, OcrZoneVisibility property, GetAnnotationMarkCount method, GetSelectedAnnotationBackColor method, GetSelectedAnnotationFillColor method, GetSelectedAnnotationFillStyle method, GetSelectedAnnotationFont method, GetSelectedAnnotationFontColor method, GetSelectedAnnotationImage method, GetSelectedAnnotationLineColor method, GetSelectedAnnotationLineStyle method, GetSelectedAnnotationLineWidth method, GetSelectedAnnotationOcrType method, SetSelectedAnnotationBackColor method, SetSelectedAnnotationFillColor method, SetSelectedAnnotationFillStyle method, SetSelectedAnnotationFont method, SetSelectedAnnotationFontColor method, SetSelectedAnnotationLineColor method, SetSelectedAnnotationLineStyle method, SetSelectedAnnotationLineWidth method, SetSelectedAnnotationOcrType method, ShowAttribsDialog method, HyperlinkGoToPage event, HyperlinkGoToDoc event, ProcessingMode property (ImageOcr), EditSelectedAnnotationText method.

AnnotationType Example — VB

This example lets the user draw OCR text regions on an image for zoned recognition. The AnnotationType property is set to the OCR Zone annotation type, and the AnnotationOcrType property is set so a Text OCR zone can be drawn.

```
Private Sub cmdDrawOcrZone_Click()
    ImgEdit1.AnnotationType = wiOcrRegion
    ImgEdit1.AnnotationOcrType = wiOcrTypeText
End Sub
```

AnnotationType Example — VC++

This example lets the user draw OCR text regions on an image for zoned recognition. The AnnotationType property is set to the OCR Zone annotation type, and the AnnotationOcrType property is set so a Text OCR zone can be drawn.

```
void CImgEdit1Dlg::OnDrawOcrZone()
{
    // This would enable the user to draw OCR text regions on an image for
    // zone recognition.
    ImgEdit1.SetAnnotationType(13);    // wiOcrRegion
    ImgEdit1.SetAnnotationOcrType(0);  // wiOcrTypeText
}
```

AutoRefresh Property

Description Returns or sets whether a displayed image is refreshed immediately after changes are made to properties that can affect its visual appearance.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**AutoRefresh**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Refreshes the image immediately when changes are made to display-related properties.
False (default)	The image is not refreshed immediately.

Remarks Set this property to True when you want the image to be refreshed every time a property is changed that affects its visual display.

Set this property to False when you do not want the image to be refreshed every time a property is changed that affects its visual display. The image can be refreshed at any time by invoking the Refresh method, which updates the image using the current settings.

When the AutoRefresh property is set to True, changing the following properties causes the displayed image to be refreshed automatically:

- DisplayScaleAlgorithm
- ImagePalette
- ImageResolutionX
- ImageResolutionY
- ScrollBars
- ScrollPositionX
- ScrollPositionY
- Zoom

See Also DisplayScaleAlgorithm property, ImagePalette property, ImageResolutionX property, ImageResolutionY property, Refresh method, ScrollBars property, ScrollPositionX property, ScrollPositionY property, Zoom property (ImageEdit).

AutoRefresh Example — VB

This example shows several ways to orient an image, and redisplay it automatically.

```
Private Sub cmdRotate_Click()
   'This method displays a dialog box which allows user to specify
   'rotation by degrees. Repaints the screen automatically.
   'Rotate all writes the file to disk.
   ImgEdit1.Rotate True 'rotate all pages
   ImgEdit1.Rotate False 'rotate current page
   'Autorefresh must be set to true or must call Refresh after these
   'methods.
   ImgEdit1.AutoRefresh = True
   ImgEdit1.RotateLeft 'defaults to 90 degrees
   ImgEdit1.RotateRight 45 'optionally can specify degrees
   ImgEdit1.Flip '180 degrees
   'You must save the image to maintain orientation.
End Sub
```

AutoRefresh Example — VC++

This example shows several ways to orient an image, and redisplay it automatically.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically.
    // Rotate all writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
```

```
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

BorderStyle Property

Description Returns or sets whether the control is displayed with a border.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**BorderStyle**[=*value*]

Data Type Integer (enumerated).

Setting	Description
0	None
1 (default)	Fixed Single

Remarks Single-line, fixed-size is the only available border style.

Available at design-time only.

CompressionInfo Property

Description Returns compression option information for the image specified in the **Image** property, and the page specified in the **Page** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**CompressionInfo**

Data Type Long.

Property settings are expressed using the following bit-wise values:

Setting	Description	Notes
0	No compression options set.	Applicable only to uncompressed image files
1	EOL — Include/expect End Of Line. Each line is terminated with a standard 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	Expand 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	Used only for JPEG compression
128	Low Resolution/Medium Quality	Used only for JPEG compression
256	Low Resolution/Low Quality	Used only for JPEG compression
512	Medium Resolution/High Quality	Used only for JPEG compression
1024	Medium Resolution/Medium Quality	Used only for JPEG compression
2048	Medium Resolution/Low Quality	Used only for JPEG compression

Setting	Description	Notes
4096	High Resolution/High Quality	Used only for JPEG compression
8192	High Resolution/Medium Quality	Used only for JPEG compression
16384	High Resolution/Low Quality	Used only for JPEG compression

Remarks Image files with compression types other than JPEG and LZW can have CompressionInfo values between 1 and 63, in any combination. Image files with the LZW compression type can have a CompressionInfo value of 8. Image files with the JPEG compression type can have only one specific value between 64 and 16384 (as indicated in the preceding list).

The CompressionInfo value is independent of an image being displayed. If the image specified in the Image and Page properties is being displayed, the value indicates the compression of the displayed image. If the image specified in the Image and Page properties is not being displayed, the value indicates the compression of the image specified in the Image property.

Available at run-time as read-only.

See Also CompressionType property, Image property, Page property, SaveAs property, SavePage method.

CompressionInfo Example — VB

This example interprets the bitwise value returned by the CompressionInfo property. It is limited to black-and-white compression types only.

```
Private Sub cmdCompInfo_Click()
    'This routine interprets the bitwise value returned in the
    'CompressionInfo property.
    'This example is limited to black and white compression types only.
    Dim lngCompInfo As Long
    lngCompInfo = ImgEdit1.CompressionInfo
    'Load form which has check boxes for various compression
    'options and check the options which are applicable to the file.
    Load frmCompInfo
    If lngCompInfo And 1 Then     'file has EOLs
      frmCompInfo.chkEOL.Value = 1
    Else
      frmCompInfo.chkEOL.Value = 0
    End If
    If lngCompInfo And 2 Then   'file has packed lines
      frmCompInfo.chkPackedLines.Value = 1
    Else
      frmCompInfo.chkPackedLines.Value = 0
    End If
    If lngCompInfo And 4 Then   'file has prefix EOLs
```

```
      frmCompInfo.chkPreEOL.Value = 1
    Else
      frmCompInfo.chkPreEOL.Value = 0
    End If
    If lngCompInfo And 8 Then   'Compressed bit order, Left-To-Right
      frmCompInfo.chkCompRBO.Value = 1
    Else
      frmCompInfo.chkCompRBO.Value = 0
    End If
    If lngCompInfo And 16 Then  'Expanded bit order, Left-To-Right
      frmCompInfo.chkExpRBO.Value = 1
    Else
      frmCompInfo.chkExpRBO.Value = 0
    End If
    If lngCompInfo And 32 Then
      frmCompInfo.chkNegate.Value = 1  'Black and white are inverted on
                                       'expansion
    Else
      frmCompInfo.chkNegate.Value = 0
    End If
    frmCompInfo.Show
End Sub
```

CompressionInfo Example — VC++

This example interprets the bitwise value returned by the CompressionInfo property. It is limited to black-and-white compression types only.

```
void CImgEdit2Dlg::OnCompressioninfo()
{
    CCompresionInfo CompInfo;
    long lCompInfo;
    lCompInfo = ImgEdit1.GetCompressionInfo();
    // file has EOLs
    if(lCompInfo & 1)
      CompInfo.m_EOL = TRUE;
    else
      CompInfo.m_EOL = FALSE;
    // file has packed lines
    if(lCompInfo & 2)
      CompInfo.m_PackedLines = TRUE;
    else
      CompInfo.m_PackedLines = FALSE;
    // file has prefix EOLs
    if(lCompInfo & 4)
      CompInfo.m_PrefixEOL = TRUE;
    else
      CompInfo.m_PrefixEOL = FALSE;
    // Compressed bit order, Left-To-Right
    if(lCompInfo & 8)
      CompInfo.m_RBO = TRUE;
    else
```

```
      CompInfo.m_RBO = FALSE;
    // Expanded bit order, Left-To-Right
    if(lCompInfo & 16)
      CompInfo.m_LTR = TRUE;
    else
      CompInfo.m_LTR = FALSE;
    // Black and white are inverted on expansion
    if(lCompInfo & 32)
      CompInfo.m_Negate = TRUE;
    else
      CompInfo.m_Negate = FALSE;
      CompInfo.DoModal();
}
```

CompressionType Property

Returns the compression type of the image specified in the **Image** property, and the page specified in the **Page** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**CompressionType**

Data Type Integer (enumerated).

Property settings are expressed using the following values:

Setting	Description
0	Unknown
1	No compression
2	Group3(1D)
3	Group3(Modified Huffman)
4	PackBits
5	Group4(2D)
6	JPEG
7	Reserved
8	Group3(2D)
9	LZW

Remarks The CompressionType value is independent of an image being displayed. If the image specified in the Image and Page properties is being displayed, the value indicates the compression type of the displayed image. If the image specified in the Image and Page properties is not being displayed, the value indicates the compression type of the image specified in the Image property.

Available at run-time as read-only.

See Also CompressionInfo property, Image property, Page property, SaveAs method, SavePage method.

CompressionType Example — VB

This example uses the CompressionType property to determine the compression type of the image specified in the Image property.

```
Private Sub cmdCompType_Click()
    'Reads the CompressionType property to determine the
    'compression type of the image specified in the Image property.
    Dim strImgCompType As String
    'Read the CompressionType and translate the value to a corresponding
    'string.
    Select Case ImgEdit1.CompressionType
      Case 0
        strImgCompType = "Unknown"
      Case 1
        strImgCompType = "No Compression"
      Case 2
        strImgCompType = "Group3(1D)"
      Case 3
        strImgCompType = "Group3(Modified Huffman)"
      Case 4
        strImgCompType = "PackBits"
      Case 5
        strImgCompType = "Group4(2D)"
      Case 6
        strImgCompType = "JPEG"
      Case 7
        strImgCompType = "RBA"
      Case 8
        strImgCompType = "Group3(2D)"
      Case 9
        strImgCompType = "LZW"
    End Select
    'Display the compression type on a form for informational purposes,
    'or to allow modifications.
    frmInfo.lblImgInfo(0).Caption = strImgCompType
End Sub
```

ContinuePrinting Property

Description Sets whether to cancel or continue printing.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ContinuePrinting**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Continue printing.
False	Cancel printing.

Remarks Before you can use the ContinuePrinting property, the **UseCheckContinuePrinting** property must be set to True.

When you use the ContinuePrinting property, you must set it within the **CheckContinuePrinting** event. This action is required because the ContinuePrinting property is initialized to True upon entry into the event.

The ContinuePrinting property indicates whether to cancel or continue printing when the CheckContinuePrinting event fires. The CheckContinuePrinting event fires whenever a print operation requests whether it should cancel or continue printing (typically, whenever a page is printed). The ContinuePrinting property provides the event with the desired response.

Available at run-time as write-only.

See Also CheckContinuePrinting event, PrintImage method, UseCheckContinuePrinting property.

ContinueWithoutUndo Property

Description Sets whether to cancel an undo operation, or continue it without the undo information.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ContinueWithoutUndo**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Continue the undo operation without the undo information.
False	Cancel the undo operation.

Remarks When you use the ContinueWithoutUndo property, you must set it within the **ErrorSavingUndoInformation** event. This action is required because the ContinueWithoutUndo property is initialized to True upon entry into the event.

The ContinueWithoutUndo property indicates whether an undo operation should be canceled or continued without the undo information. The ErrorSavingUndoInformation event fires whenever an operation cannot save the information required to undo it. The ContinueWithoutUndo property provides the desired processing alternative.

Available at run-time as write-only.

DestImageControl Property

Description Returns or sets the link to the Image Edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Annotation control.

Usage *object*.**DestImageControl**[=*controlname*]

Data Type String.

Remarks Set the DestImageControl property to the name of the desired Image Edit control. The **ImageControl** property of the Image Edit control must also be set to the same name. A unique default name is provided when you draw the control on a form.

The Image Annotation Tool Button control links to the Image Edit control to permit the annotation of displayed images. The Image Annotation Tool Button control sends messages to the Image Edit control that set annotation attributes, thereby changing all of the Image Edit annotation attributes to the values set within the Image Annotation Tool Button control. When the **Draw** method is invoked, the Image Edit control draws the annotation.

See Also Draw method, ImageControl property (ImageEdit).

DestImageControl Example — VB

In this example, the DestImageControl property determines which Image Edit control will draw Image Annotation Tool Button annotations.

```
Private Sub cmdDestImage_Click()
    'If there are multiple ImgEdit controls within an application, use the
    'DestImage control to define which image window to place annotation
    'marks on. ImageControl names will default to ImgEdit1, ImgEdit2, etc.
    'at the time the controls are drawn. The control name could be
    'changed at design time if desired.
    'Draw a straight line on the Image Edit control whose
    'ImageControl name is ImgEdit2
    ImgAnnTool1.DestImageControl = "ImgEdit2"
    ImgAnnTool1.AnnotationType = wiStraightLine
    ImgAnnTool1.Draw 10, 10, 100, 100
    'Sets the AnnotationType to a filled rectangle and the fill color
    'of the rectangle to blue using Visual Basic color constants.
    ImgAnnTool1.AnnotationType = wiFilledRect '4
    ImgAnnTool1.AnnotationFillColor = vbBlue
    'Draw the blue rectangle on the Image Edit control whose
    'ImageControl name is ImgEdit1
    ImgAnnTool1.DestImageControl = "ImgEdit1"
    ImgAnnTool1.Draw 10, 10, 100, 100
End Sub
```

DisplayICMEnabled Property

Description Sets whether Image Color Matching (ICM) support is enabled. When enabled, Imaging uses color management profiles when displaying or printing an image so the output devices display the colors of the original image as accurately as possible.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DisplayICMEnabled**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	ICM support is enabled.
False (default)	ICM support is not enabled.

Remarks If Image Color Matching is enabled, but the input device, image file, or output device does not support ICM, Imaging performs no color matching.

The image file types that support ICM are BMP Versions 4 and 5, and TIFF Version 7.

DisplayScaleAlgorithm Property

Description Returns or sets the scaling algorithm used when displaying images. It is particularly useful for implementing scale-to-gray functionality.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Integer (enumerated).

Property settings are expressed using the following values:

Constant	Setting	Description	Notes
wiScaleNormal	0 (default)	Normal	Normal decimation. All page types remain unchanged.
WiScaleGray4	1	Gray4	4-bit gray-scale, 16 shades of gray. All page types are scaled to gray.
WiScaleGray8	2	Gray8	8-bit gray-scale, 256 shades of gray. All page types are scaled to gray.
WiScaleStamp	3	Stamp	Used for thumbnail representation of images
wiScaleOptimize	4	Optimize	Changes the display algorithm based on the page type of the displayed image: **Black&white images** — Are scaled to gray. **Palettized 4- and 8-bit, RGB, and BGR images** — Remain color images.

Remarks The DisplayScaleAlgorithm value can be specified before or after an image is displayed. If the value is specified after an image is displayed, the displayed image may change visually. Note that when scaling from normal decimation to another setting, the displayed image remains unchanged.

When the **AutoRefresh** property is set to True, changing the DisplayScaleAlgorithm property causes the displayed image to be refreshed automatically. Some settings may not be apparent until the image is scaled to a Zoom value under 100 percent.

See Also AutoRefresh property, PageType property.

DisplayScaleAlgorithm Example — VB

This example shows some property settings you might want to make prior to displaying an image.

```
Private Sub ImgEdit1_Load(ByVal Zoom As Double)
    'Here are some examples of default settings you might
    'want to specify prior to displaying an image.
    'Repaint any changes immediately (ex. zoom or resolution changes, etc.)
    ImgEdit1.AutoRefresh = True
    'ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.ScrollBars = True
    'Scale black and white images to grayscale. Keep color image
    'displayed as color images.
    ImgEdit1.DisplayScaleAlgorithm = wiScaleOptimize
    'Allow user to scroll image using keyboard shortcuts
    ImgEdit1.ScrollShortcutsEnabled = True
    'Display all OCR zones
    ImgEdit1.OcrZoneVisibility = wiOcrShowAllZones  '3
End Sub
```

DisplayScaleAlgorithm Example — VC++

This example shows some property settings you might want to make prior to displaying an image.

```
void CImgEdit1Dlg::OnLoadEditctrl1(double Zoom)
{
    // Here are some examples of default settings you might want to specify
    // prior to displaying an image. Repaint any changes immediately
    // (ex. zoom or resolution changes, etc.)
    ImgEdit1.SetAutoRefresh(TRUE);
    // ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.SetScrollBars(TRUE);
    // Scale black and white images to grayscale. Keep color image
    // displayed as color images.
    ImgEdit1.SetDisplayScaleAlgorithm(4);   // wiScaleOptimize
    // Allow user to scroll image using keyboard shortcuts
    ImgEdit1.SetScrollShortcutsEnabled(TRUE);
    // Display all OCR zones
    ImgEdit1.SetOcrZoneVisibility(3);  // wiOcrShowAllZones  // 3
}
```

Enabled Property

Description Enables or disables the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit, Image Annotation, and Image Thumbnail controls.

Usage *object*.**Enabled**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Control is enabled (can respond to events).
False	Control is disabled (cannot respond to events).

Note: The properties of the Image Annotation Tool Button control cannot be updated when the control is disabled.

FileType Property

Description Returns the file type of the image file specified in the **Image** property, and the page specified in the **Page** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

> **Note:** AWD is not available with Imaging for Windows NT 4.0. GIF and WIFF are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0; and Imaging for Windows 98.

Applies To Image Edit control.

Usage *object*.**FileType**

Data Type Integer (enumerated).

Property settings are expressed using the following values:

Constant	Setting	Description
wiFileTypeTIFF	1	TIFF
wiFileTypeAWD	2	AWD
wiFileTypeBMP	3	BMP
wiFileTypePCX	4	PCX
wiFileTypeDCX	5	DCX
wiFileTypeJPG	6	JPG
wiFileTypeXIF	7	XIF
wiFileTypeGIF	8	GIF
wiFileTypeWIFF	9	WIFF

Remarks The FileType value is independent of an image being displayed. If the image specified in the Image and Page properties is being displayed, the value indicates the file type of the displayed image. If the image specified in the Image and Page properties is not being displayed, the value indicates the file type of the image specified in the Image property.

A file with a .JFX extension is also a TIFF file type.

Available at run-time as read-only.

See Also Image property, Page property, SaveAs method, SavePage method.

FileType Example — VB

This example uses the FileType property to determine the file type of the image specified in the Image property.

```
Private Sub cmdFileType_Click()
    'Reads the FileType property to determine the FileType of
    'the image specified in the Image property
    Dim strImgFileType As String
    'Read the Filetype and translate the value to a corresponding string.
    Select Case ImgEdit1.FileType
      Case 0
        strImgFileType = "Unknown"
      Case 1
        strImgFileType = "TIF"
      Case 2
        strImgFileType = "AWD"
      Case 3
        strImgFileType = "BMP"
      Case 4
        strImgFileType = "PCX"
      Case 5
        strImgFileType = "DCX"
      Case 6
        strImgFileType = "JPG"
      Case 7
        strImgFileType = "XIF"
      Case 8
        strImgFileType = "GIF"
      Case 9
        strImgFileType = "WIF"
    End Select
    'Display the file type on a form for informational purposes.
    frmInfo.lblImgInfo(0).Caption = strImgFileType
End Sub
```

hWnd Property

Description Indicates the window handle to the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit, Image Annotation, and Image Thumbnail controls.

Usage *object*.**hWnd**

Data Type Long.

Remarks Available at run-time as read-only.

Image Property

Description Returns the fully-qualified name (see page 535) of an image document file or managed image document being displayed.

Sets the fully-qualified name of an image document file or managed image document to be displayed or otherwise operated on by the Image Edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Image**[=*filename*]

Data Type String

Remarks If a URL is specified, the image is initially downloaded as a temporary file. The temporary file is deleted when a new image name is specified, or the Image Edit control goes out of scope.

See Also Display method, ReadyState property, ReadyStateChange event.

Image Example — VB

This example shows how to set the Image property using a variety of path and file name formats.

```
Private Sub cmdSetImage_Click()
    'Windows filename
    ImgEdit1.Image = "c:\insurance\claims\claim234.tif"
    'Showing syntax for 1.x Server document
    'Note: prefix Image:// denotes service name for 1.x server
    ImgEdit1.Image = "Image://srvrname\volname:\CABNAME\DRAWERNAME\
    ➥  FOLDERNAME\YOUR DOC"
    'Showing syntax for 1.x Server filename with a volume name mapped.
    ImgEdit1.Image = "Image://srvrname/volname:/dirname/temp.tif"
    'Showing syntax for 3.x Server document
    'Note: prefix Imagex:// denotes service name for 3.x server
    ImgEdit1.Image = "Imagex://docid1234"
    'URL syntax
    ImgEdit1.Image = "http://www.eastmansoftware.com/sbu/sampimg.tif"
    'UNC filename
    ImgEdit1.Image = "\\srvrname\shared3\images\bw\thisisa.tif"
End Sub
```

Image Example — VC++

This example shows how to set the Image property using a variety of path and file name formats.

```
void CImgEdit1Dlg::OnImage()
{
    // Examples of syntax for the Image property
    // The single slashes in the name can be forward or back.
    // Windows filename
    ImgEdit1.SetImage("c:\\insurance\\claims\\claim234.tif");
    // Showing syntax for 1.x Server document
    // Note: prefix Image:// denotes service name for 1.x server
    ImgEdit1.SetImage("Image://srvrname\\volname:\\CABNAME\\DRAWERNAME\\
    ➥    FOLDERNAME\\YOUR DOC");
    // Showing syntax for 1.x Server filename with a volume name mapped.
    ImgEdit1.SetImage("Image://srvrname/volname:/dirname/temp.tif");
    // Showing syntax for 3.x Server document
    // Note: prefix Imagex:// denotes service name for 3.x server
    ImgEdit1.SetImage("Imagex://docid1234");
    // URL syntax
    ImgEdit1.SetImage("http://www.eastmansoftware.com/sbu/sampimg.tif");
    // UNC filename
    ImgEdit1.SetImage("\\srvrname\shared3\images\bw\thisisa.tif");
}
```

ImageControl Property

Description Returns or sets the unique identifier to be specified for linking with another Imaging control. The Image Annotation Tool Button and Image Scan controls link to the Image Edit control in order to perform several functions.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageControl**[=*controlname*]

Data Type String.

Remarks A unique default name is provided when you draw the control on a form.

See Also DestImageControl property, Image Annotation Tool Button control, Image Scan control.

ImageDisplayed Property

Description Indicates whether an image is currently being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageDisplayed**

Data Type Boolean.

Setting	Description
True	An image is being displayed.
False	An image is not being displayed.

Remarks Available at run-time as read-only.

See Also Display method.

ImageHeight Property

Description Returns the height in full-size coordinates of the image being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageHeight**

Data Type Long.

Remarks If an image is not being displayed, the ImageHeight property returns the full-size height of the image specified in the **Image** property.

If an image is not being displayed and the Image property does not contain the name of an image, the ImageHeight property generates an error.

Available at run-time as read-only.

See Also Image property, ImageScaleHeight property, ImageWidth property.

ImageHeight Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageHeight Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR;  // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
```

```
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

ImageModified Property

Description Indicates whether an image has been modified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageModified**

Data Type Boolean.

Setting	Description
True	The image has been modified.
False	The image has not been modified.

Remarks Because the value of this property indicates whether an image file has been modified, you can use it to determine whether your program should prompt users to save the image file.

Available at run-time as read-only.

Changes to the following properties set the ImageModified property to True:

- ImageResolutionX
- ImageResolutionY

Invoking the following methods sets the ImageModified property to True:

- AddAnnotationGroup
- AutoCrop
- AutoDeskew
- BurnInAnnotations
- ClipboardCut
- ClipboardPaste
- ConvertPageType
- Redo
- RemoveAllOCRMarks
- Rotate
- RotateAll
- RotateLeft
- RotateRight
- SetSelectedAnnotationBackColor

- Crop
- DeleteAnnotationGroup
- DeleteImageData
- DeleteSelectedAnnotations
- Despeckle
- Draw method
- EditSelectedAnnotationText
- Flip
- HideAnnotationGroup
- Invert
- ManualDeSkew
- SetSelectedAnnotationFillColor
- SetSelectedAnnotationFillStyle
- SetSelectedAnnotationFont
- SetSelectedAnnotationFontColor
- SetSelectedAnnotationLineColor
- SetSelectedAnnotationLineStyle
- SetSelectedAnnotationLineWidth
- SetSelectedAnnotationOcrType
- ShowAnnotationGroup
- Undo

Drawing an annotation using the standard Annotation Tool Palette (see page 533) also sets the ImageModified property to True.

Invoking the following methods sets the ImageModified property to False (effectively, resetting the property):

- Save
- SavePage
- SaveAs

See Also AddAnnotationGroup method, AutoCrop method, AutoDeskew method, BurnInAnnotations method, ClipboardCut method, ClipboardPaste method, Close event, ConvertPageType method, Crop method, DeleteAnnotationGroup method, DeleteImageData method, DeleteSelectedAnnotations method, Despeckle method, Draw method, EditSelectedAnnotationText method, Flip method, HideAnnotationGroup method, ImageResolutionX property, ImageResolutionY property, Invert method, ManualDeskew method, Redo method, RemoveAllOCRMarks method, Rotate method, RotateAll method, RotateRight method, Save method, SaveAs method, SavePage method, SetSelectedAnnotationBackColor method, SetSelectedAnnotationFillColor method, SetSelectedAnnotationFillStyle method, SetSelectedAnnotationFont method, SetSelectedAnnotationFontColor method, SetSelectedAnnotationLineColor method, SetSelectedAnnotationLineStyle method, SetSelectedAnnotationLineWidth method, SetSelectedAnnotationOcrType method, ShowAnnotationGroup method, ShowAnnotationToolPalette method, Undo method.

ImageModified Example — VB

This example uses the ImageModified property to see if changes were made to an existing image before permitting the user to open another. If the value of the ImageModified property is True, the code prompts the user to save the image before opening the next one.

```
Private Sub ImgEdit1_Close()
    'This code could be used at the time a user attempts to open a new file
    'or clear a previously displayed image in order to prompt for changes
    'to be saved if the image has been modified.
    Dim intSaveImg As Integer
    If ImgEdit1.ImageModified = True Then
      intSaveImg = MsgBox("Do you want to save changes?", vbYesNoCancel,
      ➥ "Image has been modified")
      Select Case intSaveImg
        Case vbYes
          'Save the changes
          ImgEdit1.Save
          'Drop down to the Open file code
        Case vbNo
          'Don't save file, just drop down to the Open file code
        Case vbCancel
          'User pressed cancel, don't open a file, don't save changes
          Exit Sub
      End Select
    End If
    'Open a new file at this point
End Sub
```

ImageModified Example — VC++

This example uses the ImageModified property to see if changes were made to an existing image before permitting the user to open another. If the value of the ImageModified property is True, the code prompts the user to save the image before opening the next one.

```
void CImgEdit1Dlg::OnCloseEditctrl1()
{
    // This code could be used at the time a user attempts to open a new
    // file or clear a previously displayed image in order to prompt for
    // changes to be saved if the image has been modified.
    int iSaveImg;
    if(ImgEdit1.GetImageModified())
    {
      iSaveImg = AfxMessageBox("Do you want to save changes?",
      ➥  MB_YESNOCANCEL);
      switch (iSaveImg)
      {
        case IDYES:
          // Save the changes
          VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default for save at
                                              // zoom
          ImgEdit1.Save(evt);
          break;
          // Drop down to the Open file code
```

```
            case IDNO:
              // Don't save file, just drop down to the Open file code
              break;
            case IDCANCEL:
              // User pressed cancel, don't open a file, don't save changes
              return;
            }
        }
        // Open a new file at this point
}
```

ImagePalette Property

Returns or sets the palette to use when displaying an image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Integer (enumerated).

Property settings are expressed using the following values:

Constant	Setting	Description	Notes
wiPaletteCustom	0 (default)	Custom	Selects a custom palette.
wiPaletteCommon	1	Common	Selects the common palette.
wiPaletteGray8	2	Gray8	Selects the 8-bit gray-scale, 256 shades-of-gray palette.
wiPaletteRGB24	3	RGB24	Selects the 24-bit, millions-of-colors palette.
wiPaletteBlackWhite	4	Black-and-white	Selects the black-and-white palette.

Remarks The ImagePalette value can be specified before or after an image is displayed. If the value is specified after an image is displayed, the displayed image may change visually.

When the **AutoRefresh** property is set to True, changing the ImagePalette property causes the displayed image to be refreshed automatically.

When the Image Edit and Image Thumbnail controls are used together, the ImagePalette property must be set to the Common palette (wiPaletteCommon) when the PageType of the image being displayed in the Image Edit control is **not** RGB.

When the PageType of the image being displayed in the Image Edit control is RGB, the ImagePalette property must be set to the RGB24 palette (wiPaletteRGB24).

See Also AutoRefresh property, SetImagePalette method.

ImageResolutionX Property

Returns or sets the horizontal resolution of the image being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Long.

Remarks When the **AutoRefresh** property is set to True, changing the ImageResolutionX property causes the displayed image to be refreshed automatically.

Changing the setting of the ImageResolutionX property sets the **ImageModified** property to True.

If an image is not being displayed, the ImageResolutionX property returns or sets the horizontal resolution of the image specified in the **Image** property.

If an image is not being displayed and the Image property does not contain the name of an image, the ImageResolutionX property generates an error.

Available at run-time as read-and-write.

See Also AutoRefresh property, Image property, ImageModified property, ImageResolutionY property.

ImageResolutionX Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
```

```
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageResolutionX Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

ImageResolutionY Property

Description Returns or sets the vertical resolution of the image being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Long.

Remarks When the **AutoRefresh** property is set to True, changing the ImageResolutionY property causes the displayed image to be refreshed automatically.

Changing the setting of the ImageResolutionY property sets the **ImageModified** property to True.

If an image is not being displayed, the ImageResolutionY property returns or sets the vertical resolution of the image specified in the **Image** property.

If an image is not being displayed and the Image property does not contain the name of an image, the ImageResolutionY property generates an error.

Available at run-time as read-and-write.

See Also AutoRefresh property, Image property, ImageModified property, ImageResolutionX property.

ImageResolutionY Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
```

```
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageResolutionY Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

ImageScaleHeight Property

Description Returns the height in pixels of the image as it is displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageScaleHeight**

Data Type OLE_YSIZE_PIXELS.

Remarks The **Display** method must be invoked prior to using this property.

The ImageScaleHeight property differs from the **ImageHeight** property. The ImageHeight property returns the height in full-size coordinates, while the ImageScaleHeight property value is affected by the current zoom factor.

The ImageScaleHeight and ImageHeight properties also differ for images with asymmetric resolutions. For example, when an 8½- by 11-inch image with a resolution of 200 by 100 dots per inch (dpi) is displayed, the ImageHeight property returns a value of 1100, while the ImageScaleHeight property returns a value of 2200 due to pixel replication that occurs at display-time to optimize display quality.

Available at run-time as read-only.

See Also Display method, Image property, ImageHeight property, ImageResolutionX property, ImageResolutionY property, ImageScaleWidth property, Zoom property (Image Edit).

ImageScaleHeight Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageScaleHeight Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

ImageScaleWidth Property

Description Returns the width in pixels of the image as it is displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageScaleWidth**

Data Type OLE_XSIZE_PIXELS.

Remarks The **Display** method must be invoked prior to using this property.

The ImageScaleWidth property differs from the **ImageWidth** property. The ImageWidth property returns the width in full-size coordinates, while the ImageScaleWidth property value is affected by the current zoom factor.

The ImageScaleWidth and ImageWidth properties also differ for images with asymmetric resolutions. For example, when an 8½- by 11-inch image with a resolution of 100 by 200 dots per inch (dpi) is displayed, the ImageWidth property returns a value of 850, while the ImageScaleWidth property returns a value of 1700 due to pixel replication that occurs at display-time to optimize display quality.

Available at run-time as read-only.

See Also Display method, Image property, ImageResolutionX property, ImageResolutionY property, ImageScaleHeight property, ImageWidth property, Zoom property (ImageEdit).

ImageScaleWidth Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageScaleWidth Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
```

```
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

ImageWidth Property

Description Returns the width in full-size coordinates of the image being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ImageWidth**

Data Type Long.

Remarks If an image is not being displayed, this property returns the full-size width of the image specified in the **Image** property. If an image is not being displayed and the Image property does not contain the name of an image, the ImageWidth property generates an error.

Available at run-time as read-only.

See Also Image property, ImageHeight property, ImageScaleWidth property.

ImageWidth Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

ImageWidth Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height
    lInfo = ImgEdit1.GetImageScaleHeight();
```

```
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

MagnifierZoom Property

Description Returns or sets the zoom factor used by the magnifier window.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**MagnifierZoom**[=*value*]

Data Type Integer.

Property settings are expressed using the following values:

Constant	Setting	Description
wiLockMagnifier	-1	Locks the displayed image in the magnifier window at the current **Zoom** factor until the MagnifierZoom property is set to 0, 1, or 2.
wi2Xmagnifier	0 (default)	2 times the current Zoom factor.
wi4Xmagnifier	1	4 times the current Zoom factor.
wi8Xmagnifier	2	8 times the current Zoom factor.

Remarks Magnification is limited to 6500%.

See Also MagnifierStatus event, ShowMagnifier method, Zoom property (ImageEdit).

MagnifierZoom Example — VB

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
Private Sub cmdMagnifier_Click()
    ImgEdit1.ScrollPositionX = 800
    ImgEdit1.ScrollPositionY = 1000
    'Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.MagnifierZoom = wi4XMagnifier  '1
    ImgEdit1.ShowMagnifier True
End Sub
```

MagnifierZoom Example — VC++

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
void CImgEdit1Dlg::OnMagnifier()
{
    // This example would scroll the image to a specific location and then
    // display a magnifier window. Scroll the image to a specific
    // horizontal/vertical pixel location.
    ImgEdit1.SetScrollPositionX(800);
    ImgEdit1.SetScrollPositionY(1000);
    // Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.SetMagnifierZoom(1); // wi4XMagnifier  // 1
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ShowMagnifier(TRUE,evt,evt,evt,evt);
}
```

MouseIcon Property

Description Returns or sets the mouse icon to be displayed when the mouse pointer is moved over an Image Edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**MouseIcon**=**LoadPicture**(*pathname*)

Data Type String.

The pathname is a string expression specifying the path and file name of the file containing the custom icon (.CUR or .ICO files).

Remarks The MouseIcon property provides a custom icon that is used when the **MousePointer** property is set to 99.

See Also MousePointer property.

MouseIcon Example — VB

This example sets a custom mouse pointer. The pointer will change when it is placed over the ImageEdit control.

```
Private Sub cmdMousePointer_Click()
    Dim strMIcon As String
    strMIcon = "C:\Program Files\DevStudio\VB\samples\PGuide\
    ➥  Optimize\Litening.ico"
    ImgEdit1.MouseIcon = LoadPicture(strMIcon)
    ImgEdit1.MousePointer = wiMPCustom '99
End Sub
```

MouseIcon Example — VC++

This example sets a custom mouse pointer. The pointer will change when it is placed over the ImageEdit control.

```
void CImgEdit1Dlg::OnMouseIcon()
{
    // Sets a custom mouse pointer -- pointer will change when placed over
    // the ImgEdit control.
    // Sets the specific picture icon
    CPictureHolder pic1;
    // Create the icon
    pic1.CreateFromIcon(IDR_MAINFRAME);
    CPicture cpPicU(pic1.GetPictureDispatch());
    // Specifies the custom mouse pointer to be used when the mouse moves
    // over the control
    // The custom MousePointer must be set.
    ImgEdit1.SetMouseIcon(cpPicU);
    ImgEdit1.SetMousePointer(99);  // wiMPCustom // 99
}
```

MousePointer Property

Description Returns or sets the type of mouse pointer to be displayed when the pointer is moved over an Image Edit control at run-time.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Note: The Hyperlink and OCR Zone MousePointer icons are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

Applies To Image Edit control.

Usage *object*.**MousePointer**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiMPDefault	0 (Default)	As determined by the value of the **AnnotationType** property
wiMPArrow	1	Arrow
wiMPCross	2	Cross (cross-hair pointer)
wiMPIBeam	3	I-Beam
wiMPIcon	4	Arrow
wiMPSize	5	Size (four-pointed arrow pointing North, South, East, and West)
wiMPSizeNESW	6	Size NE SW (double arrow pointing NorthEast and SouthWest)
wiMPSizeNS	7	Size N S (double arrow pointing North and South)
wiMPSizeNWSE	8	Size NW SE (double arrow pointing NorthWest and SouthEast)
wiMPSizeWE	9	Size W E (double arrow pointing West and East)
wiMPUpArrow	10	Up Arrow
wiMPHourGlass	11	Hourglass (wait)
wiMPNoDrop	12	No Drop
wiMPArrowHourglass	13	Arrow and Hourglass
wiMPArrowQuestion	14	Arrow and Question mark
wiMPSizeAll	15	Size All
wiMPFreehandLine	16	Freehand Line
wiMPHollowRect	17	Hollow Rectangle
wiMPFilledRect	18	Filled Rectangle
wiMPRubberStamp	19	Text Stamp
wiMPText	20	Text
wiMPTextFromFile	21	Text From File
wiMPTextAttachment	22	Attach-a-Note
wiMPHand	23	Hand

Constant	Setting	Description
wiMPSelect	24	Image Selection
wiMPHyperlink	25	Hyperlink
wiMPOcrRegion	26	OCR Zone
wiMPCustom	99	Custom pointer specified by the **MouseIcon** property

Remarks The MouseIcon property must be set prior to setting the MousePointer property to 99.

Use this property to indicate changes in functionality as the mouse pointer passes over controls on a form or dialog box. Use the Hourglass setting (11) to indicate that users should wait for a process or operation to finish.

See Also AnnotationType property, MouseIcon property.

MousePointer Example — VB

This example sets a custom mouse pointer. The pointer will change when it is placed over the ImageEdit control.

```
Private Sub cmdMousePointer_Click()
    Dim strMIcon As String
    strMIcon = "C:\Program Files\DevStudio\VB\samples\PGuide
    ➥  \Optimize\Litening.ico"
    ImgEdit1.MouseIcon = LoadPicture(strMIcon)
    ImgEdit1.MousePointer = wiMPCustom '99
End Sub
```

MousePointer Example — VC++

This example sets a custom mouse pointer. The pointer will change when it is placed over the ImageEdit control.

```
void CImgEdit1Dlg::OnMouseIcon()
{
    // Sets a custom mouse pointer -- pointer will change when placed over
    // the ImgEdit control
    // Sets the specific picture icon
    CPictureHolder pic1;
    // Create the icon
    pic1.CreateFromIcon(IDR_MAINFRAME);
    CPicture cpPicU(pic1.GetPictureDispatch());
    // Specifies the custom mouse pointer to be used
    // when the mouse moves over the control.
    // The custom MousePointer must be set.
    ImgEdit1.SetMouseIcon(cpPicU);
    ImgEdit1.SetMousePointer(99);  // wiMPCustom // 99
}
```

OcrZoneVisibility Property

Description Returns or sets the OCR zones that are shown on the image page.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**OcrZoneVisibility**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiOcrHideZones	0 (default)	No OCR zones are shown
wiOcrShowTextZones	1	All OCR Text zones are shown
wiOcrShowPictureZones	2	All OCR Picture zones are shown
wiOcrShowAllZones	3	All OCR Text and Picture zones are shown

Remarks This property does not affect the visibility of other annotations on the image page. To hide and show other annotation types, use the **HideAnnotationGroup** and **ShowAnnotationGroup** methods.

See Also AnnotationOcrType property, AnnotationType property, GetSelectedAnnotationOcrType method, HideAnnotationGroup method, Image OCR control, RemoveAllOCRMarks method, SelectFirstOcrZone method, SelectNextOcrZone method, SetSelectedAnnotationOcrType method, ShowAnnotationGroup method.

OCRZoneVisibility Example — VB

This example shows some property settings you might want to make prior to displaying an image.

```
Private Sub ImgEdit1_Load(ByVal Zoom As Double)
    'Here are some examples of default settings you might want to specify
    'prior to displaying an image. Repaint any changes immediately
    '(ex. zoom or resolution changes, etc.)
    ImgEdit1.AutoRefresh = True
    'ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.ScrollBars = True
    'Scale black and white images to grayscale.
    'Keep color image displayed as color images.
    ImgEdit1.DisplayScaleAlgorithm = wiScaleOptimize
    'Allow user to scroll image using keyboard shortcuts
```

```
    ImgEdit1.ScrollShortcutsEnabled = True
    'Display all OCR zones
    ImgEdit1.OcrZoneVisibility = wiOcrShowAllZones  '3
End Sub
```

OCRZoneVisibility Example — VC++

This example shows some property settings you might want to make prior to displaying an image.

```
void CImgEdit1Dlg::OnLoadEditctrl1(double Zoom)
{
    // Here are some examples of default settings you might want to specify
    // prior to displaying an image. Repaint any changes immediately
    // (ex. zoom or resolution changes, etc.)
    ImgEdit1.SetAutoRefresh(TRUE);
    // ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.SetScrollBars(TRUE);
    // Scale black and white images to grayscale.
    // Keep color image displayed as color images.
    ImgEdit1.SetDisplayScaleAlgorithm(4);   // wiScaleOptimize
    // Allow user to scroll image using keyboard shortcuts
    ImgEdit1.SetScrollShortcutsEnabled(TRUE);
    // Display all OCR zones
    ImgEdit1.SetOcrZoneVisibility(3);  // wiOcrShowAllZones  // 3
}
```

Page Property

Description Returns the page number of an image file where an action was performed, or sets the page number of an image file where an action is to be performed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Page**[=*value*]

Data Type Long.

Remarks The default value for this property is 1. This property is typically used to specify the page to be displayed by the **Display** method.

See Also Display method, PageCount property.

Page Example — VB

This example displays the next page in a multipage file, and determines whether there is another page to display.

```
Private Sub cmdNextPage_Click()
    Dim lngNewPg As Long
    lngNewPg = ImgEdit1.Page + 1
    'Determine if there is a next page to be displayed by finding out
    'the number of pages in the file
    If lngNewPg > ImgEdit1.PageCount Then
        MsgBox "Page Number is out of range"
        'Don't set page property
        Exit Sub
    End If
    ImgEdit1.Page = lngNewPg
    ImgEdit1.Display
End Sub
```

Page Example — VC++

This example displays the next page in a multipage file, and determines whether there is another page to display.

```
void CImgEdit1Dlg::OnNextPage()
{
    // This would display next page in a multipage file
    long lngNewPg;
    lngNewPg = ImgEdit1.GetPage() + 1;
    // Determine if there is a next page to be displayed by finding out
    // the number of pages in the file
    if(lngNewPg > ImgEdit1.GetPageCount())
    {
        AfxMessageBox("Page Number is out of range");
        // Don't set page property
        return;
    }
    ImgEdit1.SetPage(lngNewPg);
    ImgEdit1.Display();
}
```

PageCount Property

Description Returns the total number of pages in the image file that is currently being displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**PageCount**

Data Type Long.

Remarks If an image is not being displayed, the PageCount property returns the total number of pages in the image file specified in the **Image** property.

If an image is not being displayed and the Image property does not contain the name of an image, the PageCount property generates an error.

Available at run-time as read-only.

See Also Image property, Page property.

PageCount Example — VB

This example displays the next page in a multipage file, and determines whether there is another page to display.

```
Private Sub cmdNextPage_Click()
    Dim lngNewPg As Long
    lngNewPg = ImgEdit1.Page + 1
    'Determine if there is a next page to be displayed by finding out
    'the number of pages in the file
    If lngNewPg > ImgEdit1.PageCount Then
        MsgBox "Page Number is out of range"
        'Don't set page property
        Exit Sub
    End If
    ImgEdit1.Page = lngNewPg
    ImgEdit1.Display
End Sub
```

PageCount Example — VC++

This example displays the next page in a multipage file, and determines whether there is another page to display.

```
void CImgEdit1Dlg::OnNextPage()
{
    // This would display next page in a multipage file
    long lngNewPg;
    lngNewPg = ImgEdit1.GetPage() + 1;
    // Determine if there is a next page to be displayed by finding out
    // the number of pages in the file
    if(lngNewPg > ImgEdit1.GetPageCount())
    {
        AfxMessageBox("Page Number is out of range");
        // Don't set page property
        return;
    }
```

```
    ImgEdit1.SetPage(lngNewPg);
    ImgEdit1.Display();
}
```

PageType Property

Description Returns the page type of the image specified in the **Image** property and the page specified in the **Page** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**PageType**

Data Type Integer (enumerated).

Property settings are expressed using the following values:

Constant	Setting	Description
wiPageTypeBW	1	Black-and-white
wiPageTypeGray4	2	Gray scale, 4-bit
wiPageTypeGray8	3	Gray scale, 8-bit
wiPageTypePal4	4	Palettized, 4-bit
wiPageTypePal8	5	Palettized, 8-bit
wiPageTypeRGB24	6	RGB, 24-bit
wiPageTypeBGR24	7	BGR, 24-bit

Remarks The PageType property value is independent of an image being displayed. If the image page specified in the Page property is being displayed, the value is the page type of the displayed image. If the image page specified in the Image and Page properties is not being displayed, the value is the page type of the page specified in the Image and Page properties.

Available at run-time as read-only.

See Also ConvertPageType method, Page property, SaveAs method, SavePage method.

PageType Example — VB

This example uses the PageType property to determine the page type of the image specified in the Image property.

```
Private Sub cmdPageType_Click()
    'Reads the PageType property to determine the page type of
    'the image specified in the Image property
    Dim strImgPageType As String
    'Read the PageType property and translate the value to a
    'corresponding string.
    Select Case ImgEdit1.PageType
      Case 1
        strImgPageType = "Black and White"
      Case 2
        strImgPageType = "4 bit grayscale"
      Case 3
        strImgPageType = "8 bit grayscale"
      Case 4
        strImgPageType = "4 bit palettized"
      Case 5
        strImgPageType = "8 bit palettized"
      Case 6
        strImgPageType = "24 bit RGB"
      Case 7
        strImgPageType = "24 bit BGR"
    End Select
    'Display the page type on a form for informational purposes, or to
    'allow it to be modified.
    frmInfo.lblImgInfo(1).Caption = strImgPageType
End Sub
```

PictureDisabled Property

Description Returns or sets the bitmap image that appears on the control when its **Enabled** property is set to False (disabled).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Annotation control.

Usage *object*.**PictureDisabled**[=*picture*]

Data Type Picture.

Remarks The bitmap is scaled to fit the size of the control. If no bitmap is specified, the default bitmap is used.

See Also Enabled property, PictureDown property, PictureUp property.

PictureDisabled Example — VB

This example demonstrates how to use the PictureUp, PictureDown, and PictureDisabled properties to display the appropriate bitmap images on an Image Annotation Tool Button control.

PictureUp — Sets the bitmap picture that is displayed when the button is enabled, but not selected.

PictureDown — Sets the bitmap picture that is displayed when the button is selected.

PictureDisabled — Sets the bitmap picture that is displayed when the button is disabled.

```
Private Sub frmToolPal_Load()
    'When using the annotation control to create a custom tool palette, you
    'can designate bitmaps to be used on the tools when they are in various
    'states. If these values are not specified, default values will be used.
    'Note that it may be more efficient to set these properties at
    'design time.
    'Defines what picture will be shown if the button is enabled
    ImgAnnTool1.PictureUp = LoadPicture("D:\yourapp\lineup.bmp")
    'Defines what picture will be shown when the button is selected
    ImgAnnTool1.PictureDown = LoadPicture("D:\yourapp\linedown.bmp")
    'Defines what picture will be shown when the button is disabled (grayed)
    ImgAnnTool1.PictureDisabled = LoadPicture("D:\yourapp\lineoff.bmp")
End Sub
```

PictureDown Property

Description Returns or sets the bitmap image that appears on the control when it is selected. A control is selected when an end user clicks on it, or when its **Value** property is set to True within a program.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Annotation control.

Usage *object*.**PictureDown**[=*picture*]

Data Type Picture.

Remarks The bitmap is scaled to fit the size of the control. If no bitmap is specified, the default bitmap is used.

The following remark applies when more than one Image Annotation Tool Button control is used with an Image Edit control (in other words, when each button has the same **DestImageControl** property value):

When an Image Annotation Tool Button control is clicked by the end user, its Value property changes to True and its bitmap changes to the image assigned to its PictureDown property. For each of the other Image Annotation Tool Button controls, the **Value** property changes to False and the bitmap assigned to its **PictureUp** property is displayed.

See Also DestImageControl property, PictureDisabled property, PictureUp property, Value property.

PictureDown Example — VB

This example demonstrates how to use the PictureUp, PictureDown, and PictureDisabled properties to display the appropriate bitmap images on an Image Annotation Tool Button control.

PictureUp — Sets the bitmap picture that is displayed when the button is enabled, but not selected.

PictureDown — Sets the bitmap picture that is displayed when the button is selected.

PictureDisabled — Sets the bitmap picture that is displayed when the button is disabled.

```
Private Sub frmToolPal_Load()
    'When using the annotation control to create a custom tool palette, you 'can designate
     bitmaps to be used on the tools when they are in various
    'states. If these values are not specified, default values will be used.
    'Note that it may be more efficient to set these properties at
    'design time.
    'Defines what picture will be shown if the button is enabled
    ImgAnnTool1.PictureUp = LoadPicture("D:\yourapp\lineup.bmp")
    'Defines what picture will be shown when the button is selected
    ImgAnnTool1.PictureDown = LoadPicture("D:\yourapp\linedown.bmp")
    'Defines what picture will be shown when the button is disabled (grayed)
    ImgAnnTool1.PictureDisabled = LoadPicture("D:\yourapp\lineoff.bmp")
End Sub
```

PictureUp Property

Description Returns or sets the bitmap image that appears on the control when it is enabled but not selected. A control is enabled when its **Enabled** property is set to True. A control is not selected (deselected) when the end user has not clicked on it, or when its **Value** property is set to False within a program.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Annotation control.

Usage *object*.**PictureUp**[=*picture*]

Data Type Picture.

Remarks The bitmap is scaled to fit the size of the control. If no bitmap is specified, the default bitmap is used.

See Also Enabled property, PictureDisabled property, PictureDown property, Value property.

PictureUp Example — VB

This example demonstrates how to use the PictureUp, PictureDown, and PictureDisabled properties to display the appropriate bitmap images on an Image Annotation Tool Button control.

PictureUp — Sets the bitmap picture that is displayed when the button is enabled, but not selected.

PictureDown — Sets the bitmap picture that is displayed when the button is selected.

PictureDisabled — Sets the bitmap picture that is displayed when the button is disabled.

```
Private Sub frmToolPal_Load()
    'When using the annotation control to create a custom tool palette, you
    'can designate bitmaps to be used on the tools when they are in various
    'states. If these values are not specified, default values will be used.
    'Note that it may be more efficient to set these properties at
    'design time.
    'Defines what picture will be shown if the button is enabled
    ImgAnnTool1.PictureUp = LoadPicture("D:\yourapp\lineup.bmp")
    'Defines what picture will be shown when the button is selected
    ImgAnnTool1.PictureDown = LoadPicture("D:\yourapp\linedown.bmp")
    'Defines what picture will be shown when the button is disabled (grayed)
    ImgAnnTool1.PictureDisabled = LoadPicture("D:\yourapp\lineoff.bmp")
End Sub
```

ReadyState Property

Description Returns a value that indicates the readiness of the control to respond to methods. The **ReadyStateChange** event fires when the readiness state of the control changes.

The ReadyState property and ReadyStateChange event are intended for use with HTML scripts only. Use them to manage the asynchronous downloading of images from the World Wide Web.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**ReadyState**[=*value*]

Data Type Long.

Property settings are expressed using the following values:

Setting	Description
0 (default)	The control is initializing and retrieving properties; cannot perform methods.
2	The control has initialized and downloading has started; cannot perform methods.
4	The control is ready for all requests; can perform methods.

Remarks This property is useful when the **Image** property is set to a URL pathname.

The control performs methods only when its ReadyState property is 4.

Available at run-time as read-only.

See Also Image property, ReadyStateChange event.

ScrollBars Property

Description Returns or sets whether a displayed image contains horizontal and/or vertical scroll bars.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ScrollBars**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Horizontal and/or vertical scroll bars are displayed.
False	No scroll bars are displayed.

Remarks If this property is set to True, horizontal and/or vertical scroll bars are displayed only when the image extends beyond the object's borders. If the image has scroll bars and the **ScrollShortcutsEnabled** property is set to True, accelerator keys are provided to permit users to scroll an image using the keyboard.

When the **AutoRefresh** property is set to True, changing the ScrollBars property causes the displayed image to be refreshed automatically.

See Also AutoRefresh property, Scroll event, ScrollImage method, ScrollPositionX property, ScrollPositionY property, ScrollShortcutsEnabled property.

ScrollBars Example — VB

This example shows some property settings you might want to make prior to displaying an image.

```
Private Sub ImgEdit1_Load(ByVal Zoom As Double)
    'Here are some examples of default settings you might want to specify
    'prior to displaying an image. Repaint any changes immediately
    '(ex. zoom or resolution changes, etc.)
    ImgEdit1.AutoRefresh = True
    'ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.ScrollBars = True
    'Scale black and white images to grayscale. Keep color image
    'displayed as color images.
    ImgEdit1.DisplayScaleAlgorithm = wiScaleOptimize
    'Allow user to scroll image using keyboard shortcuts
    ImgEdit1.ScrollShortcutsEnabled = True
    'Display all OCR zones
    ImgEdit1.OcrZoneVisibility = wiOcrShowAllZones  '3
End Sub
```

ScrollBars Example — VC++

This example shows some property settings you might want to make prior to displaying an image.

```
void CImgEdit1Dlg::OnLoadEditctrl1(double Zoom)
{
    // Here are some examples of default settings you might want to specify
    // prior to displaying an image. Repaint any changes immediately
    // (ex. zoom or resolution changes, etc.)
    ImgEdit1.SetAutoRefresh(TRUE);
    // ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.SetScrollBars(TRUE);
    // Scale black and white images to grayscale. Keep color image
    // displayed as color images.
    ImgEdit1.SetDisplayScaleAlgorithm(4);   // wiScaleOptimize
    // Allow user to scroll image using keyboard shortcuts
    ImgEdit1.SetScrollShortcutsEnabled(TRUE);
    // Display all OCR zones
    ImgEdit1.SetOcrZoneVisibility(3);  // wiOcrShowAllZones  // 3
}
```

ScrollPositionX Property

Description Returns or sets the horizontal scroll position in pixels.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Long.

Remarks Setting the ScrollPositionX property causes the displayed image to scroll horizontally to the pixel position specified.

To set the scroll position prior to displaying an image, the ScrollPositionX and **ScrollPositionY** properties must be set in the **Load** event.

When the image is first displayed using the **Display** method, this property is set to 0 by default. When the **AutoRefresh** property is set to True, changing the ScrollPositionX property causes the displayed image to be refreshed automatically.

Available at run-time as read-and-write.

See Also AutoRefresh property, Display method, Load event, Scroll event, ScrollBars property, ScrollImage method, ScrollPositionY property.

ScrollPositionX Example — VB

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
Private Sub cmdMagnifier_Click()
    ImgEdit1.ScrollPositionX = 800
    ImgEdit1.ScrollPositionY = 1000
    'Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.MagnifierZoom = wi4XMagnifier  '1
    ImgEdit1.ShowMagnifier True
End Sub
```

ScrollPositionX Example — VC++

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
void CImgEdit1Dlg::OnMagnifier()
{
    // This example would scroll the image to a specific location and then
    // display a magnifier window. Scroll the image to a specific
    // horizontal/vertical pixel location.
    ImgEdit1.SetScrollPositionX(800);
    ImgEdit1.SetScrollPositionY(1000);
    // Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.SetMagnifierZoom(1); // wi4XMagnifier  // 1
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ShowMagnifier(TRUE,evt,evt,evt,evt);
}
```

ScrollPositionY Property

Description Returns or sets the vertical scroll position in pixels.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Long.

Remarks Setting the ScrollPositionY property causes the displayed image to scroll vertically to the pixel position specified.

To set the scroll position prior to displaying an image, the **ScrollPositionX** and ScrollPositionY properties must be set in the **Load** event.

When the image is first displayed using the **Display** method, this property is set to 0 by default.

When the **AutoRefresh** property is set to True, changing the ScrollPositionY property causes the displayed image to be refreshed automatically.

Available at run-time as read-and-write.

See Also AutoRefresh property, Display method, Load event, Scroll event, ScrollBars property, ScrollImage method, ScrollPositionX property.

ScrollPositionY Example — VB

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
Private Sub cmdMagnifier_Click()
    ImgEdit1.ScrollPositionX = 800
    ImgEdit1.ScrollPositionY = 1000
    'Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.MagnifierZoom = wi4XMagnifier  '1
    ImgEdit1.ShowMagnifier True
End Sub
```

ScrollPositionY Example — VC++

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
void CImgEdit1Dlg::OnMagnifier()
{
    // This example would scroll the image to a specific location and then
    // display a magnifier window. Scroll the image to a specific
    // horizontal/vertical pixel location.
    ImgEdit1.SetScrollPositionX(800);
    ImgEdit1.SetScrollPositionY(1000);
    // Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.SetMagnifierZoom(1); // wi4XMagnifier  // 1
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ShowMagnifier(TRUE,evt,evt,evt,evt);
}
```

ScrollShortcutsEnabled Property

Description Returns or sets whether the object recognizes a set of accelerator keys for scrolling a displayed image provided horizontal and/or vertical scroll bars are present.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ScrollShortcutsEnabled**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Accelerator keys can be used to scroll an image.
False	Accelerator keys cannot be used.

Remarks If this property and the **ScrollBars** property are set to True, and the image extends beyond the object's borders, end users can use accelerator keys to scroll the displayed image.

Under certain circumstances, it may be necessary to set the focus to the Image Edit control for the accelerator keys to function.

The following accelerator keys provide scrolling capabilities:

Accelerator Key	Description
Page Down	Scrolls the image downward by one full screen.
Page Up	Scrolls the image upward by one full screen.
Ctrl + Page Down	Scrolls the image to the right by one full screen.
Ctrl + Page Up	Scrolls the image to the left by one full screen.
Right Arrow	Scrolls the image to the right by 4 pixels.
Left Arrow	Scrolls the image to the left by 4 pixels.
Up Arrow	Scrolls the image upward by 4 pixels.
Down Arrow	Scrolls the image downward by 4 pixels.

See Also Scroll event, ScrollBars property, ScrollImage method.

ScrollShortcutsEnabled Example — VB

This example shows some property settings you might want to make prior to displaying an image.

```
Private Sub ImgEdit1_Load(ByVal Zoom As Double)
    'Here are some examples of default settings you might want to specify
    'prior to displaying an image. Repaint any changes immediately
    '(ex. zoom or resolution changes, etc.)
    ImgEdit1.AutoRefresh = True
    'ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.ScrollBars = True
    'Scale black and white images to grayscale. Keep color image
    'displayed as color images.
    ImgEdit1.DisplayScaleAlgorithm = wiScaleOptimize
    'Allow user to scroll image using keyboard shortcuts
    ImgEdit1.ScrollShortcutsEnabled = True
    'Display all OCR zones
    ImgEdit1.OcrZoneVisibility = wiOcrShowAllZones  '3
End Sub
```

ScrollShortcutsEnabled Example — VC++

This example shows some property settings you might want to make prior to displaying an image.

```
void CImgEdit1Dlg::OnLoadEditctrl1(double Zoom)
{
    // Here are some examples of default settings you might want to specify
    // prior to displaying an image. Repaint any changes immediately
    // (ex. zoom or resolution changes, etc.)
    ImgEdit1.SetAutoRefresh(TRUE);
    // ScrollBars already default to true, but this is modifiable if desired
    ImgEdit1.SetScrollBars(TRUE);
    // Scale black and white images to grayscale. Keep color image
    // displayed as color images.
    ImgEdit1.SetDisplayScaleAlgorithm(4);   // wiScaleOptimize
    // Allow user to scroll image using keyboard shortcuts
    ImgEdit1.SetScrollShortcutsEnabled(TRUE);
    // Display all OCR zones
    ImgEdit1.SetOcrZoneVisibility(3);  // wiOcrShowAllZones  // 3
}
```

SelectionRectangle Property

Description Returns or sets whether a selection box will be drawn when end users click the left mouse button and drag the mouse pointer over a displayed image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SelectionRectangle**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Selection box drawing is enabled.
False	Selection box drawing is not enabled.

Remarks Once an end user draws a selection box, cutting and copying to the Clipboard can be accomplished.

See Also ClipboardCopy method, ClipboardCut method, SelectionRectDrawn event, ZoomToSelection method.

SelectionRectangle Example — VB

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
Private Sub cmdZoomSelect_Click()
    'Enable rectangle drawing
    ImgEdit1.SelectionRectangle = True
    'Draw rectangle
    ImgEdit1.DrawSelectionRect 150, 200, 400, 400
    ImgEdit1.ZoomToSelection
End Sub
```

SelectionRectangle Example — VC++

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
void CImgEdit1Dlg::OnZoomto()
{
    // This would draw a rectangle on an image and would zoom in on the area
    // bound by the rectangle, and enable rectangle drawing
    ImgEdit1.SetSelectionRectangle(TRUE);
    // Draw rectangle
    ImgEdit1.DrawSelectionRect( 150, 200, 400, 400);
    ImgEdit1.ZoomToSelection();
}
```

StatusCode Property

Description Returns a status code, which identifies the error that occurred when the last property was set or read, or the last method was invoked. A status code of 0 indicates that no error has occurred.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit, Image Scan, Image Admin (Win 98 and Pro V2.0 only), Image Annotation, and Image Thumbnail (Win 98 and Pro V2.0 only) controls.

Usage *object*.**StatusCode**

Data Type Long.

Remarks The default value for this property is 0.

Available at run-time as read-only.

UndoLevels Property

Description Returns or sets the number of permissible undo levels.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**UndoLevels**[=*value*]

Data Type Long.

Setting	Description
0 (default)	No levels of undo
1	1 level of undo

Remarks The **Undo** method is limited to the number of undo levels specified in this property.

See Also ContinueWithoutUndo property, ErrorSavingUndoInformation event, Redo method, Undo method.

UndoLevels Example — VB

This example illustrates how to enable Undo functionality in an application.

```
Private Sub Form_Load()
    'UndoLevels must be set at form load to allocate undo buffers
    ImgEdit1.UndoLevels = 1
End Sub
Private Sub cmdUndo_Click()
    'The Undo method will undo the last imaging operation performed.
    ImgEdit1.Undo
End Sub
```

UndoLevels Example — VC++

This example illustrates how to set Undo levels on initialization.

```
BOOL CImgEdit1Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    blnIsRectangle = FALSE;
    // UndoLevels must be set at form load to allocate undo buffers
    ImgEdit1.SetUndoLevels(1);
    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

```
void CImgEdit1Dlg::OnUndo()
{
    // The Undo method will undo the last imaging operation performed.
    VARIANT evt; V_VT(&evt) = VT_ERROR;    // set to default
    ImgEdit1.Undo(evt);
}
```

UseCheckContinuePrinting Property

Description Returns or sets how the control handles a print cancellation/continuation request.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**UseCheckContinuePrinting**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Print cancellation/continuation is handled through the CheckContinuePrinting event.
False (default)	Standard print cancellation/continuation handling.

Remarks Setting this property to True enables you to use the **ContinuePrinting** property and the **CheckContinuePrinting** event to handle a print cancellation/continuation request.

The CheckContinuePrinting event fires whenever a print operation requests whether it should cancel or continue printing. The ContinuePrinting property provides the event with the desired response.

See Also CheckContinuePrinting event, ContinuePrinting property, PrintImage method.

Value Property

Description Selects or deselects the control at run-time. Setting this property to True selects the control, which is the same as an end user clicking on it. Setting this property to False deselects the control.

When a control is selected, the following events occur:

- The bitmap specified in the **PictureDown** property is drawn on the control
- Other controls on the form with the same **DestImageControl** property value have their Value property set to False automatically
- The **AnnotationType** property in the Image Edit control is changed to the value of the AnnotationType property in the Image Annotation Tool Button control.

When a control is deselected, the bitmap specified in the **PictureUp** property is drawn on the control.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 95 and 98

√ Imaging for Windows NT 4.0

Applies To Image Annotation control.

Usage *object*.**Value**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	The control is selected.
False (default)	The control is deselected.

Remarks This property is available only at run-time.

See Also AnnotationType property, DestImageControl property, PictureDown property, PictureUp property.

Zoom Property

Description Returns or sets the zoom factor for an image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

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

Data Type Single.

Remarks The Zoom value is expressed as a percentage. The valid range is from 2 to 6500 percent; the default value is 100 percent.

When invoked, the **FitTo** and **ZoomToSelection** methods update the value of the Zoom property.

A Zoom value can be specified before or after an image is displayed. If the image currently specified in the **Image** property is being displayed when the Zoom value is changed, the displayed image may change visually.

When scanning to the Image Edit control, the **Zoom** property of the Image Scan control should be set to the same value as the Zoom property of the Image Edit control.

When the **AutoRefresh** property is set to True, changing the Zoom property causes the displayed image to be refreshed automatically.

See Also AutoRefresh property, FitTo method, Image property, MagnifierStatus event, MagnifierZoom property, Zoom property (Image Scan), ZoomToSelection method.

Zoom Example — VB

The following example reduces the current zoom factor by one half, and then repaints the image display.

```
Private Sub cmdZoom_Click()
    ImgEdit1.Zoom = ImgEdit1.Zoom / 2
    ImgEdit1.Refresh
End Sub
```

Zoom Example — VC++

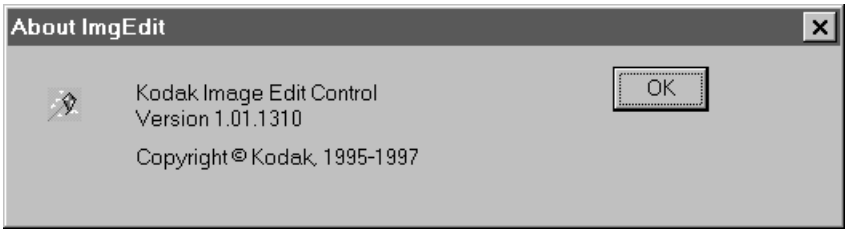
The following example reduces the current zoom factor by one half, and then repaints the image display.

```
void CImgEdit1Dlg::OnZoom()
{
    // This would reduce the current zoom factor by one half and then
```

```
    // repaint the image display.
    ImgEdit1.SetZoom(ImgEdit1.GetZoom() / 2);
    ImgEdit1.Refresh();
}
```

AboutBox Method

Description Displays an About dialog box (shown here) that contains the version number of the control.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit, Image Annotation, Image Admin, Image OCR, Image Scan, and Image Thumbnail controls.

Usage *object*.**AboutBox**

Arguments None.

Returns None.

AddAnnotationGroup Method

Description Adds a new annotation group to an image page.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**AddAnnotationGroup** *GroupName*

Arguments The AddAnnotationGroup method has the following argument:

Parameter	Data Type	Description
GroupName	String	The name of the new annotation group.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The annotation group is not actually created until the first annotation mark is placed on the image.

Invoking the AddAnnotationGroup method sets the **ImageModified** property to True.

See Also DeleteAnnotationGroup method, Display method, ImageModified property.

AddAnnotationGroup Example — VB

This example adds an annotation group, sets the annotation type, sets the background color of the annotation, and then draws the mark. (Annotation groups are not actually created until a mark is drawn in the group.)

```
Private Sub cmdAddGroup_Click()
    ImgEdit1.AddAnnotationGroup "Accounting Dept"
    ImgEdit1.AnnotationType = wiTextAttachment  '10
    ImgEdit1.AnnotationBackColor = vbYellow
    ImgEdit1.Draw 10, 10, 150, 100
    'The user must manually enter the text on the attachment
End Sub
```

AddAnnotationGroup Example — VC++

This example adds an annotation group, sets the annotation type, sets the background color of the annotation, and then draws the mark. (Annotation groups are not actually created until a mark is drawn in the group.)

```
void CFrmGroup::OnAddGroup()
{
    // Adds a group and draws a mark in the group. Annotation groups
    // are not actually created until a mark is drawn in the group.
    CString szCurGroup;
    szCurGroup.Format("Accounting Dept %i",rand());
    // Get the name of the group the user clicked on in the listbox
    m_GroupList.AddString(szCurGroup);
    if(pParentDlg)
    {
      pParentDlg->ImgEdit1.AddAnnotationGroup(szCurGroup);
      pParentDlg->ImgEdit1.SetAnnotationType(10); // wiTextAttachment // 10
      pParentDlg->ImgEdit1.SetAnnotationBackColor(0xFFFF); // vbYellow
      VARIANT vWidth; V_VT(&vWidth) = VT_I2; V_I2(&vWidth) = 150;
```

```
        VARIANT vHeight; V_VT(&vHeight) = VT_I2; V_I2(&vHeight) = 100;
        pParentDlg->ImgEdit1.Draw(10, 10, vWidth, vHeight);
      }
      // The user must manually enter the text on the attachment
}
```

AutoCrop Method

Description Automatically crops the displayed image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**AutoCrop** [*Option*]

Arguments The AutoCrop method has the following argument:

Parameter	Data Type	Setting	Description
Option	Long (enumerated)	0 (default)	Crops all solid black-and-white image data from outside edges.
		1	Crops all solid black image data from outside edges.
		2	Crops all solid white image data from outside edges.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

After invoking the AutoCrop method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the AutoCrop method sets the **ImageModified** property to True.

See Also Crop method, Display method, ImageModified property, Save method, SaveAs method, SavePage method.

AutoCrop Example — VB

This example shows how to crop an image automatically. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
Private Sub cmdCleanup_Click()
    'Perform automatic straighten on the image
```

```
    ImgEdit1.AutoDeskew
    'Remove speckles on the image using the default
    'threshold value for dot size
    ImgEdit1.Despeckle
    'If there is a lot of extraneous space around image
    'borders, cropping might reduce processing time
    ImgEdit1.AutoCrop
    'Show the OCR dialog box
    ImgOCR1.ShowOCR
End Sub
```

AutoCrop Example — VC++

This example shows how to crop an image automatically. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
void CImgEdit1Dlg::OnCleanup()
{
    // Here are some functions you might want to perform on an image prior
    // to OCR.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    // Perform automatic straighten on the image
    ImgEdit1.AutoDeskew();
    // Remove speckles on the image using the default threshold value for
    // dot size
    ImgEdit1.Despeckle(evt);
    // If there is a lot of extraneous space around image
    // borders, cropping might reduce processing time
    ImgEdit1.AutoCrop(evt);
    // Show the OCR dialog box
    ImgOcr1.ShowOcr();
}
```

AutoDeskew Method

Description Automatically deskews (straightens) the displayed image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**AutoDeskew**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

After invoking the AutoDeskew method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the AutoDeskew method sets the **ImageModified** property to True.

See Also Display method, ImageModified property, ManualDeskew method, Save method, SaveAs method, SavePage method.

AutoDeskew Example — VB

This example shows how to deskew an image automatically. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
Private Sub cmdCleanup_Click()
    'Perform automatic straighten on the image
    ImgEdit1.AutoDeskew
    'Remove speckles on the image using the default threshold value for dot
    'size
    ImgEdit1.Despeckle
    'If there is a lot of extraneous space around image
    'borders, cropping might reduce processing time
    ImgEdit1.AutoCrop
    'Show the OCR dialog box
    ImgOCR1.ShowOCR
End Sub
```

AutoDeskew Example — VC++

This example shows how to deskew an image automatically. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
void CImgEdit1Dlg::OnCleanup()
{
    // Here are some functions you might want to perform on an image prior
    // to OCR.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    // Perform automatic straighten on the image
    ImgEdit1.AutoDeskew();
    // Remove speckles on the image using the default threshold value for
    // dot size
    ImgEdit1.Despeckle(evt);
    // If there is a lot of extraneous space around image
    // borders, cropping might reduce processing time
    ImgEdit1.AutoCrop(evt);
    // Show the OCR dialog box
    ImgOcr1.ShowOcr();
}
```

BurnInAnnotations Method

Description Burns annotations onto an image, causing them to be permanently incorporated into the image. Use this method with caution because once annotations are burned-in, they can no longer be removed or modified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**BurnInAnnotations** *Option*,*MarkOption*[,*GroupName*]

Arguments The BurnInAnnotations method has the following arguments:

Parameter	Data Type	Setting	Description
Option	Integer (enumerated)	0	All annotations marks, invisible and visible, are saved to the base image
		1	All annotations that are visible are saved to the base image
		2	All annotations that are selected are saved to the base image
MarkOption (See Notes)	Integer (enumerated)	0	Changes annotations to black (on black-and-white images only)
		1	Changes annotations to white (on black-and-white images only)
		2	Default rendering — Does not change the colors of annotations for non-bilevel images (for example, color and gray-scale images)
GroupName	String	Name	The name of the group that contains the annotations (if specified, the Option parameter is ignored)

Notes Regarding the Use of the MarkOption Parameter

MarkOption Set to 0

For black-and-white page types, the BurnInAnnotations method with a MarkOption of 0 changes annotations to black.

MarkOption Set to 1

For black-and-white page types, the BurnInAnnotations method with a MarkOption of 1 changes annotations to white.

MarkOption Set to 2

For black-and-white page types, the BurnInAnnotations method with a MarkOption of 2 changes annotations to black, white, or black-and-white dither patterns based on the type of annotation mark, its color, and its transparency setting. Refer to the following list:

Annotation Type	MarkOption Burn-In Format
Attach-a-Note	Burned-in as a black or white rectangle based on the original background color, with text burned-in as black or white based on the original color of the font. Light colors, like yellow, are burned-in as white; dark colors, like red, are burned-in as black. The underlying image data is obscured.
Freehand Line, Hollow Rectangle, Straight Line, Text, Text From File, and Text Stamp Hyperlink	Burned-in as black or white based on the original color of the annotation. Light colors, like yellow, are burned-in as white; dark colors, like red, are burned-in as black. Linked annotations cease to function as hypertext jumps when they are burned in.
Filled Rectangle	Burned-in as black-and-white dithered rectangles. The underlying image data is obscured.
Highlighter	Burned-in as black-and-white transparent dithered rectangles. The underlying image data is visible.
Image Embedded and Image Reference	Burned-in as a black-and-white representation of the original image.
OCR Zones	You cannot burn-in OCR Zone annotations.

For all other page types, the BurnInAnnotations method with a MarkOption of 2 burns annotations onto the image using the displayed colors. Then it converts the image to a 24-bit color image (BGR24 or RGB24). Because 24-bit color images require a large amount of memory and disk space, use caution when selecting this option. To minimize the memory and disk space requirements of 24-bit color images, you can use the **SaveAs** or **SavePage** method to save the 24-bit color images as 8-bit Palettized images (recommended).

To save color annotations on a black-and-white image, use the **ConvertPageType** method to convert the image to a page type other than black-and-white. Then invoke the BurnInAnnotations method with a Mark Option of 2; the BurnInAnnotations method burns the annotations onto the image in color.

Returns None.

Remarks After invoking the BurnInAnnotations method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the combined image and annotation.

When a GroupName is specified, the Option parameter is ignored and all annotation marks in the specified GroupName are burned-in.

Keep in mind that if you burn-in a Hyperlink annotation, it ceases to function as a hypertext jump.

Invoking the BurnInAnnotations method sets the **ImageModified** property to True.

> **Note:** Hyperlink and OCR Zone annotations are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

See Also ConvertPageType method, Display method, ImageModified property, Save method, SaveAs mehtod, SavePage method.

BurnInAnnotations Example — VB

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
Private Sub cmdBurnIn_Click()
    Const ALL_MARKS = 0
    Const WINDOWS_DEFAULT = 2
    'If you wish to save color annotations on a black and white image you
    'must first convert the image to a non black and white type.
    ImgEdit1.ConvertPageType wiPageTypePal8, True
    'Save all marks in their original colors
    ImgEdit1.BurnInAnnotations ALL_MARKS, WINDOWS_DEFAULT
    'Color Burn in always converts to RGB24. To save disk space,
    'save the image as an 8 bit palettized image.
    ImgEdit1.SaveAs "C:\imgsave\annotate.tif", wiFileTypeTIFF,
     ➥  wiPageTypePal8
End Sub
```

BurnInAnnotations Example — VC++

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
void CImgEdit1Dlg::OnBurnin()
{
    // Converts an image, burns in annotation marks and saves it under a new
    // file name.
    #define ALL_MARKS  0
    #define WINDOWS_DEFAULT  2
    // If you wish to save color annotations on a black and white image you
    // must first convert the image to a non black and white type.
    VARIANT vRepaint; V_VT(&vRepaint) = VT_BOOL; V_BOOL(&vRepaint) = TRUE;
    VARIANT evt; V_VT(&evt) = VT_ERROR;  // set to default
    ImgEdit1.ConvertPageType(5,vRepaint);  // wiPageTypePal8, True
```

```
    // Save all marks in their original colors
    ImgEdit1.BurnInAnnotations(ALL_MARKS, WINDOWS_DEFAULT,evt);
    // Color Burn in always converts to RGB24. To save disk space, save the
    // image as an 8 bit palettized image.
    VARIANT vFileType; V_VT(&vFileType) = VT_I2;  // set FileType for saveas
    VARIANT vPageType; V_VT(&vPageType) = VT_I2;  // set to PageType for
                                                  // saveas
    V_I2(&vFileType) = 1;
    V_I2(&vPageType) = 5;
    // wiFileTypeTIFF, wiPageTypePal8
    ImgEdit1.SaveAs("C:\\imgsave\\annotate.tif",vFileType,vPageType,
     ➥   evt,evt,evt);
}
```

ClearDisplay Method

Description Clears a displayed image in the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ClearDisplay**

Arguments None.

Returns None.

Remarks An image must be displayed before the ClearDisplay method can be invoked.

The ClearDisplay method does not save any changes that may have been made to an existing image, nor does it change the value of the **Image** property. It does, however, trigger the **Close** event.

See Also Close event, Image property.

ClipboardCopy Method

Description Copies the selected or specified portion of an image or annotation to the Clipboard.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ClipboardCopy** [*Left*,*Top*,*Width*,*Height*]

Arguments The ClipboardCopy method has the following parameters:

Parameter	Data Type	Setting
Left	Long	The upper-left corner of the selection rectangle in pixel coordinates of the displayed image
Top	Long	The top of the selection rectangle in pixel coordinates of the displayed image
Width	Long	The width of the selection rectangle in pixels
Height	Long	The height of the selection rectangle in pixels

Returns None.

Remarks If the ClipboardCopy method is invoked without parameters after an end user draws a selection rectangle, the image data bounded by the selection rectangle is copied to the Clipboard.

If the ClipboardCopy method is invoked without parameters and without an end user drawing a selection rectangle, the ClipboardCopy method generates an error. Check the StatusCode property for the error.

To copy annotations to the Clipboard, the end user selects the desired annotation marks using the Select Annotations tool.

To copy image data to the Clipboard, the end user selects the desired area using the image selection rectangle, which is the default when no annotation tool is selected.

It is not possible to copy both image and annotation data to the Clipboard simultaneously.

See Also StatusCode property, ClipboardCut method, ClipboardPaste method, DrawSlectionRect method, SelectionRectangle property, SelectionRectDrawn event.

ClipboardCopy Example — VB

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
Private Sub cmdClipCopy_Click()
    'Global blnIsRectangle As Boolean
    'The value of blnIsRectangle is set in the SelectionRectDrawn event.
    If blnIsRectangle = False Then
        MsgBox "No rectangle drawn", vbExclamation
        Exit Sub
    End If
    'If no parameters specified will copy area bounded by rectangle.
    ImgEdit1.ClipboardCopy
    'Create 8 1/2 X 11 black and white 300 dpi image.
    ImgEdit1.DisplayBlankImage 2550, 3300, 300, 300, wiPageTypeBW
```

```
    'Reset blnIsRectangle.
    blnIsRectangle = False
    'Paste data to new image, call CompletePaste to commit the paste
    'operation.
    ImgEdit1.ClipboardPaste
    ImgEdit1.CompletePaste
End Sub
```

ClipboardCopy Example — VC++

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
void CImgEdit1Dlg::OnCopyPaste()
{
    // Copy the area bounded by user drawn selection rectangle to new page.
    // Global blnIsRectangle As Boolean. The value of blnIsRectangle is set
    // in the SelectionRectDrawn event.
    if(!blnIsRectangle)
    {
       AfxMessageBox ("No rectangle drawn", MB_ICONEXCLAMATION);
       return;
    }
    // If no parameters specified will copy area bounded by rectangle.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ClipboardCopy(evt,evt,evt,evt);
    // Create 8 1/2 X 11 black and white 300 dpi image.
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 300;
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vPageType) = 1; // wiPageTypeBW
    ImgEdit1.DisplayBlankImage(2550, 3300, vRes, vRes, vPageType);
    blnIsRectangle = FALSE;
    // Paste data to new image, call CompletePaste to commit the paste
    // operation.
    ImgEdit1.ClipboardPaste(evt,evt);
    ImgEdit1.CompletePaste();
}
```

ClipboardCut Method

Description Copies the selected or specified portion of an image or annotation to the Clipboard and then clears (cuts) the portion from the image or annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ClipboardCut** [*Left*,*Top*,*Width*,*Height*]

Arguments The ClipboardCut method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The upper-left corner of the selection rectangle in pixel coordinates of the displayed image.
Top	Long	The top of the selection rectangle in pixel coordinates of the displayed image.
Width	Long	The width of the selection rectangle in pixels.
Height	Long	The height of the selection rectangle in pixels.

Returns None.

Remarks If the ClipboardCut method is invoked without parameters after an end user draws a selection rectangle, the image data bounded by the selection rectangle is copied to the Clipboard and cleared from the image. If the ClipboardCut method is invoked without parameters and without an end user drawing a selection rectangle, the ClipboardCut method generates an error. Check the StatusCode property for the error.

To cut annotations to the Clipboard, the end user selects the desired annotation marks using the Select Annotations tool. To cut image data to the Clipboard, the end user selects the desired area using the image selection rectangle, which is the default when no annotation tool is selected.

It is not possible to cut both image and annotation data to the Clipboard simultaneously.

Invoking the ClipboardCut method sets the **ImageModified** property to True.

See Also ClipboardCopy method, StatusCode property, ClipboardPaste method, DrawSelectionRect method, ImageModified property, SelectionRectangle property, SelectionRectDrawn event.

ClipboardPaste Method

Description Pastes image or annotation data from the Clipboard.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ClipboardPaste** [*Left*,*Top*]

Arguments The ClipboardPaste method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The left point of the displayed image in pixels.
Top	Long	The top point of the displayed image in pixels.

Returns None.

Remarks To determine if data is available for pasting from the Clipboard, invoke the **IsClipboardDataAvailable** method.

If the Left and Top parameters are not entered, the data is pasted at the left-top point of the displayed image.

Invoking the ClipboardPaste method sets the **ImageModified** property to True.

See Also ClipboardCopy method, ClipboardCut method, CompletePaste method, ImageModified property, IsClipboardDataAvailable method, PasteClip event, PasteCompleted event.

ClipboardPaste Example — VB

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
Private Sub cmdClipCopy_Click()
    'Global blnIsRectangle As Boolean. The value of blnIsRectangle is set in
    'the SelectionRectDrawn event.
    If blnIsRectangle = False Then
        MsgBox "No rectangle drawn", vbExclamation
        Exit Sub
    End If
    'If no parameters specified will copy area bounded by rectangle.
    ImgEdit1.ClipboardCopy
    'Create 8 1/2 X 11 black and white 300 dpi image.
    ImgEdit1.DisplayBlankImage 2550, 3300, 300, 300, wiPageTypeBW
    'Reset blnIsRectangle.
    blnIsRectangle = False
    'Paste data to new image, call CompletePaste to commit the paste
    'operation.
    ImgEdit1.ClipboardPaste
    ImgEdit1.CompletePaste
End Sub
```

ClipboardPaste Example — VC++

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
void CImgEdit1Dlg::OnCopyPaste()
{
    // This would copy the area bounded by a user drawn selection rectangle
    // to a new page.
    // Global blnIsRectangle As Boolean
    // The value of blnIsRectangle is set in the SelectionRectDrawn event.
```

```
    if(!blnIsRectangle)
    {
      AfxMessageBox ("No rectangle drawn", MB_ICONEXCLAMATION);
      return;
    }
    // If no parameters specified will copy area bounded by rectangle.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ClipboardCopy(evt,evt,evt,evt);
    // Create 8 1/2 X 11 black and white 300 dpi image.
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 300;
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vPageType) = 1; // wiPageTypeBW
    ImgEdit1.DisplayBlankImage(2550, 3300, vRes, vRes, vPageType);
    // Reset blnIsRectangle.
    blnIsRectangle = FALSE;
    // Paste data to new image, call CompletePaste to commit the paste
    // operation.
    ImgEdit1.ClipboardPaste(evt,evt);
    ImgEdit1.CompletePaste();
}
```

CompletePaste Method

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

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**CompletePaste**

Returns None.

Remarks The **ClipboardPaste** method pastes image or annotation data from the Clipboard into the control. After the image or annotation data appears in the control, it can be moved at will and is therefore not part of the original image. To make the pasted image or annotation data part of the original image, the Clipboard paste operation must be completed. When invoked, the CompletePaste method completes a Clipboard paste operation. A Clipboard paste operation can also be completed by an end user clicking on the form, outside of the pasted data.

See Also ClipboardPaste method, PasteClip event, PasteCompleted event.

CompletePaste Example — VB

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
Private Sub cmdClipCopy_Click()
    'Global blnIsRectangle As Boolean
    'The value of blnIsRectangle is set in the SelectionRectDrawn event.
    If blnIsRectangle = False Then
        MsgBox "No rectangle drawn", vbExclamation
        Exit Sub
    End If
    'If no parameters specified will copy area bounded by rectangle.
    ImgEdit1.ClipboardCopy
    'Create 8 1/2 X 11 black and white 300 dpi image.
    ImgEdit1.DisplayBlankImage 2550, 3300, 300, 300, wiPageTypeBW
    'Reset blnIsRectangle.
    blnIsRectangle = False
    'Paste data to new image, call CompletePaste to commit the paste
    'operation.
    ImgEdit1.ClipboardPaste
    ImgEdit1.CompletePaste
End Sub
```

CompletePaste Example — VC++

This example copies the area bound by a user-drawn selection rectangle to a new page.

```
void CImgEdit1Dlg::OnCopyPaste()
{
    // This would copy the area bounded by a user drawn selection rectangle
    // to a new page. Global blnIsRectangle As Boolean
    // The value of blnIsRectangle is set in the SelectionRectDrawn event.
    if(!blnIsRectangle)
    {
      AfxMessageBox ("No rectangle drawn", MB_ICONEXCLAMATION);
      return;
    }
    // If no parameters specified will copy area bounded by rectangle.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ClipboardCopy(evt,evt,evt,evt);
    // Create 8 1/2 X 11 black and white 300 dpi image.
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 300;
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vPageType) = 1; // wiPageTypeBW
    ImgEdit1.DisplayBlankImage(2550, 3300, vRes, vRes, vPageType);
    // Reset blnIsRectangle.
    blnIsRectangle = FALSE;
    // Paste data to new image, call CompletePaste to commit the paste
    // operation.
    ImgEdit1.ClipboardPaste(evt,evt);
    ImgEdit1.CompletePaste();
}
```

ConvertPageType Method

Description Converts a displayed image to the page type specified and optionally repaints the image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ConvertPageType** *PageType*[,*Repaint*]

Arguments The ConvertPageType method has the following arguments:

Parameter	Data Type	Setting
PageType	Integer (enumerated)	The desired PageType setting (see page 528).
Repaint	Boolean	True — Image is repainted (default). False — Image is not repainted.

Returns None.

Remarks An image must be displayed before the ConvertPageType method can be invoked. If the Repaint parameter is not specified, True is entered as a default (the image is repainted). Invoking this method sets the **ImageModified** property to True.

To save the converted image, the **Save**, **SaveAs**, or **SavePage** method must be invoked.

An image cannot be converted to a PageType of Palettized4 unless it presently has a 4-bit palette.

Invoking the ConvertPageType method sets the **ImageModified** property to True.

See Also ImageModified property, PageType property, Save method, SaveAs method, SavePage method.

ConvertPageType Example — VB

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
Private Sub cmdBurnIn_Click()
    Const ALL_MARKS = 0
    Const WINDOWS_DEFAULT = 2
    'If you wish to save color annotations on a black and white image you
    'must first convert the image to a non black and white type.
    ImgEdit1.ConvertPageType wiPageTypePal8, True
    'Save all marks in their original colors.
    ImgEdit1.BurnInAnnotations ALL_MARKS, WINDOWS_DEFAULT
```

```
    'Color Burn in always converts to RGB24. To save disk space, save the
    'image as an 8 bit palettized image
    ImgEdit1.SaveAs "C:\imgsave\annotate.tif", wiFileTypeTIFF,
    ➥  wiPageTypePal8
End Sub
```

ConvertPageType Example — VC++

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
void CImgEdit1Dlg::OnBurnin()
{
    // Converts an image, burns in annotation marks and saves it under a new
    // file name.
    #define ALL_MARKS  0
    #define WINDOWS_DEFAULT  2
    // To save color annotations on a black and white image you must first
    // convert the image to a non black and white type.
    VARIANT vRepaint; V_VT(&vRepaint) = VT_BOOL; V_BOOL(&vRepaint) = TRUE;
    VARIANT evt; V_VT(&evt) = VT_ERROR;  // set to default
    ImgEdit1.ConvertPageType(5,vRepaint);  // wiPageTypePal8, True
    // Save all marks in their original colors.
    ImgEdit1.BurnInAnnotations(ALL_MARKS, WINDOWS_DEFAULT,evt);
    // Color Burn in always converts to RGB24. To save disk space, save the
     // image as an 8 bit palettized image.
    VARIANT vFileType; V_VT(&vFileType) = VT_I2; // set FileType for saveas
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vFileType) = 1;
    V_I2(&vPageType) = 5;
    // wiFileTypeTIFF, wiPageTypePal8
    ImgEdit1.SaveAs("C:\\imgsave\\annotate.tif",vFileType,
     ➥  vPageType,evt,evt,evt);
}
```

Crop Method

Description Crops the displayed image to the rectangle specified. The current selection rectangle is used if no rectangle is specified.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Crop** [*Left*,*Top*,*Width*,*Height*]

Arguments The Crop method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The upper-left corner of the rectangle in pixel coordinates of the displayed image.
Top	Long	The top of the rectangle in pixel coordinates of the displayed image.
Width	Long	The width of the rectangle in pixels.
Height	Long	The height of the rectangle in pixels.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

If the Crop method is invoked without parameters after an end user draws a selection rectangle, the image data outside of the selection rectangle is cropped.

The Crop method generates an error if it is invoked without parameters and without an end user drawing a selection rectangle. Check the StatusCode property for the error.

After invoking the Crop method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the Crop method sets the **ImageModified** property to True.

See Also StatusCode property, AutoCrop method, Display method, DrawSelectionRect method, ImageModified property, Save method, SaveAs method, SavePage method, SelectionRectangle property, SelectionRectDrawn event.

DeleteAnnotationGroup Method

Description Deletes an annotation group and its annotation marks, and then redisplays the image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DeleteAnnotationGroup** *GroupName*

Arguments The DeleteAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
GroupName	String	The name of the annotation group to delete.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

To save the image without the annotation marks, the **Save**, **SaveAs**, or **SavePage** method must be invoked.

Invoking the DeleteAnnotationGroup method sets the **ImageModified** property to True.

See Also AddAnnotationGroup method, DeleteSelectedAnnotations method, Display method, ImageModified property, Save method, SaveAs method, SavePage method.

DeleteAnnotationGroup Example — VB

This example lets the user select an annotation group from a list and delete the group. See the example for the **GetAnnotationGroup** method for instructions on how to list annotation groups.

```
Private Sub cmdDeleteGroup_Click()
    Dim strCurGroup As String
    'Get the name of the group the user clicked on in the listbox.
    strCurGroup = AnnoGroups.List(AnnoGroups.ListIndex)
    Form1.ImgEdit1.DeleteAnnotationGroup (strCurGroup)
End Sub
```

DeleteAnnotationGroup Example — VC++

This example lets the user select an annotation group from a list and delete the group. See the example for the **GetAnnotationGroup** method for instructions on how to list annotation groups.

```
void CFrmGroup::OnDeleteGroup()
{
    // This would allow a user to select a Group from a list and delete it.
    // See example for GetAnnotationGroup method for how to list
    // annotation groups.
    CString szCurGroup;
    // Get the name of the group the user clicked on in the listbox.
    m_GroupList.GetText(m_GroupList.GetCurSel(),szCurGroup);
    // Remove the group name.
    m_GroupList.DeleteString(m_GroupList.GetCurSel());
    if(pParentDlg)
    {
      pParentDlg->ImgEdit1.DeleteAnnotationGroup (szCurGroup);
    }
}
```

DeleteImageData Method

Description Deletes a selected or specified portion of an image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DeleteImageData** [*Left*,*Top*,*Width*,*Height*]

Arguments The DeleteImageData method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The upper-left corner of the selection rectangle in pixel coordinates of the displayed image.
Top	Long	The top of the selection rectangle in pixel coordinates of the displayed image.
Width	Long	The width of the selection rectangle in pixels.
Height	Long	The height of the selection rectangle in pixels.

Returns None.

Remarks If the DeleteImageData method is invoked without parameters after an end user draws a selection rectangle, the area bounded by the selection rectangle is deleted. This method does not delete annotation data.

If the DeleteImageData method is invoked without parameters and without an end user drawing a selection rectangle, the DeleteImageData method generates an error. Check the StatusCode property for the error.

After invoking the DeleteImageData method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the DeleteImageData method sets the **ImageModified** property to True.

See Also StatusCode property, DeleteSelectedAnnotations method, DrawSelectionRect method, ImageModified property, Save method, SaveAs method, SavePage method, SelectionRectangle property, SelectionRectDrawn event.

DeleteSelectedAnnotations Method

Description Deletes selected annotations.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DeleteSelectedAnnotations**

Arguments None.

Returns None.

Remarks To be deleted, annotations must first be selected using the Select Annotations tool (SelectTool method), or the **SelectAnnotationGroup** method. If no annotations are selected, the DeleteSelectedAnnotations method generates an error. Check the StatusCode property for the error.

After invoking the DeleteSelectedAnnotations method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Note that annotations can also be deleted by invoking the **DeleteAnnotationGroup** method.

Invoking the DeleteSelectedAnnotations method sets the **ImageModified** property to True.

See Also StatusCode property, DeleteAnnotationGroup method, ImageModified property, Save method, SaveAs method, SavePage method, SelectAnnotationGroup method.

Despeckle Method

Description Despeckles (removes black pixels from) the displayed image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Despeckle** [*Pattern*]

Arguments The Despeckle method has the following argument:

Parameter	Data Type	Setting	Description
Pattern	Long (enumerated)	0 (default)	Removes dots using a single-pixel pattern.
		1	Removes dots smaller than average.
		2	Removes dots of average size, those typically encountered and safely removed.
		3	Removes dots slightly larger than average.
		4	Removes most dots.
		5	Remove very large dots.

Returns None.

Remarks This method despeckles black-and-white images only. Use caution when setting the Pattern parameter so that the despeckling process does not negatively alter the image.

The **Display** method must be invoked prior to calling this method.

After invoking the Despeckle method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the Despeckle method sets the **ImageModified** property to True.

See Also Display method, ImageModified property, Save method, SaveAs method, SavePage method.

Despeckle Example — VB

This example shows how to despeckle an image. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
Private Sub cmdCleanup_Click()
    'Perform automatic straighten on the image.
    ImgEdit1.AutoDeskew
    'Remove speckles on the image using the default threshold value for
    'dot size.
    ImgEdit1.Despeckle
    'If there is a lot of extraneous space around image
    'borders, cropping might reduce processing time.
    ImgEdit1.AutoCrop
    'Show the OCR dialog box.
    ImgOCR1.ShowOCR
End Sub
```

Despeckle Example — VC++

This example shows how to despeckle an image. (It also shows how to invoke other functions that can prepare an image for OCR processing.)

```
void CImgEdit1Dlg::OnCleanup()
{
    // Here are some functions you might want to perform on an image
    // prior to OCR.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    // Perform automatic straighten on the image.
    ImgEdit1.AutoDeskew();
    // Remove speckles on the image using the default threshold value
    // for dot size.
    ImgEdit1.Despeckle(evt);
    // If there is a lot of extraneous space around image
    // borders, cropping might reduce processing time.
    ImgEdit1.AutoCrop(evt);
    // Show the OCR dialog box.
    ImgOcr1.ShowOcr();
}
```

Display Method

Description Displays the image specified in the **Image** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Display**

Arguments None.

Returns None.

Remarks The Display method triggers a **Load** event before the actual image is displayed. End users can then hide or show any of the annotation groups before they are actually displayed, or set any other display-oriented properties (for example, Zoom, ScrollPosition, etc.). If an image is already displayed, this method triggers a **Close** event.

See Also Close event, Image property, ImageDisplayed property, Load event, Page property.

Display Example — VB

This example displays a user-selected image file in an Image Edit control.

```
Private Sub cmdDisplayImage_Click()
    'Use the ImgAdmin Open dialog box to display a file in the Image
    'Edit and Thumbnail controls.
    ImgAdmin1.ShowFileDialog OpenDlg
    'See the sample code for handling the Cancel key in the
    'ShowFileDialog method.
    'Set the image properties in the Image Edit and Thumbnail controls
    'to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Set the page to 1 in case a page other than 1 was shown for an image
    'displayed prior to this one
    ImgEdit1.Page = 1
    ImgEdit1.Display
    'Set the focus to the image window so that keystrokes such
    'as scroll keys are sent to the image window.
    ImgEdit1.SetFocus
End Sub
```

Display Example — VC++

This example displays a user-selected image file in an Image Edit control.

```
void CImgEdit1Dlg::OnDisplay()
{
    // Use the ImgAdmin Open dialog box to display a file in the
    // Image Edit and Thumbnail controls.
     VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg
    // See the sample code for handling the Cancel key in the
    // ShowFileDialog method.
    // Set the image properties in the Image Edit and Thumbnail controls
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage(ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage(ImgAdmin1.GetImage());
    // Set the page to 1 in case a page other than 1 was shown
    // for an image displayed prior to this one.
    ImgEdit1.SetPage(1);
    ImgEdit1.Display();
    // Set the focus to the image window so that keystrokes such
    // as scroll keys are sent to the image window.
    ImgEdit1.SetFocus();
}
```

DisplayBlankImage Method

Description Displays a blank image according to the dimensions specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DisplayBlankImage** *ImageWidth*, *ImageHeight* [, *ResolutionX*, *ResolutionY*, *PageType*]

Arguments The DisplayBlankImage method has the following arguments:

Parameter	Data Type	Setting
ImageWidth	Long	The width of the image in pixels at full-scale (100 percent).
ImageHeight	Long	The height of the image in pixels at full-scale (100 percent).
ResolutionX	Long	The horizontal resolution of the image in dots per inch.
ResolutionY	Long	The vertical resolution of the image in dots per inch.
PageType	Integer (enumerated)	The desired PageType setting (see page 528).

Returns None.

Remarks When the DisplayBlankImage method is invoked with an image already displayed, it triggers a **Close** event. The displayed image file is closed without being saved, and the new image data is displayed. The value of the **Image** property remains unchanged.

If the ResolutionX parameter or the ResolutionY parameter is not entered, a value of 200 is used as a default. If the PageType parameter is not entered, the Black-and-White page type is used as a default.

See Also Close event.

DisplayBlankImage Example — VB

This example creates a new blank image and then places a company logo at the top of the new page.

```
Private Sub cmdDrawLogo_Click()
    'Create a new 8 1/2 X 11" black and white image at 100 dpi.
    ImgEdit1.DisplayBlankImage 850, 1100, 100, 100, wiPageTypeBW
    ImgEdit1.AnnotationImage = "D:\image2\corplogo.bmp"
    'Logo would obscure any text under it if Opaque.
    ImgEdit1.AnnotationFillStyle = wiOpaque   '1
```

```
    'If image is embedded rather than referenced, the source
    'image need not be present when the image is displayed.
    ImgEdit1.AnnotationType = wiImageEmbedded   '5
    'Image mark only requires left and top parameters to draw.
    'Start not quite halfway across page, 25 pixels from top (1/4").
    ImgEdit1.Draw 350, 25
End Sub
```

DisplayBlankImage Example — VC++

This example creates a new blank image and then places a company logo at the top of the new page.

```
void CImgEdit1Dlg::OnImageStamp()
{
    // Create a new 8 1/2 X 11" black and white image at 100 dpi.
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vPageType) = 1; // wiPageTypeBW
    VARIANT vRes; V_VT(&vRes) = VT_I4; V_I4(&vRes) = 100;
    ImgEdit1.DisplayBlankImage (850, 1100, vRes, vRes, vPageType);
    ImgEdit1.SetAnnotationImage("D:\\image2\\corplogo.bmp");
    // Logo would obscure any text under it if Opaque.
    ImgEdit1.SetAnnotationFillStyle(1);  // wiOpaque   // 1
    // If image is embedded rather than referenced, the source
    // image need not be present when the image is displayed.
    ImgEdit1.SetAnnotationType(5);  // wiImageEmbedded   // 5
    // Image mark only requires left and top parameters to draw.
    // Start not quite halfway across page, 25 pixels from top (1/4").
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(350, 25, evt, evt);
}
```

Draw Method

Description Draws the annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage *object*.**Draw** *Left,Top*[,*Width,Height*]

Arguments The Draw method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The starting horizontal coordinate of the annotation, in image pixels (see page 544).
Top	Long	The starting vertical coordinate of the annotation, in image pixels.
Width	Long	The width of the annotation in image pixels.
Height	Long	The height of the annotation in image pixels.

Returns None.

Remarks The Draw method draws the annotation specified by the **AnnotationType** property, standard Annotation Tool Palette, or **SelectTool** method on the displayed image. The method draws the annotation in accordance with the coordinates specified in the left, top, width, and height parameters.

The width and height parameters must be specified when using the following annotation types:

- Attach-a-Note
- Filled Rectangle
- Freehand Line
- Hollow Rectangle
- Select Annotations (optional; see the following Note)
- Straight Line
- Text

Invoking the Draw method sets the **ImageModifed** property of the Image Edit control to True.

Note: Width and height parameter entry is optional when using the Select Annotations annotation type. If only the left and top parameters are specified, selection can be made by point. If all of the parameters are specified, selection can be made using the Select Annotations rectangle.

See Also AnnotationType property, HyperlinkGoToDoc event, HyperlinkGoToPage event, ImageModified property, MarkEnd event, SelectTool method.

Draw Example — VB

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
Private Sub cmdSelectTool_Click()
    'If the annotation tool palette is utilized within an application,
    'SelectTool should be used rather than setting the AnnotationType
    'property for drawing marks programatically.
    ImgEdit1.SelectTool 9 'Text from file
    ImgEdit1.AnnotationTextFile = "C:\text\disclaim.txt"
    ImgEdit1.Draw 20, 20
End Sub
```

Draw Example — VC++

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
void CImgEdit1Dlg::OnSelectTool()
{
    // Uses the SelectTool method to select the Text from file annotation
    // type. Overlays text from a text file onto image.
    // If the annotation tool palette is utilized within an application,
    // SelectTool should be used rather than setting the AnnotationType
    // property for drawing marks programatically.
    ImgEdit1.SelectTool(9); // Text from file
    ImgEdit1.SetAnnotationTextFile("C:\\text\\disclaim.txt");
    VARIANT evt; V_VT(&evt) = VT_ERROR;// set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

DrawSelectionRect Method

Description Draws a selection rectangle on an image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**DrawSelectionRect** *Left*,*Top*,*Width*,*Height*

Arguments The DrawSelectionRect method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The beginning horizontal position of the rectangle relative to the image window in pixels.
Top	Long	The beginning vertical position of the rectangle relative to the image window in pixels.
Width	Long	The width of the selection rectangle in pixels.
Height	Long	The height of the selection rectangle in pixels.

Returns None.

Remarks None.

See Also ClipboardCopy method, ClipboardCut method, DeleteImageData method, SelectionRectangle property, SelectionRectDrawn event, ZoomToSelection event.

DrawSelectionRect Example — VB

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
Private Sub cmdZoomSelect_Click()
    'Enable rectangle drawing.
    ImgEdit1.SelectionRectangle = True
    'Draw rectangle.
    ImgEdit1.DrawSelectionRect 150, 200, 400, 400
    ImgEdit1.ZoomToSelection
End Sub
```

DrawSelectionRect Example — VC++

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
void CImgEdit1Dlg::OnZoomto()
{
    // This would draw a rectangle on an image and would zoom in on the area
    // bound by the rectangle.
    // Enable rectangle drawing.
    ImgEdit1.SetSelectionRectangle(TRUE);
    // Draw rectangle.
    ImgEdit1.DrawSelectionRect(150, 200, 400, 400);
    ImgEdit1.ZoomToSelection();
}
```

EditSelectedAnnotationText Method

Description Displays a dialog box (shown here) containing the selected text annotation. The dialog box enables end users to modify the text.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**EditSelectedAnnotationText** *Left*,*Top*

Arguments The EditSelectedAnnotationText method has the following arguments:

Parameter	Data Type	Setting
Left	Long	The beginning horizontal position of the mark to be edited.
Top	Long	The beginning vertical position of the mark to be edited.

Returns None.

Remarks The annotation must be selected for this method to function. Use the Select Annotations tool (SelectTool method) or the SelectAnnotationGroup method to select annotations.

The left and top coordinates of the annotation to be edited can be obtained from the **MarkSelect** event.

The **EditingTextAnnotation** event fires whenever the Image Edit control enters or exits Text Edit mode.

After invoking the EditSelectedAnnotationText method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered annotation.

Invoking the EditSelectedAnnotationText method sets the **ImageModified** property to True.

This method functions with Text and Hyperlink annotation types.

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

See Also AnnotationType property, EditingTextAnnotation event, ExecuteTextEditCommand method, ImageModified property, MarkSelect event, Save method, SaveAs method, SavePage method, SelectAnnotationGroup method.

EditSelectedAnnotationText Example — VB

This example uses the MarkSelect event and the EditSelectedAnnotationText method to allow the user to edit the text of an existing, selected annotation mark.

```
Private Sub ImgEdit1_MarkSelect(ByVal Button As Integer, ByVal Shift As
➥  Integer, ByVal Left As Long, ByVal Top As Long, ByVal Width As Long,
➥  ByVal Height As Long, ByVal MarkType As Integer, ByVal GroupName As
➥  String)
    'Dimension MarkLeft and MarkTop as global variables so these
    'values can be passed back to the EditSelectedAnnotationText method.
    MarkLeft = Left
    MarkTop = Top
End Sub
Private Sub cmdEditText_Click()
    'This would allow the user to edit the text of an existing selected
    'annotation mark.
    On Error GoTo HandleIt
    'Coordinates of the mark are obtained from the MarkSelect event
    '(see code above).
    ImgEdit1.EditSelectedAnnotationText MarkLeft, MarkTop
    Exit Sub
HandleIt:
     'Handle error if no mark is selected or the wrong type of mark is
     'selected.
      MsgBox Err.Description
End Sub
```

EditSelectedAnnotationText Example — VC++

This example uses the MarkSelect event and the EditSelectedAnnotationText method to allow the user to edit the text of an existing, selected annotation mark.

```
void CImgEdit2Dlg::OnAnnotext()
{
    // This would allow the user to edit the text of an existing selected
    // annotation mark.
    TRY
    {
      // Coordinates of the mark are obtained from the MarkSelect event
      // (see code below).
      ImgEdit1.EditSelectedAnnotationText(m_MarkLeft, m_MarkTop);
    }
    CATCH (COleDispatchException, e)
    {
      // Handle error if no mark is selected or the wrong type of mark is
      // selected.
       AfxMessageBox(e->m_strDescription);
```

```
        //Err.Description
    }
    END_CATCH
}
void CImgEdit2Dlg::OnMarkSelect(short Button, short Shift, long Left,
➥ long Top, long Width, long Height, short MarkType, LPCTSTR GroupName)
{
    m_MarkLeft = Left;
    m_MarkTop = Top;
}
```

ExecuteTextEditCommand Method

Description Executes commands on the Text Edit dialog box when the Image Edit control is in Text Edit mode, as indicated by the **EditingTextAnnotation** event.

When the Image Edit control is in Text Edit mode, it displays the Text Edit dialog box to let end users create or modify a Text, Attach-a-Note, or Hyperlink annotation.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ExecuteTextEditCommand** *EditCommand*

Arguments The ExecuteTextEditCommand method has the following argument:

Parameter	Data Type	Constant	Value	Command — Description
EditCommand	Integer	**wiCutEditText**	0	Cut — Cuts any selected text to the clipboard.
		wiCopyEditText	1	Copy — Copies any selected text to the clipboard.
		wiPasteEditText	2	Paste — Pastes text from the clipboard.
		wiUndoEditText	3	Undo/Redo — Undoes or Redoes the last textual operation.
		wiSelectAllEditText	4	Select All — Selects All text.
		wiDeleteEditText	5	Delete/Clear — Deletes or Clears any selected text.
		wiCanUndoEditText	6	Can Undo — Checks to see if there is anything to be undone.
		wiIsEditTextSelected	7	Is Text Selected — Checks to see if there is any text selected.
		wiFinishEditText	8	Finish — Saves any text changes, closes the text editing session.
		wiCancelEditText	9	Cancel — Discards any text changes and closes the text editing session.
		wiIsModifiedEditText	10	Is Modified — Checks to see if anything was changed.

Returns Long.

Command	Value	Description
Cut	0	—
Copy	0	—
Paste	0	—
Undo/Redo	0	Failed
	Not 0	Success
Select All	0	—
Delete/Clear	0	—
Can Undo	0	Cannot undo
	1	Can undo
Is Text Selected	0	No text selected
	1	Text selected
Finish	0	—
Cancel	0	—
Is Modified	0	Not modified
	Not 0	Modified

Remarks An image must be displayed and the Image Edit control must be in Text Edit mode for the commands of this method to function. The Image Edit control is in Text Edit mode when:

- An end user double-clicks on a selected Text, Attach-a-Note, or Hyperlink annotation
- The **EditSelectedAnnotationText** method is invoked with a Text, Attach-a-Note, or Hyperlink annotation selected.
- A new Text, Attach-a-Note, or Hyperlink annotation is created.

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

See Also EditingTextAnnotation event, EditSelectedAnnotationText method, SelectAnnotationGroup method.

FitTo Method

Description Scales the image relative to the image window.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**FitTo** *Option*[,*Repaint*]

Arguments The FitTo method has the following arguments:

Parameter	Data Type	Constant	Setting	Description
Option	Integer	**BEST_FIT**	0	Best fit — Fits the image to the size of the window.
		FIT_TO_WIDTH	1	Fit to width — Fits the image to the width of the window.
		FIT_TO_HEIGHT	2	Fit to height — Fits the image to the height of the window.
		INCH_TO_INCH	3	Inch to inch — Maintains the relative size of the original image regardless of the resolution of the image.
Repaint	Boolean		True False	Image is repainted (default) Image is not repainted.

Returns None.

Remarks The Inch to inch setting calculates the size of the image dependent upon the dots per inch reported by the display driver.

When invoked, the FitTo method updates the **Zoom** property with the new zoom factor.

The FitTo method can be invoked before or after an image is displayed. If an image is displayed when the method is invoked, and the Repaint parameter is set to True, the image is redisplayed using the Option setting specified.

See Also Zoom property (ImageEdit), ZoomToSelection method.

FitTo Example — VB

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
Private Sub cmdWidthHeight_Click()
    ImgEdit1.Image = "D:\image2\ocr.tif"
    ImgEdit1.FitTo BEST_FIT
    ImgEdit1.Display
    Load frmInfo
    'Display the dimensions of the image page.
    frmInfo.lblImgHeight.Caption = ImgEdit1.ImageHeight
    frmInfo.lblImgWidth.Caption = ImgEdit1.ImageWidth
    'Display the X and Y resolution of the image page.
    frmInfo.lblImgResX.Caption = ImgEdit1.ImageResolutionX
    frmInfo.lblImgResY.Caption = ImgEdit1.ImageResolutionY
    'Display Image Scale Height.
    frmInfo.lblImgScaleHeight.Caption = ImgEdit1.ImageScaleHeight
    frmInfo.lblImgScaleWidth.Caption = ImgEdit1.ImageScaleWidth
    frmInfo.Show 1
End Sub
```

FitTo Example — VC++

This example displays an image and scales it so it fits into the window. Then it displays the height, width, resolution, scaled height, and scaled width of the image to illustrate the difference between ImageHeight/Width and ImageScaleHeight/Width.

```
void CImgEdit1Dlg::OnWidthHeight()
{
    // Displays an image that is fit to window and then displays height,
    // width, resolution, scaled height and scaled width. This illustrates
    // the difference between ImageHeight/Width and ImageScaleHeight/Width.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.FitTo(0,evt); //BEST_FIT
    ImgEdit1.Display();
    CFrmInfo frmInfo;
    // Display the dimensions of the image page.
    CString szWindowText;
    long lInfo;
    lInfo = ImgEdit1.GetImageHeight();
    frmInfo.lblImgHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageWidth();
    frmInfo.lblImgWidth.Format("%i",lInfo);
    // Display the X and Y resolution of the image page.
    lInfo = ImgEdit1.GetImageResolutionX();
    frmInfo.lblImgResX.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageResolutionY();
    frmInfo.lblImgResY.Format("%i",lInfo);
    // Display Image Scale Height.
    lInfo = ImgEdit1.GetImageScaleHeight();
```

```
    frmInfo.lblImgScaleHeight.Format("%i",lInfo);
    lInfo = ImgEdit1.GetImageScaleWidth();
    frmInfo.lblImgScaleWidth.Format("%i",lInfo);
    frmInfo.DoModal();
}
```

Flip Method

Description Rotates a displayed image by 180 degrees.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Flip**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

Invoking the Flip method modifies the displayed image. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the Flip method sets the **ImageModified** property to True.

See Also Display method, ImageModified property, Rotate method, RotateAll method, RotateLeft method, RotateRight method, Save method, SaveAs method, SavePage method.

Flip Example — VB

This example shows several ways to orient an image, and then redisplay it.

```
Private Sub cmdRotate_Click()
    ImgEdit1.Rotate True 'rotate all pages
    ImgEdit1.Rotate False 'rotate current page
    'Autorefresh must be set to true or must call Refresh after these
    'methods.
    ImgEdit1.AutoRefresh = True
    ImgEdit1.RotateLeft 'defaults to 90 degrees
    ImgEdit1.RotateRight 45 'optionally can specify degrees
    ImgEdit1.Flip '180 degrees
    'You must save the image to maintain orientation.
End Sub
```

Flip Example — VC++

This example shows several ways to orient an image, and then redisplay it.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically. Rotate all
    // writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

GetAnnotationGroup Method

Description Returns the name of the annotation group based on the index specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetAnnotationGroup**(*Index*)

Arguments The GetAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
Index	Integer	The index value of the annotation group.

Returns String.

Remarks The **Display** method must be invoked prior to calling this method.

Index is a 0-based value, where 0 is the first annotation group in the array, 1 is the second, 2 is the third, and so on. The upper bound value of the index can be obtained from the **AnnotationGroupCount** property.

The GetAnnotationGroup method is valid only at run-time.

See Also AnnotationGroupCount property, Display method, GetCurrentAnnotationGroup method, HideAnnotationGroup method, SelectAnnotationGroup method, SetCurrentAnnotationGroup method, ShowAnnotationGroup method.

GetAnnotationGroup Example — VB

This example uses the AnnotationGroupCount property to determine the number of groups on an image. Then it uses the GetAnnotationGroup method to get the group names.

```
Private Sub cmdListGroups_Click()
    Dim intNumGroups As Integer
    Dim strGroupName As String
    Dim intCount As Integer
    'Find out how many groups are on the page.
    intNumGroups = ImgEdit1.AnnotationGroupCount
    Load frmGroupList
    'Since index values of GetAnnotationGroup start with
    '0 have to loop one less than the number of groups.
    For intCount = 0 To intNumGroups - 1
        strGroupName = ImgEdit1.GetAnnotationGroup(intCount)
        'Write names of groups to a listbox.
        frmGroupList.AnnoGroups.AddItem (strGroupName)
    Next intCount
    frmGroupList.Show
End Sub
```

GetAnnotationGroup Example — VC++

This example uses the AnnotationGroupCount property to determine the number of groups on an image. Then it uses the GetAnnotationGroup method to get the group names.

```
BOOL CFrmGroup::OnInitDialog()
{
    CDialog::OnInitDialog();
    // This routine uses AnnotationGroupCount to determine the number of
    // groups on an image and then gets the group names using the
    // GetAnnotationGroup method.
    int iNumGroups;
    CString szGroupName;
    int iCount;
    // Find out how many groups are on the page.
    iNumGroups = pParentDlg->ImgEdit1.GetAnnotationGroupCount();
    // Since index values of GetAnnotationGroup start with
    // 0 have to loop one less than the number of groups.
    for (iCount = 0;iCount < iNumGroups ; iCount++)
    {
        szGroupName = pParentDlg->ImgEdit1.GetAnnotationGroup(iCount);
        // Write names of groups to a listbox.
        m_GroupList.AddString (szGroupName);
```

```
 }
 return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

GetAnnotationMarkCount Method

Description Returns the number of annotation marks on an image page or in an annotation group.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetAnnotationMarkCount**([*GroupName*,*AnnotationType*])

Arguments The GetAnnotationMarkCount method has the following arguments:

Parameter	Data Type	Setting
GroupName	String	The name of the annotation group.
AnnotationType	Integer (enumerated)	The desired AnnotationType setting (see page 529).

Returns Integer.

Remarks The **Display** method must be invoked prior to calling this method.

If the GroupName and Annotation Type parameters are both entered, the GetAnnotationMarkCount method returns the number of annotation marks of the type specified that are in the annotation group specified.

If the GroupName parameter is entered, but the Annotation Type is not entered, the GetAnnotationMarkCount method returns the total number of annotation marks that are in the annotation group specified.

If the GroupName parameter is not entered, but the Annotation Type is entered, the GetAnnotationMarkCount method returns the number of annotation marks of the type specified that are in all of the annotation groups.

If no parameters are entered, the GetAnnotationMarkCount method returns the total number of annotation marks that are on the page.

The GetAnnotationMarkCount method is valid only at run-time.

See Also AnnotationGroupCount property, AnnotationType property, Display method.

GetAnnotationMarkCount Example — VB

This example shows some methods for determining the number and types of annotations on a page.

```
Private Sub cmdMarkCount_Click()
    Dim intMarkCount As Integer
    'Returns a count of all marks on the page displayed.
    intMarkCount = ImgEdit1.GetAnnotationMarkCount
    'Returns a count of all marks in the annotation group called accounting.
    intMarkCount = ImgEdit1.GetAnnotationMarkCount("accounting")
    'Returns a count of all OCR regions.
    intMarkCount = ImgEdit1.GetAnnotationMarkCount(, wiOcrRegion)
    'Returns a count of all text annotation marks in the annotation group
    'called "accounting".
    intMarkCount = ImgEdit1.GetAnnotationMarkCount("accounting", wiText)
End Sub
```

GetAnnotationMarkCount Example — VC++

This example shows some methods for determining the number and types of annotations on a page.

```
void CImgEdit1Dlg::OnGetCount()
{
    // Some methods for determining the number and types of annotations on
    // a page.
    int iMarkCount;
    CString szCurGroup("accounting");
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    VARIANT vCurGroup; V_VT(&vCurGroup) = VT_BSTR;V_BSTR(&vCurGroup) =
    ➥ szCurGroup.AllocSysString();
    VARIANT vCurType; V_VT(&vCurType) = VT_I2;
    // Returns a count of all marks on the page displayed.
    iMarkCount = ImgEdit1.GetAnnotationMarkCount(evt,evt);
    // Returns a count of all marks in the annotation group called
    // accounting.
    iMarkCount = ImgEdit1.GetAnnotationMarkCount(vCurGroup,evt);
    // Returns a count of all OCR regions.
    V_I2(&vCurType) = 12;  // wiOcrRegion
    iMarkCount = ImgEdit1.GetAnnotationMarkCount(evt, vCurType);
    // Returns a count of all text annotation marks in the annotation group
    // called "accounting".
    V_I2(&vCurType) = 7;   // wiText
    iMarkCount = ImgEdit1.GetAnnotationMarkCount(vCurGroup, vCurType);
}
```

GetCurrentAnnotationGroup Method

Description Returns the name of the annotation group to which subsequent annotations will belong.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetCurrentAnnotationGroup**

Arguments None.

Returns String.

Remarks The GetCurrentAnnotationGroup method is valid only at run-time.

See Also GetAnnotationGroup method, SetCurrentAnnotationGroup method.

GetRubberStampItem Method

Description Returns the item number for the currently selected rubber stamp, according to its position on the Rubber Stamp Properties dialog box (see page 497) and drop-down menu.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetRubberStampItem**

Arguments None.

Returns Integer.

Remarks The item number is zero based. For example, if the Received stamp is selected, an ItemNumber of 2 is returned because it is the third item on the Rubber Stamp Properties dialog box and drop-down menu.

The Rubber Stamp Properties dialog box is accessible by right-clicking on the Rubber Stamp tool in the Annotation Tool Palette (see page 533). The drop-down menu is accessible by left-clicking on the tool.

See Also GetRubberStampMenuItems method, SetRubberStampItem method, ShowRubberStampDialog method.

GetRubberStampItem Example — VB

This example uses the GetRubberStampMenuItems method to populate a list box with the names of the rubber stamps that are currently available. It also uses the GetRubberStampItem method to obtain the item number of the stamp that is currently selected.

You can use these methods when designing a custom tool palette to let users select the type of rubber stamp they want to use.

```
Private Sub cmdStamps_Click()
    'Get the names of all available rubber stamps and write them to a
    'listbox. This could be used when designing a custom tool palette so
    'that when the user clicks on the stamp icon, they could
    'then select the desired stamp from a list.
    Dim strAllStamps As String
    Dim strStampName As String
    Dim intCurStamp As Integer
    Dim intStartPos As Integer
    Dim intEndPos As Integer
    'Get the stamp list.
    strAllStamps = frmMain.ImgEdit1.GetRubberStampMenuItems
    'Load the list box.
    Load frmStampList
    intStartPos = 0
    intEndPos = 1
    'Loop to parse stamps.
    Do While intEndPos <> 0
      'Find location of line feed characters which separate stamp names.
      intEndPos = InStr(intStartPos + 1, strAllStamps, vbLf)
      'If intEndPos is 0 we are at the end of the string.
      If intEndPos = 0 Then Exit Do
      'Retrieve text up to the line feed character.
      strStampName = Mid(strAllStamps, (intStartPos + 1),
      ➥  (intEndPos - intStartPos - 1))
      frmStampList.List1.AddItem strStampName
      intStartPos = intEndPos
    Loop
    'Highlight the stamp that is the currently selected stamp. This will be
    'the first stamp in the list if none have been selected; otherwise it
    'will be the most recently selected stamp.
    intCurStamp = frmMain.ImgEdit1.GetRubberStampItem
    frmStampList.List1.ListIndex = intCurStamp
    frmStampList.Show
End Sub
```

GetRubberStampItem Example — VC++

This example uses the GetRubberStampMenuItems method to populate a list box with the names of the currently available rubber stamps. It also uses the GetRubberStampItem method to obtain the item number of the stamp that is currently selected.

You can use these methods when designing a custom tool palette to let users select the type of rubber stamp they want to use.

```
BOOL CStampList::OnInitDialog()
{
    // Get the names of all available rubber stamps and write them to a
    // listbox. This could be used when designing a custom tool palette so
    // that when the user clicks on the stamp icon, they
    // could then select the desired stamp from a list.
    CDialog::OnInitDialog();
    CString strAllStamps;
    int intCurStamp;
    // Get the stamp list.
    strAllStamps = m_pParent->ImgEdit1.GetRubberStampMenuItems();
    // Loop to parse stamps.
    char sep[] = "\n";
    char *  token;
    /* Establish string and get the first token: */
    token = _tcstok( strAllStamps.GetBuffer(strAllStamps.GetLength
    ➥  ()),sep);
    while( token != NULL )
    {
      m_StampList.AddString(token);
      token = _tcstok( NULL, sep );
    }
    // Highlight the stamp that is the currently selected stamp.
    intCurStamp = m_pParent->ImgEdit1.GetRubberStampItem();
    m_StampList.SetCurSel(intCurStamp);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
void CStampList::OnOK()
{
    // Get the index number of the stamp that was selected in the list box.
    m_intNewStamp = m_StampList.GetCurSel();
    // Set the selected stamp as the current stamp to be used.
    CDialog::OnOK();
}
```

GetRubberStampMenuItems Method

Description Returns the menu items for the Rubber Stamp tool on the Annotation Tool Palette (see page 533).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetRubberStampMenuItems**

Arguments None.

Returns String.

Remarks The returned strings are separated by a new line character (\n).

The Rubber Stamp drop-down menu is accessible by left-clicking the Rubber Stamp tool on the palette.

See Also GetRubberStampItem method, SetRubberStampItem method.

GetRubberStampMenuItems Example — VB

This example uses GetRubberStampMenuItems to populate a list box with the names of the rubber stamps that are currently available. It also uses the GetRubberStampItem method to obtain the item number of the stamp that is currently selected.

You can use these methods when designing a custom tool palette to let users select the type of rubber stamp they want to use.

```
Private Sub cmdStamps_Click()
    'Get the names of all available rubber stamps and write them to a
    'listbox. This could be used when designing a custom tool palette so
    'that when the user clicks on the stamp icon, they could
    'then select the desired stamp from a list.
    Dim strAllStamps As String
    Dim strStampName As String
    Dim intCurStamp As Integer
    Dim intStartPos As Integer
    Dim intEndPos As Integer
    'Get the stamp list.
    strAllStamps = frmMain.ImgEdit1.GetRubberStampMenuItems
    'Load the list box.
    Load frmStampList
    intStartPos = 0
```

```
    intEndPos = 1
    'Loop to parse stamps.
    Do While intEndPos <> 0
      'Find location of line feed characters which separate stamp names.
      intEndPos = InStr(intStartPos + 1, strAllStamps, vbLf)
      'If intEndPos is 0 we are at the end of the string.
      If intEndPos = 0 Then Exit Do
      'Retrieve text up to the line feed character.
      strStampName = Mid(strAllStamps, (intStartPos + 1),
      ➥  (intEndPos - intStartPos - 1))
      frmStampList.List1.AddItem strStampName
      intStartPos = intEndPos
    Loop
    'Highlight the stamp that is the currently selected stamp. This will be
    'the first stamp in the list if none have been selected; otherwise it
    'will be the most recently selected stamp.
    intCurStamp = frmMain.ImgEdit1.GetRubberStampItem
    frmStampList.List1.ListIndex = intCurStamp
    frmStampList.Show
End Sub
```

GetRubberStampMenuItems Example — VC++

This example uses the GetRubberStampMenuItems method to populate a list box with the names of the rubber stamps that are currently available. It also uses the GetRubberStampItem method to obtain the item number of the stamp that is currently selected.

You can use these methods when designing a custom tool palette to let users select the type of rubber stamp they want to use.

```
BOOL CStampList::OnInitDialog()
{
    // Get the names of all available rubber stamps and write them to a
    // listbox. This could be used  when designing a custom tool palette so
    // that when the user clicks on the stamp icon, they
    // could then select the desired stamp from a list.
    CDialog::OnInitDialog();
    CString strAllStamps;
    int intCurStamp;
    // Get the stamp list.
    strAllStamps = m_pParent->ImgEdit1.GetRubberStampMenuItems();
    // Loop to parse stamps.
    char sep[] = "\n";
    char *  token;
    /* Establish string and get the first token: */
    token = _tcstok( strAllStamps.GetBuffer(strAllStamps.GetLength
    ➥  ()),sep);
    while( token != NULL )
    {
      m_StampList.AddString(token);
      token = _tcstok( NULL, sep );
```

```
    }
    // Highlight the stamp that is the currently selected stamp.
    intCurStamp = m_pParent->ImgEdit1.GetRubberStampItem();
    m_StampList.SetCurSel(intCurStamp);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
void CStampList::OnOK()
{
    // Get the index number of the stamp that was selected in the list box.
    m_intNewStamp = m_StampList.GetCurSel();
    // Set the selected stamp as the current stamp to be used.
    CDialog::OnOK();
}
```

GetSelectedAnnotationBackColor Method

Description Returns the background color of a selected Attach-a-Note annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationBackColor**

Arguments None.

Returns OLE_COLOR.

Remarks The GetSelectedAnnotationBackColor method is valid only at run-time. It is used with annotations of the Attach-a-Note type.

Color is expressed using the RGB format (see page 540).

See Also AnnotationBackColor property, AnnotationType property, SetSelectedAnnotationBackColor method.

GetSelectedAnnotationBackColor Example — VB

This example uses the GetSelectedAnnotationBackColor method, the SetSelectedAnnotationBackColor method, and the Microsoft common dialog box to change the background color of a selected text annotation.

```
Private Sub cmdBackColor_Click()
    'This example uses the Microsoft color dialog box to change
    'a selected text attachment annotation mark's background color.
    Dim BackColor As OLE_COLOR
```

```
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    BackColor = ImgEdit1.GetSelectedAnnotationBackColor
    CommonDialog1.Color = BackColor
    CommonDialog1.Flags = cdlCCRGBInit
    'Display the dialog box, and set the annotation background color to the
    'color selected.
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationBackColor CommonDialog1.Color
End Sub
```

GetSelectedAnnotationBackColor Example — VC++

This example uses the GetSelectedAnnotationBackColor method, the SetSelectedAnnotationBackColor method, and the Microsoft common dialog box to change the background color of a selected text annotation.

```
void CImgEdit2Dlg::OnBackcolor()
{
    // This example uses the Microsoft color dialog box to change
    // a selected text attachment annotation mark's background color.
    long BackColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    BackColor = ImgEdit1.GetSelectedAnnotationBackColor();
    CommonDialog1.SetColor(BackColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationBackColor(CommonDialog1.GetColor());
}
```

GetSelectedAnnotationFillColor Method

Description Returns the color used to fill a selected Filled Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationFillColor**

Arguments None.

Returns OLE_COLOR.

Remarks The GetSelectedAnnotationFillColor method is valid only at run-time. It is used with annotations of the Filled Rectangle type.

Color is expressed using the RGB format (see page 540).

See Also AnnotationFillColor property, AnnotationType property, SetSelectedAnnotationFillColor method.

GetSelectedAnnotationFillColor Example — VB

This example uses the GetSelectedAnnotationFillColor method to determine the fill color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFillColor method to change the fill color of a selected annotation.

```
Private Sub cmdFillColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's fill color.
    Dim FillColor As OLE_COLOR
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    FillColor = ImgEdit1.GetSelectedAnnotationFillColor
    CommonDialog1.Color = FillColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationFillColor CommonDialog1.Color
End Sub
```

GetSelectedAnnotationFillColor Example — VC++

This example uses the GetSelectedAnnotationFillColor method to determine the fill color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFillColor method to change the fill color of a selected annotation.

```
void CImgEdit2Dlg::OnFillcolor()
{
    // This example uses the Microsoft color dialog box to
    // change a selected annotation mark's fill color.
    long FillColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    FillColor = ImgEdit1.GetSelectedAnnotationFillColor();
    CommonDialog1.SetColor(FillColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationFillColor(CommonDialog1.GetColor());
}
```

GetSelectedAnnotationFillStyle Method

Description Returns the style used to fill a selected image or Filled Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationFillStyle**

Arguments None.

Returns Integer (enumerated).

Contant	Value	Description
wiTransparent	0	Transparent
wiOpaque	1	Opaque

Remarks The GetSelectedAnnotationFillStyle method is valid only at run-time. It is used with annotations of the following types:

- Filled Rectangle
- Image Embedded
- Image Reference

See Also AnnotationFillStyle property, AnnotationType property, SetSelectedAnnotationFillStyle method.

GetSelectedAnnotationFillStyle Example — VB

This example uses the GetSelectedAnnotationFillStyle method to determine whether a selected image or filled rectangle annotation is opaque or transparent, and the SetSelectedAnnotationFillStyle method to change the fill style.

```
Private Sub Form_Load()
'This example determines whether a selected annotation image or
'filled rectangle is opaque or transparent and allows that
'attribute to be changed. The form used has 2 option buttons --
'one labeled "Opaque" and one labeled "Transparent".
Dim FillStyle As AnnotationStyleConstants
'Determine if the selected image annotation is opaque or
'transparent and set the corresponding option button on the form.
FillStyle = frmMain.ImgEdit1.GetSelectedAnnotationFillStyle
If FillStyle = wiOpaque Then
  optOpaque.Value = True
```

```
    Else
      optTransparent.Value = True
    End If
    frmFillStyle.Show
End Sub
Private Sub optOpaque_Click()
     'If the user clicks on the Opaque option button the annotation is
     'changed to be opaque.
     frmMain.ImgEdit1.SetSelectedAnnotationFillStyle wiOpaque
End Sub
```

GetSelectedAnnotationFillStyle Example — VC++

This example uses the GetSelectedAnnotationFillStyle method to determine whether a selected image or filled rectangle annotation is opaque or transparent, and the SetSelectedAnnotationFillStyle method to change the fill style.

```
void CLineStyles::OnOK()
{
    if(m_Transparent.GetCheck() && m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(1);
    if(m_Transparent.GetCheck() && !m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(1);
    CDialog::OnOK();
}
BOOL CLineStyles::OnInitDialog()
{
    CDialog::OnInitDialog();
    short LineStyle;
    // Determine if the selected line or fill annotation is opaque or
    // transparent and set the corresponding option button on the form.
    if(m_bLineStyle)
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationLineStyle();
    else
     {
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationFillStyle();
      SetWindowText("Fill Styles");
     }
    if(LineStyle == 1)  //  wiOpaque
     {
      m_Opaque.SetCheck(1);
      m_Transparent.SetCheck(0);
     }
    else
     {
      m_Opaque.SetCheck(0);
      m_Transparent.SetCheck(1);
     }
```

```
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

GetSelectedAnnotationFont Method

Description Returns a font object's properties. Applies to selected text annotations.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationFont**

Arguments None.

Returns IFontDisp.

Remarks The GetSelectedAnnotationFont method is valid only at run-time. It is used with annotations of the following types:

- Attach-a-Note
- Text
- Hyperlink
- Text From File
- Text Stamp

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

See Also AnnotationFont property, AnnotationType property, SetSelectedAnnotationFont method.

GetSelectedAnnotationFont Example — VB

This example uses the GetSelectedAnnotationFont method to determine the font attributes of a selected annotation. It then updates the Microsoft common dialog box to indicate the attributes obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFont method to change the font attributes of a selected annotation.

```
Private Sub cmdGetAnnoFont_Click()
    'This example uses the Microsoft font dialog box to
    'change a selected text annotation mark's font.
    Dim AnnoFont As IFontDisp
```

```
    Dim FontColor As OLE_COLOR
    'Determine the current font used in the selected annotation
    'mark and init the dialog box font property to that font.
    Set AnnoFont = ImgEdit1.GetSelectedAnnotationFont
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor
    CommonDialog1.FontName = AnnoFont.Name
    CommonDialog1.FontBold = AnnoFont.Bold
    CommonDialog1.FontItalic = AnnoFont.Italic
    CommonDialog1.FontSize = AnnoFont.Size
    CommonDialog1.FontUnderline = AnnoFont.Underline
    CommonDialog1.FontStrikethru = AnnoFont.Strikethrough
    CommonDialog1.Color = FontColor
    CommonDialog1.Flags = cdlCFScreenFonts Or cdlCFEffects
    CommonDialog1.ShowFont
    'Take values set in the font dialog box and set these attributes in the
    'annotation font.
    AnnoFont.Name = CommonDialog1.FontName
    AnnoFont.Bold = CommonDialog1.FontBold
    AnnoFont.Italic = CommonDialog1.FontItalic
    AnnoFont.Size = CommonDialog1.FontSize
    AnnoFont.Underline = CommonDialog1.FontUnderline
    AnnoFont.Strikethrough = CommonDialog1.FontStrikethru
    'Apply the new font and font color to the selected annotation.
    ImgEdit1.SetSelectedAnnotationFont AnnoFont
    ImgEdit1.SetSelectedAnnotationFontColor CommonDialog1.Color
End Sub
```

GetSelectedAnnotationFont Example — VC++

This example uses the GetSelectedAnnotationFont method to determine the font attributes of a selected annotation. It then updates the Microsoft common dialog box to indicate the attributes obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFont method to change the font attributes of a selected annotation.

```
void CImgEdit2Dlg::OnAnnofont()
{
    // This example uses the Microsoft font dialog box to
    // change a selected text annotation mark's font.
    COleFont AnnoFont;
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    CY cy;
    long FontColor;
    // Determine the current font used in the selected annotation
    // mark and init the dialog box font property to that font.
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor();
    CommonDialog1.SetFontName(AnnoFont.GetName());
    CommonDialog1.SetFontBold(AnnoFont.GetBold());
    CommonDialog1.SetFontItalic(AnnoFont.GetItalic());
    cy = AnnoFont.GetSize();
```

```
    CommonDialog1.SetFontSize((float)cy.Lo / 10000);
    CommonDialog1.SetFontUnderLine(AnnoFont.GetUnderline());
    CommonDialog1.SetFontStrikeThru(AnnoFont.GetStrikethrough());
    CommonDialog1.SetColor(FontColor);
    CommonDialog1.SetFlags(0x101); //cdlCFScreenFonts Or cdlCFEffects
    CommonDialog1.ShowFont();
    // Take values set in the font dialog box and set these attributes in
    // the annotation font.
    AnnoFont.SetName(CommonDialog1.GetFontName());
    AnnoFont.SetBold(CommonDialog1.GetFontBold());
    AnnoFont.SetItalic(CommonDialog1.GetFontItalic());
    cy.Lo = (unsigned long)CommonDialog1.GetFontSize() * 10000;
    AnnoFont.SetSize(cy);
    AnnoFont.SetUnderline(CommonDialog1.GetFontUnderLine());
    AnnoFont.SetStrikethrough(CommonDialog1.GetFontStrikeThru());
    // Apply the new font and font color to the selected annotation.
    ImgEdit1.SetSelectedAnnotationFont(AnnoFont);
    ImgEdit1.SetSelectedAnnotationFontColor(CommonDialog1.GetColor());
}
```

GetSelectedAnnotationFontColor Method

Description Returns the color used in a selected text annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationFontColor**

Arguments None.

Returns OLE_COLOR.

Remarks The GetSelectedAnnotationFontColor method is valid only at run-time. It is used with annotations of the following types:

- Attach-a-Note
- Hyperlink
- Text
- Text From File
- Text Stamp

Color is expressed using the RGB format (see page 540).

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

See Also AnnotationFontColor property, AnnotationType property, SetSelectedAnnotationFontColor method.

GetSelectedAnnotationFontColor Example — VB

This example uses the GetSelectedAnnotationFontColor method to determine the font color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFontColor method to change the font color of a selected annotation.

```
Private Sub cmdFontColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's font color.
    Dim FontColor As OLE_COLOR
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor
    CommonDialog1.Color = FontColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationFontColor CommonDialog1.Color
End Sub
```

GetSelectedAnnotationFontColor Example — VC++

This example uses the GetSelectedAnnotationFontColor method to determine the font color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFontColor method to change the font color of a selected annotation.

```
void CImgEdit2Dlg::OnAnnofont()
{
    // This example uses the Microsoft font dialog box to
    // change a selected text annotation mark's font.
    COleFont AnnoFont;
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    CY cy;
    long FontColor;
    // Determine the current font used in the selected annotation
    // mark and init the dialog box font property to that font.
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor();
```

```
    CommonDialog1.SetFontName(AnnoFont.GetName());
    CommonDialog1.SetFontBold(AnnoFont.GetBold());
    CommonDialog1.SetFontItalic(AnnoFont.GetItalic());
    cy = AnnoFont.GetSize();
    CommonDialog1.SetFontSize((float)cy.Lo / 10000);
    CommonDialog1.SetFontUnderLine(AnnoFont.GetUnderline());
    CommonDialog1.SetFontStrikeThru(AnnoFont.GetStrikethrough());
    CommonDialog1.SetColor(FontColor);
    CommonDialog1.SetFlags(0x101); //cdlCFScreenFonts Or cdlCFEffects
    CommonDialog1.ShowFont();
    // Take values set in the font dialog box and set these attributes in
    // the annotation font.
    AnnoFont.SetName(CommonDialog1.GetFontName());
    AnnoFont.SetBold(CommonDialog1.GetFontBold());
    AnnoFont.SetItalic(CommonDialog1.GetFontItalic());
    cy.Lo = (unsigned long)CommonDialog1.GetFontSize() * 10000;
    AnnoFont.SetSize(cy);
    AnnoFont.SetUnderline(CommonDialog1.GetFontUnderLine());
    AnnoFont.SetStrikethrough(CommonDialog1.GetFontStrikeThru());
    // Apply the new font and font color to the selected annotation.
    ImgEdit1.SetSelectedAnnotationFont(AnnoFont);
    ImgEdit1.SetSelectedAnnotationFontColor(CommonDialog1.GetColor());
}
```

GetSelectedAnnotationImage Method

Description Returns the fully-qualified name (see page 535) of the image that is being used in a selected image annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationImage**

Arguments None.

Returns String.

Remarks The GetSelectedAnnotationImage method is valid only at run-time. It is used with annotations of the following types:

- Image Embedded
- Image Reference

See Also AnnotationImage property, AnnotationType property.

GetSelectedAnnotationImage Example — VB

This example uses the GetSelectedAnnotationImage method to obtain the fully-qualified file name of the image being used in a selected image annotation.

```
Private Sub CmdGetAnnoImg_Click()
    'Displays a message box which shows the fully qualified
    'filename for a selected annotation image.
    Dim strAnnImg As String
    'The MarkSelect event could be used to determine if
    'the appropriate type of annotation mark is selected.
    strAnnImg = ImgEdit1.GetSelectedAnnotationImage
    MsgBox "The annotation image is " & strAnnImg
End Sub
```

GetSelectedAnnotationLineColor Method

Description Returns the line color used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationLineColor**

Arguments None.

Returns OLE_COLOR.

Remarks The GetSelectedAnnotationLineColor method is valid only at run-time. It is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

Color is expressed using the RGB format (see page 540).

See Also AnnotationLineColor property, AnnotationType property, SetSelectedAnnotationLineColor method.

GetSelectedAnnotationLineColor Example — VB

This example uses the GetSelectedAnnotationLineColor method to determine the color of a selected line annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationLineColor method to change the color of a selected line annotation.

```
Private Sub cmdLineColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's line color.
    Dim LineColor As OLE_COLOR
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    LineColor = ImgEdit1.GetSelectedAnnotationLineColor
    CommonDialog1.Color = LineColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationLineColor CommonDialog1.Color
End Sub
```

GetSelectedAnnotationLineColor Example — VC++

This example uses the GetSelectedAnnotationLineColor method to determine the color of a selected line annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationLineColor method to change the color of a selected line annotation.

```
void CImgEdit2Dlg::OnLinecolor()
{
    // This example uses the Microsoft color dialog box to
    // change a selected annotation mark's line color.
    long LineColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    LineColor = ImgEdit1.GetSelectedAnnotationLineColor();
    CommonDialog1.SetColor(LineColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationLineColor(CommonDialog1.GetColor());
}
```

GetSelectedAnnotationLineStyle Method

Description Returns the line style used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationLineStyle**

Arguments None.

Returns Integer (enumerated).

Contant	Value	Description
wiTransparent	0	Transparent
wiOpaque	1	Opaque

Remarks The GetSelectedAnnotationLineStyle method is valid only at run-time. It is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

See Also AnnotationLineStyle property, AnnotationType property, SetSelectedAnnotationLineStyle method.

GetSelectedAnnotationLineStyle Example — VB

This example uses the GetSelectedAnnotationLineStyle method to determine whether a selected line or hollow rectangle annotation is opaque or transparent, and the SetSelectedAnnotationLineStyle method to change the line style.

```
Private Sub Form_Load()
    'This example determines whether a selected annotation image or
    'filled rectangle is opaque or transparent and allows that
    'attribute to be changed. The form used has 2 option buttons --
    'one labeled "Opaque" and one labeled "Transparent".
    Dim LineStyle As AnnotationStyleConstants
    'Determine if the selected line annotation is opaque or transparent and
    'set the corresponding option button on the form.
    LineStyle = frmMain.ImgEdit1.GetSelectedAnnotationLineStyle
    If LineStyle = wiOpaque Then
      optOpaque.Value = True
```

```
    Else
      optTransparent.Value = True
    End If
    frmLineStyle.Show
End Sub
Private Sub optOpaque_Click()
    'If the user clicks on the Opaque option button the annotation is
    'changed to be opaque.
    frmMain.ImgEdit1.SetSelectedAnnotationLineStyle wiOpaque
End Sub
Private Sub optTransparent_Click()
    'If the user clicks on the Transparent option button the annotation
    'is changed to be transparent.
    frmMain.ImgEdit1.SetSelectedAnnotationLineStyle wiTransparent
End Sub
```

GetSelectedAnnotationLineStyle Example — VC++

This example uses the GetSelectedAnnotationLineStyle method to determine whether a selected line or hollow rectangle annotation is opaque or transparent, and the SetSelectedAnnotationLineStyle method to change the line style.

```
void CLineStyles::OnOK()
{
    if(m_Transparent.GetCheck() && m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(1);
    if(m_Transparent.GetCheck() && !m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(1);
    CDialog::OnOK();
}
BOOL CLineStyles::OnInitDialog()
{
    CDialog::OnInitDialog();
    short LineStyle;
    // Determine if the selected line or fill annotation is opaque or
    // transparent and set the corresponding option button on the form.
    if(m_bLineStyle)
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationLineStyle();
    else
     {
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationFillStyle();
      SetWindowText("Fill Styles");
     }
    if(LineStyle == 1)  //  wiOpaque
     {
      m_Opaque.SetCheck(1);
      m_Transparent.SetCheck(0);
     }
```

```
    else
     {
      m_Opaque.SetCheck(0);
      m_Transparent.SetCheck(1);
     }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

GetSelectedAnnotationLineWidth Method

Description Returns the line width in pixels used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationLineWidth**

Arguments None.

Returns Integer.

Remarks The GetSelectedAnnotationLineWidth method is valid only at run-time. It is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

See Also AnnotationLineWidth property, AnnotationType property, SetSelectedAnnotationLineWidth method.

GetSelectedAnnotationLineWidth Example — VB

This example uses the GetSelectedAnnotationLineWidth method to determine the width of a selected line annotation. It then updates a text box and a Line control to illustrate the width.

```
Private Sub Form_Load()
    'First, determine an annotation line's width and color. This is then
    'used in a dialog box which contains a sample line with a text box to
    'indicate the line width (in pixels) and an updown control linked to the
    'text box to allow the width to be scrolled up and down. The user can
    'change the width of the line in the dialog box in order to change the
    'width of the annotation.
    Dim intLineWidth As Integer
```

```
    Dim LineColor As OLE_COLOR
    'Get the line width and color to initialize the dialog box.
    intLineWidth = frmMain.ImgEdit1.GetSelectedAnnotationLineWidth
    LineColor = frmMain.ImgEdit1.GetSelectedAnnotationLineColor
    'Set current line width in the text box to the width of the pixel width
    'of the annotation mark.
    txtWidth.Text = Str(intLineWidth)
    'Set the value of the UpDown control to the current line width.
    UpDown1.Value = intLineWidth
    'Set the width of the sample line in the dialog box.
    Line1.BorderWidth = intLineWidth
    'Set the color of the sample line.
    Line1.BorderColor = LineColor
    'Display the dialog box.
    frmLineWidth.Show
End Sub
```

GetSelectedAnnotationLineWidth Example — VC++

This example uses the GetSelectedAnnotationLineWidth method to determine the width of a selected line annotation. It then displays the line width in a text box control.

It also uses the SetSelectedAnnotationLineWidth method to set the width of a selected line annotation to the value entered in the text box control.

```
BOOL CLineWidth::OnInitDialog()
{
    CDialog::OnInitDialog();
    int intLineWidth;
    long LineColor;
    // Get the line width and color to initialize the dialog box.
    intLineWidth = m_pParent->ImgEdit1.GetSelectedAnnotationLineWidth();
    LineColor = m_pParent->ImgEdit1.GetSelectedAnnotationLineColor();
    // Set the current line width in the text box.
    char tLineWidth [5];
    _itoa(intLineWidth,tLineWidth,10);
    m_Edit.SetWindowText(tLineWidth);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
void CLineWidth::OnOK()
{
    CString tLineWidth;
    int iLineWidth;
    m_Edit.GetWindowText(tLineWidth);
    iLineWidth = atoi(tLineWidth);
    m_pParent->ImgEdit1.SetSelectedAnnotationLineWidth(iLineWidth);
    CDialog::OnOK();
}
```

GetSelectedAnnotationOcrType Method

Description Returns the type of OCR zone drawn on an image page.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**GetSelectedAnnotationOcrType**

Arguments None.

Returns Integer (enumerated)

Constant	Value	Description
wiOcrTypeText	0	OCR Text zone
wiOcrTypePicture	1	OCR Picture zone

Remarks The GetSelectedAnnotationOcrType method is valid only at run-time. An OCR zone must be selected prior to invoking this method.

Applies to the OCR Zones annotation type only.

See Also AnnotationOcrType property, AnnotationType property, ImageOCR control, OcrZoneVisibility property, SelectFirstOcrZone method, SelectNextOcrZone method, SetSelectedAnnotationOcrType method.

GetSelectedAnnotationOcrType Example — VB

This example uses the GetSelectedAnnotationOcrType method to determine whether a selected OCR Zone annotation is a Picture zone or a Text zone. It then updates the appropriate option button on the form to indicate the OCR type.

This example also allows the user to change the OCR type of a selected OCR Zone annotation using the option buttons on the form. It uses the SetSelectedAnnotationOcrType method to set the OCR type.

```
Private Sub Form_Load()
    'This example determines whether a selected OCR recognition zone is
    'defined as a Picture Zone or a Text Zone and shows a dialog box to
    'allow the user to change the OCR zone type. Form used has 2
    'option buttons labeled Picture Zone and Text Zone.
    Dim OCRType As AnnotationOcrTypeConstants
    'Determine if the selected OCR annotation mark is a Text Zone or a
    'Picture Zone and set the corresponding option button on the form.
    OCRType = frmMain.ImgEdit1.GetSelectedAnnotationOcrType
```

```
    If OCRType = wiOcrTypeText Then
      optTextZone.Value = True
    Else
      optPictureZone.Value = True
    End If
   frmOCRType.Show
End Sub
Private Sub optPictureZone_Click()
    'If the user clicks on the Picture Zone option button, the OCR
    'Zone type is changed to a Picture Zone.
    frmMain.ImgEdit1.SetSelectedAnnotationOcrType wiOcrTypePicture
End Sub
Private Sub optTextZone_Click()
    'If the user clicks on the Text Zone option button, the OCR
    'Zone type is changed to a Text Zone.
    frmMain.ImgEdit1.SetSelectedAnnotationOcrType wiOcrTypeText
End Sub
```

GetSelectedAnnotationOcrType Example — VC++

This example uses the GetSelectedAnnotationOcrType method to determine whether a selected OCR Zone annotation is a Picture zone or a Text zone. It then updates the appropriate check box to indicate the OCR type.

This example also allows the user to change the OCR type of a selected OCR Zone annotation using a check box. It uses the SetSelectedAnnotationOcrType method to set the OCR type.

```
void COCRZones::OnOK()
{
    if(m_TextZone.GetCheck())
      m_pParent->ImgEdit1.SetSelectedAnnotationOcrType(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationOcrType(1);
    CDialog::OnOK();
}
BOOL COCRZones::OnInitDialog()
{
    CDialog::OnInitDialog();
    short OCRType;
    // Determine if the selected OCR annotation mark is a Text Zone or a
    // Picture Zone and set the corresponding option button on the form.
    OCRType = m_pParent->ImgEdit1.GetSelectedAnnotationOcrType();
    if(OCRType == 0)   // wiOcrTypeText
    {
      m_TextZone.SetCheck(1);
      m_PictureZone.SetCheck(0);
    }
    else
    {
      m_TextZone.SetCheck(0);
      m_PictureZone.SetCheck(1);
```

```
    }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

GetVersion Method

Description Returns the Product Version number of the Imaging control (for example, 1.09.1403).

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

The GetVersion method of the Image Annotation Tool Button control is available in all versions of Imaging for Windows.

Applies To Image Edit, Image Annotation, Image Admin, Image OCR, Image Scan, and Image Thumbnail controls.

Usage *object*.**GetVersion**

Arguments None.

Returns String.

Remarks None.

HideAnnotationGroup Method

Description Hides an annotation group.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**HideAnnotationGroup** [*GroupName*]

Arguments The HideAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
GroupName	String	The name of the annotation group to hide

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

If the GroupName parameter is not specified, the HideAnnotationGroup method hides all annotation groups. This method does not hide OCR Zone annotations. To hide OCR Zone annotations, use the **OcrZoneVisibility** property.

Invoking the HideAnnotationGroup method sets the **ImageModified** property to True.

Note: OCR Zone annotations are available with Imaging for Windows Professional Edition only.

See Also Display method, GetAnnotationGroup method, GetCurrentAnnotationGroup method, ImageModified property, OcrZoneVisibility property, SetCurrentAnnotationGroup method, ShowAnnotationGroup method.

HideAnnotationGroup Example — VB

This example lets a user select an annotation group from a list and hide it. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
Private Sub cmdHideGroup_Click()
    Dim strCurGroup As String
    'Determine which group the user selected from the listbox.
    strCurGroup = AnnoGroups.List(AnnoGroups.ListIndex)
    Form1.ImgEdit1.HideAnnotationGroup (strCurGroup)
    'Call refresh to repaint with the group hidden.
    Form1.ImgEdit1.Refresh
End Sub
```

HideAnnotationGroup Example — VC++

This example lets a user select an annotation group from a list and hide it. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
void CFrmGroup::OnHideGroup()
{
    // This allows a user to select a Group from a list and hide it. See
    // GetAnnotationGroup method example for how to list annotation groups.
    CString szCurGroup;
    // Get the name of the group the user clicked on in the listbox.
    m_GroupList.GetText(m_GroupList.GetCurSel(),szCurGroup);
    if(pParentDlg)
    {
      VARIANT vCurGroup; V_VT(&vCurGroup) = VT_BSTR; V_BSTR(&vCurGroup) =
      ➥ szCurGroup.AllocSysString();
      pParentDlg->ImgEdit1.HideAnnotationGroup (vCurGroup);
      // Call refresh to repaint with the group hidden.
      pParentDlg->ImgEdit1.Refresh();
    }
}
```

HideAnnotationToolPalette Method

Description Hides the Annotation Tool Palette (see page 533).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**HideAnnotationToolPalette**

Arguments None.

Returns None.

Remarks The **ShowAnnotationToolPalette** method must be invoked prior to calling this method.

See Also ShowAnnotationToolPalette method, ToolPaletteHidden event.

Invert Method

Description Inverts the image, making it appear like a photographic negative.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Invert**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

After invoking the Invert method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image.

Invoking the Invert method sets the **ImageModified** property to True.

See Also Display method, ImageModified property, Save method, SaveAs method, SavePage method.

IsClipboardDataAvailable Method

Description Determines whether image or annotation data is present on the Clipboard.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**IsClipboardDataAvailable**

Arguments None.

Returns Boolean.

Value	Description
True	Image or annotation data is on the clipboard.
False	Image or annotation data is not on the clipboard.

Remarks Use the IsClipboardDataAvailable method to determine whether the **ClipboardPaste** method can be invoked.

See Also ClipboardCopy method, ClipboardCut method, ClipboardPaste method.

IsClipboardDataAvailable Example — VB

This example uses the IsClipboardDataAvailable method to determine whether there is data on the Clipboard. If True, it enables the Paste item on the Edit menu.

```
Private Sub mnuEdit_Click()
    'This example uses the IsClipboardDataAvailable method to determine
    'whether there is data on the clipboard, and if so, will enable the
    'Paste menu item on the Edit menu.
    Dim blnClipData As Boolean
    blnClipData = ImgEdit1.IsClipboardDataAvailable
    If blnClipData Then
      mnuEditPaste.Enabled = True
    Else
      mnuEditPaste.Enabled = False
    End If
End Sub
```

IsClipboardDataAvailable Example — VC++

This example uses the IsClipboardDataAvailable method to determine whether there is data on the Clipboard. If True, it enables the Paste item on the Edit menu.

```
void CImgEdit2Dlg::OnEnablepaste()
{
    // This example uses the IsClipboardDataAvailable method to determine
    // whether to enable the Paste menu item on the Edit menu.
    int nlnClipData;
    nlnClipData = ImgEdit1.IsClipboardDataAvailable();
    if(nlnClipData)
      AfxMessageBox("Paste Enabled");
    else
      AfxMessageBox("Paste Disabled");
}
```

LoadAnnotations Method

Description Loads annotations from a file and places them on the image currently being displayed.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**LoadAnnotations** *FileName*,*StartPage*,*EndPage*,*Flags*

Arguments The LoadAnnotations method has the following arguments:

Parameter	Data Type	Setting
FileName	String	The fully-qualified name (see page 535) of the file containing the annotations.
StartPage	Long	The starting page in the currently displayed file where the first annotation will be placed.
EndPage	Long	The ending page in the currently displayed file where the last annotation will be placed.
Flags	Long	Not used currently.

Returns Long.

Value	Description
0	Failed
1	Success

Remarks The **Display** method must be invoked prior to calling this method.

To support StartPage and EndPage parameter entries that span more than one page, the image file on display must be a multipage image file type (AWD or TIFF).

Notes Regarding the StartPage and EndPage Parameters

If the StartPage parameter is set to 5, the first annotation will be placed on page 5 of the image file currently on display. If the EndPage parameter is set to 8, the first four groups of annotations will be placed on pages 5, 6, 7 and 8 of the image file currently on display. If there are more annotations than pages, or more pages than annotations, the LoadAnnotations method will perform the best fit it can.

After invoking the LoadAnnotations method, the **Save**, **SaveAs**, or **SavePage** method must be invoked to save the altered image. Unless the current image is a TIFF image, annotations must be burned-in to be saved.

See Also Display method, Save method, SaveAnnotations method, SaveAs method, SavePage method.

LoadAnnotations Example — VB

This example uses the LoadAnnotations method to load annotations from a file that is separate from the image file.

```
Private Sub cmdLoadAnnos_Click()
    'Load annotations which were previously saved to a new image file. You
    'could create a template for OCR recognition zones using these methods.
    'Used in conjunction with the SaveAnnotations method. Load annotations
    'from file ocrzones.ano onto page 1 of the currently displayed image.
    ImgEdit1.LoadAnnotations "D:\ocrzones.ano", 1, 1, 0
End Sub
```

LoadAnnotations Example — VC++

This example uses the LoadAnnotations method to load annotations from a file that is separate from the image file.

```
void CImgEdit2Dlg::OnLoadannotations()
{
    // Load annotations which were previously saved on to a new image file.
    // You could create a template for OCR recognition zones using these
    // methods. Used in conjunction with the SaveAnnotations method.
    // Load annotations from file ocrzones.ano onto page 1 of the the
    // currently displayed image.
    ImgEdit1.LoadAnnotations ("c:\\savemark.ano", 1, 1, 0);

}
```

ManualDeSkew Method

Description Displays the Straighten Page dialog box (see page 523), which permits end users to manually deskew (straighten) the displayed image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ManualDeSkew**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The ManualDeSkew method causes the Image Edit control to capture all mouse events. Therefore, you should not display any dialog boxes or other objects that expect to receive mouse events.

The ManualDeSkew method modifies the displayed image. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the ManualDeSkew method sets the **ImageModified** property to True.

See Also AutoDeskew method, Display method, ImageModified property, Page property, Save method, SaveAs method, SavePage method, StraightenPage event.

PrintImage Method

Description Prints the image specified in the **Image** property.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 95 and 98

√ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**PrintImage** [*StartPage*,*EndPage*,*OutputFormat*, *Annotations*,*Printer*,*Driver*,*PortNumber*]

Arguments The PrintImage method has the following arguments:

Parameter	Data Type	Setting
StartPage	Long	The starting page in a range of pages to be printed.
EndPage	Long	The ending page in a range of pages to be printed.
OutputFormat	Integer (enumerated)	The output format to be used: 0 — Pixel to pixel 1 — Inch to inch 2 — Fit to page 3 — Best fit
Annotations	Boolean	Whether annotations and OCR zones will be printed with the image: True — Annotations and OCR zones will be printed. False — Annotations and OCR zones will not be printed.
Printer	String or	The name of the printer.
	Long	The Printer Device Context (HDC) value (C++ only — see “Binder Printing” on page 434).
Driver	String	The name of the printer driver (applicable only when the Printer parameter is specified as a string).
PortNumber	String	The name of the output port (applicable only when the Printer parameter is specified as a string).

The OutputFormat parameter setting of 3 (Best fit) is available with Imaging for Windows Professional Edition V2.0 only.

The Printer Device Context (HDC) Printer parameter setting is available with Imaging for Windows Professional Edition and Imaging for Windows 98 only.

OutputFormat Default Settings By Product Version

Product Version	Default Setting
Imaging for Windows Professional Edition V1.0 and V1.1 Imaging for Windows 98	1 — Inch to inch
Imaging for Windows 95 and NT 4.0	2 — Fit to page
Imaging for Windows Professional Edition V2.0	3 — Best fit

Returns None.

Remarks If an image is displayed when the PrintImage method is invoked, the image is printed as it appears, including any changes made.

If an image is not being displayed, the PrintImage method prints the image specified in the **Image** property. If an image is not being displayed and the Image property does not contain the name of an image, the PrintImage method generates an error.

If the StartPage and EndPage parameters are not entered, this method prints all pages by default.

If the Annotations parameter is not specified, all annotations are printed by default.

Binder Printing

This option is intended for use when printing images from Microsoft Binder.

If you use a hardware device context (HDC) value to specify the Printer parameter, you must call StartDoc and StartPage prior to passing the HDC value. And you must call EndPage and EndDoc after passing the HDC value.

HDC is a C++ interface only; it is generated through C++ calls. StartDoc and StartPage are C++ calls that must be executed prior to calling the PrintImage method. EndPage and EndDoc are C++ calls that must be executed after calling the PrintImage method. Note that if you specify an HDC value for the Printer parameter, you do not have to specify the Driver and PortNumber parameters.

See Also CheckContinuePrinting event, ContinuePrinting property, Image property, UseCheckContinuePrinting property.

PrintImage Example — VB

This example prints the image that is currently displayed in the Image Edit control. It uses the Print Dialog box in the Admin control to obtain parameters for the print-out.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Set filename to be printed to the displayed file. If this property is
    'not set the dialog will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
     'User pressed the cancel button.
     Exit Sub
End Sub
```

PrintImage Example — VC++

This example prints the image that is currently displayed in the Image Edit control. It uses the Print Dialog box in the Admin control to obtain parameters for the print-out.

```
void CImgEdit1Dlg::OnPrint()
{
    // This example will print the image that is displayed in the Image Edit
    // control. It uses the Print Dialog box in the Admin control to obtain
    // parameters for the print output.
    // Display an image.
    ImgEdit1.SetImage("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Set filename to be printed to the displayed file. If this property is
    // not set the dialog will not display.
    ImgAdmin1.SetImage(ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.PrintImage(evt, evt, evt, evt, evt, evt, evt);
}
```

Redo Method

Description Redoes the last imaging operation that was undone by the **Undo** method.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 98

Imaging for Windows 95

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Redo** [*Option*]

Arguments The Redo method has the following argument:

Parameter	Data Type	Setting
Option	Long	0 (default)

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The Redo method requires the successful completion of an undo operation (using the Undo method). Imaging deletes the redo information when the next imaging operation is performed. If Redo is called before anything is undone, an error is generated. Check the StatusCode property for the error.

The Redo method modifies the displayed image. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the Redo method sets the **ImageModified** property to True.

See Also StatusCode property, Display method, ImageModified property, Save method, SaveAs method, SavePage method, Undo method.

Refresh Method

Description Repaints the contents of the Image Edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Refresh**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

See Also Display method, DisplayScaleAlgorithm property, ImagePalette property, ImageResolutionX property, ImageResolutionY property, ScrollBars property, ScrollPositionX property, ScrollPositionY property, Zoom property (Image Edit).

Refresh Example — VB

This example shows how to let a user select an annotation group from a list, hide it, and then refresh the Image Edit control. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
Private Sub cmdHideGroup_Click()
    Dim strCurGroup As String
    'Determine which group the user selected from the listbox.
    strCurGroup = AnnoGroups.List(AnnoGroups.ListIndex)
    Form1.ImgEdit1.HideAnnotationGroup (strCurGroup)
    'Call refresh to repaint with the group hidden.
    Form1.ImgEdit1.Refresh
End Sub
```

Refresh Example — VC++

This example shows how to let a user select an annotation group from a list, hide it, and then refresh the Image Edit control. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
void CFrmGroup::OnHideGroup()
{
    // This would allow a user to select a Group from a list and hide it. See
    // example for GetAnnotationGroup method for how to list annotation
    // groups.
    CString szCurGroup;
    // Get the name of the group the user clicked on in the listbox.
    m_GroupList.GetText(m_GroupList.GetCurSel(),szCurGroup);
    if(pParentDlg)
    {
      VARIANT vCurGroup; V_VT(&vCurGroup) = VT_BSTR; V_BSTR(&vCurGroup) =
      ➥    szCurGroup.AllocSysString();
      pParentDlg->ImgEdit1.HideAnnotationGroup (vCurGroup);
      // Call refresh to repaint with the group hidden.
      pParentDlg->ImgEdit1.Refresh();
    }
}
```

RemoveAllOCRMarks Method

Description Removes all OCR zones from all image pages in the displayed image document.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**RemoveAllOCRMarks**

Arguments None.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The RemoveAllOCRMarks method saves the displayed image to disk. As such, you cannot use the **Undo** method to undo the operation performed.

Invoking the RemoveAllOCRMarks method sets the **ImageModified** property to True.

See Also Display method, Image OCR control, Image property, ImageModified property, Save method, SaveAs method, SavePage method, Undo method.

Rotate Method

Description Displays a dialog box that lets end users rotate one or all of the pages in a displayed image document. End users can specify the degree of rotation applied as well as the direction.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

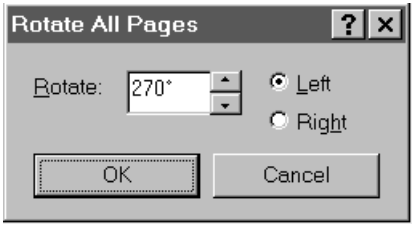
Imaging for Windows NT 4.0

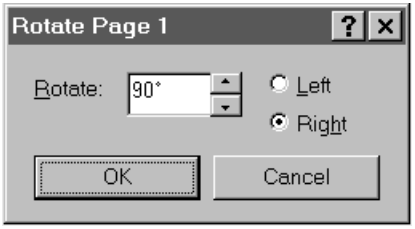
Applies To Image Edit control.

Usage *object*.**Rotate** *Option*

Arguments The Rotate method has the following argument:

Parameter	Data Type	Setting
Option	Boolean	True — Displays the Rotate All Pages dialog box (see below); rotates all pages in the displayed image document as directed by the user. False — Displays the Rotate Page dialog box (see below); rotates the displayed image page as directed by the user.





Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

This method rotates annotations only when 90, 180, or 270 degrees is specified.

When set to True, the Rotate method saves the image document to disk. As such, you cannot use the **Undo** method to undo the rotation performed.

When set to False, the Rotate method modifies the displayed image, but does not save it. In this case, you can use the Undo method to undo the rotation performed. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the Rotate method sets the **ImageModified** property to True.

See Also Display method, Flip method, ImageModified property, RotateAll method, RotateLeft method, RotateRight method, Save method, SaveAs method, SavePage method, Undo method.

Rotate Example — VB

This example shows several ways to orient an image, and then redisplay it.

```
Private Sub cmdRotate_Click()
    ImgEdit1.Rotate True 'rotate all pages
    ImgEdit1.Rotate False 'rotate current page
    'Must set AutoRefresh to true or must call Refresh after these methods.
    ImgEdit1.AutoRefresh = True
    ImgEdit1.RotateLeft 'defaults to 90 degrees
    ImgEdit1.RotateRight 45 'optionally can specify degrees
    ImgEdit1.Flip '180 degrees
    'You must save the image to maintain orientation.
End Sub
```

Rotate Example — VC++

This example shows several ways to orient an image, and then redisplay it.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically. Rotate all
    // writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

RotateAll Method

Description Rotates all pages within the displayed image document to the degree and in the direction specified. Then saves the image document.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**RotateAll** [*Degrees*]

Arguments The RotateAll method has the following argument:

Parameter	Data Type	Setting
Degrees	Long	Amount to rotate the displayed image in tenths of a degree (for example, enter 450 to rotate the image 45 degrees). Enter positive values for counter-clockwise rotation, negative values for clockwise rotation.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

This method saves the image document to disk. As such, you cannot use the **Undo** method to undo the rotation performed.

This method rotates annotations only when 900, 1800, or 2700 tenths of a degree (90, 180, or 270 degrees) is specified (positive or negative).

See Also Display method, Flip method, Rotate method, RotateLeft method, RotateRight method, Undo method.

RotateAll Example — VB

This example shows several ways to orient an image, and then redisplay it.

```
Private Sub cmdRotate_Click()
    ImgEdit1.Rotate True 'rotate all pages
    ImgEdit1.Rotate False 'rotate current page
    'Autorefresh must be set to true or must call Refresh after these
    'methods.
    ImgEdit1.AutoRefresh = True
    ImgEdit1.RotateLeft 'defaults to 90 degrees
    ImgEdit1.RotateRight 45 'optionally can specify degrees
    ImgEdit1.RotateAll 900  'rotate all pages 90 degrees left
```

```
    ImgEdit1.Flip '180 degrees
    'You must save the image to maintain orientation.
End Sub
```

RotateAll Example — VC++

This example shows several ways to orient an image, and then redisplay it.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically. Rotate all
    // writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    VARIANT vRotateAllAmt; V_VT(&vRotateAllAmt) = VT_I4;
    ➥    V_I4(&vRotateAllAmt) = 900;
    ImgEdit1.RotateAll(vRotateAllAmt); //rotate all pages 90 degrees left
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

RotateLeft Method

Description Rotates the displayed image 90 degrees to the left, or to the degree specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**RotateLeft** [*Degrees*]

Arguments The RotateLeft method has the following argument:

Parameter	Data Type	Setting
Degrees	Long	Amount to rotate the displayed image in tenths of a degree (for example, enter 450 to rotate the image 45 degrees).

Note: The Degrees argument is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0.
It is also available with Imaging for Windows 98, but accepts only the following values: 900, 1800, and 2700 tenths of a degree.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

This method rotates annotations only when 900, 1800, or 2700 tenths of a degree (90, 180, or 270 degrees) is specified.

Invoking the RotateLeft method modifies the displayed image. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the RotateLeft method sets the **ImageModified** property to True.

See Also Display method, Flip method, ImageModified property, Rotate method, RotateAll method, RotateRight method, Save method, SaveAs method, SavePage method.

RotateLeft Example — VB

This example shows several ways to orient an image, and then redisplay it.

```
Private Sub cmdRotate_Click()
    ImgEdit1.Rotate True 'rotate all pages
    ImgEdit1.Rotate False 'rotate current page
    'Autorefresh must be set to true or must call Refresh after these
    'methods.
    ImgEdit1.AutoRefresh = True
    ImgEdit1.RotateLeft 'defaults to 90 degrees
    ImgEdit1.RotateRight 45 'optionally can specify degrees
    ImgEdit1.Flip '180 degrees
    'You must save the image to maintain orientation.
End Sub
```

RotateLeft Example — VC++

This example shows several ways to orient an image, and then redisplay it.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically. Rotate all
    // writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
```

```
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

RotateRight Method

Description Rotates the displayed image 90 degrees to the right, or to the degree specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**RotateRight** [*Degrees*]

Arguments The RotateRight method has the following argument:

Parameter	Data Type	Setting
Degrees	Long	Amount to rotate the displayed image in tenths of a degree (for example, enter 450 to rotate the image 45 degrees)

Note: The Degrees argument is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0.
It is also available with Imaging for Windows 98, but accepts only the following values: 900, 1800, and 2700 tenths of a degree.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

This method rotates annotations only when 900, 1800, or 2700 tenths of a degree (90, 180, or 270 degrees) is specified.

Invoking the RotateRight method modifies the displayed image. The **Save**, **SaveAs**, or **SavePage** method must be invoked to save the modified image.

Invoking the RotateRight method sets the **ImageModified** property to True.

See Also Display method, Flip method, ImageModified property, Rotate method, RotateAll method, RotateLeft method, Save method, SaveAs method, SavePage method.

RotateRight Example — VB

This example shows several ways to orient an image, and then redisplay it.

```
Private Sub cmdRotate_Click()
    ImgEdit1.Rotate True 'rotate all pages
    ImgEdit1.Rotate False 'rotate current page
    'Autorefresh must be set to true or must call Refresh after these
    'methods.
    ImgEdit1.AutoRefresh = True
    ImgEdit1.RotateLeft 'defaults to 90 degrees
    ImgEdit1.RotateRight 45 'optionally can specify degrees
    ImgEdit1.Flip '180 degrees
    'You must save the image to maintain orientation.
End Sub
```

RotateRight Example — VC++

This example shows several ways to orient an image, and then redisplay it.

```
void CImgEdit1Dlg::OnRotate()
{
    // An example of some of the different ways an image could be oriented.
    // This method displays a dialog box which allows user to specify
    // rotation by degrees. Repaints the screen automatically. Rotate all
    // writes the file to disk.
    ImgEdit1.Rotate(TRUE); // rotate all pages
    ImgEdit1.Rotate(FALSE); // rotate current page
    // Autorefresh must be set to true or must call Refresh after these
    // methods.
    ImgEdit1.SetAutoRefresh(TRUE);
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.RotateLeft(evt); // defaults to 90 degrees
    VARIANT vRotateAmt; V_VT(&vRotateAmt) = VT_I4; V_I4(&vRotateAmt) = 45;
    ImgEdit1.RotateRight(vRotateAmt); // optionally can specify degrees
    ImgEdit1.Flip(); // 180 degrees
    // You must save the image to maintain orientation.
}
```

Save Method

Description Saves the displayed image (page) to the original path and file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Save** [*SaveAtZoom*]

Arguments The Save method has the following argument:

Parameter	Data Type	Setting
SaveAtZoom	Boolean	Whether the image is saved using the current or original scale percentage value: True — The image is saved using the current scale percentage value, as specified in the Zoom property. False — The image is saved using the original scale percentage value (Zoom value of 100 percent).

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

For the TIFF file type, annotations are saved as annotation data. For all other file types, annotations are lost unless the **BurnInAnnotations** method is invoked prior to calling the Save method. The BurnInAnnotations method burns annotations onto an image, causing them to be permanently incorporated into the image.

The Save method saves the image to the same file from which it was opened. The saved image remains in the display window at its original resolution.

If the SaveAtZoom parameter is not specified, the image is saved using the original zoom scale percentage value.

Invoking the Save method sets the **ImageModified** property to False.

Note: The Save method does not function when a URL has been entered into the **Image** property.

See Also BurnInAnnotations method, Close event, Display method, Image property, ImageModified property, SaveAs method, SavePage method, Zoom property (Image Edit).

Save Example — VB

This example uses the ImageModified property to see if changes were made to an existing image before permitting the user to open another. If the value of the ImageModified property is True, the code prompts the user to save the image before opening the next one.

If the user answers the prompt with Yes, this example uses the Save method to save the existing image.

```
Private Sub ImgEdit1_Close()
    'This code could be used at the time a user attempts to open a new file
    'or clear a previously displayed image in order to prompt for changes to
    'be saved if the image has been modified.
    Dim intSaveImg As Integer
```

```
    If ImgEdit1.ImageModified = True Then
      intSaveImg = MsgBox("Do you want to save changes?", vbYesNoCancel,
      ➥ "Image has been modified")
      Select Case intSaveImg
        Case vbYes
          'Save the changes
          ImgEdit1.Save
          'Drop down to the Open file code
        Case vbNo
          'Don't save file, just drop down to the Open file code
        Case vbCancel
          'User pressed cancel. Don't open a file, don't save changes
          Exit Sub
      End Select
    End If
    'Open a new file at this point
End Sub
```

Save Example — VC++

This example uses the ImageModified property to see if changes were made to an existing image before permitting the user to open another. If the value of the ImageModified property is True, the code prompts the user to save the image before opening the next one.

If the user answers the prompt with Yes, this example uses the Save method to save the existing image.

```
void CImgEdit1Dlg::OnCloseEditctrl1()
{
    // This code could be used at the time a user attempts to open a new file
    // or clear a previously displayed image in order to prompt for changes
    // to be saved if the image has been modified.
    int iSaveImg;
    if(ImgEdit1.GetImageModified())
    {
      iSaveImg = AfxMessageBox("Do you want to save changes?",
      ➥  MB_YESNOCANCEL);
      switch (iSaveImg)
      {
        case IDYES:
          // Save the changes.
          VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default for save at
                                              // zoom
          ImgEdit1.Save(evt);
          break;
          // Drop down to the Open file code.
        case IDNO:
          // Don't save file, just drop down to the Open file code.
          break;
        case IDCANCEL:
          // User pressed cancel. Don't open a file, don't save changes.
          return;
```

```
            }
        }
        // Open a new file at this point.
    }
```

SaveAnnotations Method

Description Saves the annotations on the image currently being displayed to a file.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SaveAnnotations** *FileName*,*StartPage*,*EndPage*,*Flags*

Arguments The SaveAnnotations method has the following arguments:

Parameter	Data Type	Setting
FileName	String	The fully-qualified name (see page 535) of the file to which the annotations will be saved.
StartPage	Long	The starting page in the currently displayed file containing the annotations to be saved.
EndPage	Long	The ending page in the currently displayed file containing the annotations to be saved.
Flags	Long	KEEP_ANNOTATIONS — Retain the annotations in the currently displayed file. REMOVE_ANNOTATIONS — Remove the annotations from the currently displayed file that have been saved to the file specified in the FileName parameter.

Returns Long.

Value	Description
0	Failed
1	Success

Remarks The **Display** method must be invoked prior to calling this method.

To support StartPage and EndPage parameter entries that span more than one page, the image file on display must be a multipage image file type (AWD or TIFF).

Note: The SaveAnnotations method does not function when a URL has been entered into the **Image** property.

Notes Regarding the StartPage and EndPage Parameters

If the StartPage and EndPage parameters are set to the first and last pages of the source image file (currently on display), the annotations in the source file will be saved to the same page numbers in the destination file (as specified in the FileName parameter). For example, if annotations exist on pages 1, 2, 4, and 7 in the source file, they appear on pages 1, 2, 4, and 7 in the destination file.

If the StartPage parameter is set to 5 and the EndPage parameter is set to 8, the annotations on pages 5, 6, 7 and 8 of the source image file will be saved to the destination file specified in the FileName parameter.

The SaveAnnotations method does not retain the page numbers in the source image file. Annotations are saved to the destination file corresponding to page numbers that begin with 1. Continuing the previous example, annotations that appear on pages 5, 6, 7, and 8 in the source file are saved to pages 1, 2, 3, and 4 in the destination file. If there are no annotations on a page in the source file, a blank page is retained in the destination file.

If there are more annotations than pages, or more pages than annotations, the SaveAnnotations method will perform the best fit it can.

See Also Display method, Image property, LoadAnnotations method.

SaveAnnotations Example — VB

This example uses the SaveAnnotations method to save annotations to a file that is separate from the image file.

```
Private Sub cmdSaveAnnos_Click()
    'This example enables annotations to be saved to a file and stored
    'separately from the image file. It is useful if you want to annotate
    'images but not modify the original file (for legal purposes, or if the
    'image is on optical disk). Use in conjunction with the LoadAnnotations
    'method. Save the annotation marks on page 1 of the displayed file to a
    'new file and remove them from the existing file.
    ImgEdit1.SaveAnnotations "D:\template\savemark.ano", 1, 1,
    ➥    REMOVE_ANNOTATIONS
End Sub
```

SaveAnnotations Example — VC++

This example uses the SaveAnnotations method to save annotations to a file that is separate from the image file.

```
void CImgEdit2Dlg::OnSaveanotation()
```

```
{
    // This example would enable annotations to be saved to a file and
    // stored separately from the image file. Useful if you want to annotate
    // images but not modify the original file (ex. for legal purposes, or
    // if the image is on optical disk). Use in conjunction with
    // LoadAnnotations.
    // Save the annotation marks on page 1 of the displayed file to a new
    // file and remove them from the existing file.
    #define         KEEP_ANNOTATIONS     0
    #define         REMOVE_ANNOTATIONS   1
    ImgEdit1.SaveAnnotations ("c:\\savemark.ano", 1, 1,
    ➥    REMOVE_ANNOTATIONS);
}
```

SaveAs Method

Description Saves the displayed image document file or managed image document using the fully-qualified name (see page 535) specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SaveAs** *Image*[,*FileType*,*PageType*,*CompressionType*, *CompressionInfo*,*SaveAtZoom*]

Arguments The SaveAs method has the following arguments:

Parameter	Data Type	Setting
Image	String	The fully-qualified name (see page 535) of the image document being saved.
FileType	Integer (enumerated)	The FileType (see page 530) the image is being saved as.
PageType	Integer (enumerated)	The PageType (see page 529) the image is being saved as.
CompressionType	Integer (enumerated)	The CompressionType (see page 531) at which the image is being saved.
CompressionInfo	Integer (enumerated)	The CompressionInfo setting (see page 531) of the image being saved.

Parameter	Data Type	Setting
SaveAtZoom	Boolean	Whether the image is being saved using the current or original scale percentage value: True — The image is being saved using the current scale percentage value. False — The image is being saved using the original scale percentage value.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The SaveAs method saves the image using the name specified in the Image parameter. If the image document already exists, it is overwritten.

Upon successful completion of the SaveAs method, the image display window continues to display the original image; the **Image** property value remains unchanged. Continuing to display the original image ensures that the image on display is the same as the image specified in the Image property. For example, assume the currently displayed image is **payroll.tif** and the value of the Image property is also **payroll.tif**. When the SaveAs method is invoked with its Image parameter set to **payroll98.tif**, the image is saved as **payroll98.tif**. The image display window continues to display **payroll.tif**, and the Image property value remains unchanged at **payroll.tif**.

If a multipage image file (AWD, DCX, TIFF, WIFF, or XIF) is saved as a file type that does not support multipage image files (for example, BMP), only the page currently being displayed is saved to the new file. Note that this restriction does not apply to Imaging 1x documents. The entire document is saved in the original file type — except for the displayed image, which is converted.

For the TIFF file type, annotations are saved as annotation data separate from the image data. For all other file types, annotations are lost unless the **BurnInAnnotations** method is invoked prior to calling the SaveAs method. The BurnInAnnotations method burns annotations onto an image, causing them to be permanently incorporated into the image.

In addition, document properties are saved within TIFF images only.

If the FileType parameter is not entered, the SaveAs method uses the original FileType setting of the image as a default. If the original FileType setting cannot be determined or is read-only, and the SaveAs method is saving to a local drive or file server, the method throws an Invalid File Format error. If the SaveAs method is saving to a document server, it saves the image as a TIFF file.

If the PageType and FileType parameters are not entered, the SaveAs method uses the original PageType setting of the image as a default. However, if a FileType is entered, the

SaveAs method uses a PageType default setting that is associated with the FileType setting of the image. The following list describes the default PageType settings:

FileType Setting	Default PageType Setting
1 — TIFF	PageType setting of the displayed image.
2 — AWD	1 — Black-and-white.
3 — BMP	PageType setting of the displayed image.

If the CompressionType parameter is Group3(2D), the SaveAs method saves the image using the Group4(2D) compression type. If the CompressionType parameter is LZW, the SaveAs method saves the image using no compression.

If the CompressionType parameter is not entered, the SaveAs method uses a default CompressionType setting that is associated with the FileType and PageType settings of the image. The following list describes the default CompressionType settings:

FileType Setting	PageType Setting	Default CompressionType Setting
2 — AWD	1 — Black-and-white	1 — No compression
3 — BMP	1,5,7 — All supported PageTypes	1 — No compression
1 — TIFF	1 — Black-and-white	2 — Group3(1D)
1 — TIFF	2 — Gray4	1 — No compression
1 — TIFF	3 — Gray8	1 — No compression
1 — TIFF	4 — Palettized4	1 — No compression
1 — TIFF	5 — Palettized8	1 — No compression
1 — TIFF	6 — RGB24	1 — No compression

Note: AWD is not available with Imaging for Windows NT 4.0.

To use a PageType setting of 4 (Palettized4), the displayed image must already be a Palettized4 image.

If the CompressionInfo parameter is not entered, the SaveAs method uses a default CompressionInfo setting that is associated with the CompressionType setting of the image. The following list describes the default CompressionInfo settings:

CompressionType Setting	CompressionInfo Default Setting
1 — No compression	0 — No compression
2 — Group3(1D)	8 — Compressed LTR, 16 — Expand LTR, 1 — EOL, and 4 — Prefixed EOL
3 — Group3(Modified Huffman)	8 — Compressed LTR and 16 — Expand LTR
4 — PackBits	16 — Expand LTR
5 — Group4(2D)	2 — Packed Lines, 8 — Compressed LTR, and 16 — Expand LTR
6 — JPEG	1024 — Medium Resolution/Medium Quality
9 — LZW	0 — No compression

When a CompressionInfo value of 0 is entered with the CompressionType parameter set to 1 (no compression), the file is saved using no compression.

When a CompressionInfo value of 0 is entered with the CompressionType parameter set to a value other than 1, the file is saved using the default CompressionInfo parameter settings that correspond to the setting of the CompressionType parameter (as explained in the preceding list).

For example, if the end user specifies a CompressionType value of 1 (no compression) and then a CompressionInfo value of 0, the file is saved using no compression. If the end user specifies a CompressionType value of 2 (Group3[1D]) and then a CompressionInfo value of 0, the file is saved using the default CompressionInfo values for Group3(1D):

8 — Compressed LTR,
16 — Expand LTR,
1 — EOL, and
4 — Prefixed EOL

If the SaveAtZoom parameter is not specified, the image is saved using the original zoom scale percentage value.

Invoking the SaveAs method sets the **ImageModified** property to False.

Special Note Regarding Imaging 1.x/3.x Documents and Files

If you want your program to process Imaging 1.x managed documents or files, you must include an Image Admin control in your project and set its **FileStgLoc1x** property to the appropriate value.

When saving existing Imaging 1.x managed documents as new Imaging 1.x managed documents, you can use the **ForceFileLinking1x** property of the Image Admin control to determine whether the SaveAs method creates copies of the individual image files in the file repository, or creates links to them. When creating copies of the individual image files, you can use the **FileStgLoc1x** property of the Image Admin control to determine the location of the file repository.

You cannot save image documents to Imaging 3.x Servers.

Imaging 1.x and 3.x Server access is available with Imaging for Windows Professional Edition V1.1 and V2.0 only.

Special Note Regarding URLs

The SaveAs method does not function when a URL has been entered into the **Image** property.

See Also BadDocumentFileType event, BurnInAnnotations method, Close event, CompressionInfo property, CompressionType property, Display method, FileStgLoc1x property, FileType property, ForceFileLinking1x property, Image property, ImageModified property, PageType property, Save method, SavePage method.

SaveAs Example — VB

Example 1

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
Private Sub cmdBurnIn_Click()
    Const ALL_MARKS = 0
    Const WINDOWS_DEFAULT = 2
    'If you wish to save color annotations on a black and white image you
    'must first convert the image to a non black and white type.
    ImgEdit1.ConvertPageType wiPageTypePal8, True
    'Save all marks in their original colors.
    ImgEdit1.BurnInAnnotations ALL_MARKS, WINDOWS_DEFAULT
    'Color Burn in always converts to RGB24. To save disk space, save the
    'image as an 8 bit palettized image.
    ImgEdit1.SaveAs "C:\imgsave\annotate.tif", wiFileTypeTIFF,
    ➥   wiPageTypePal8
End Sub
```

Example 2

This example shows the options that are available for file linking/copying when performing a SaveAs of an Imaging 1x server document.

```
Private Sub cmdLink_Click()
    'An example of the options available for file linking/copying when
    'performing a SaveAs of a 1x server document. These functions require
    'both ImgEdit and Admin controls to be utilized.
'Option 1
    'Request files associated with the document being saved be copied to new
    'files in a location to be specified. This is the default and in most
    'cases the preferred method to use.
    ImgAdmin1.ForceFileLinking1x = False
    'To specify location of the new files, use the Admin control
    'FileStgLoc1x property.
    ImgAdmin1.FileStgLoc1x = "Image://srvrname\new_images:"
    'Displayed 1x document is saved to a new document name Associated image
    'files will be copied to new randomly generated file names in directory
    'specified.
    ImgEdit1.SaveAs "Image://srvrname\doc_db:\CABNAME\DRAWER\FOLDER
    ➥  \NEW DOC"
'Option 2
    'Request files associated with the document being saved reference the
    'original files. This results in multiple documents accessing the same
    'files -- could be useful to prevent rewrites of optical disk files.
    ImgAdmin1.ForceFileLinking1x = True
    'Displayed 1x document is saved to a new document name
    'but original image files are linked to the new document.
    ImgEdit1.SaveAs "Image://srvrname\doc_db:\CABNAME\DRAWER\FOLDER
    ➥  \NEW DOC"
End Sub
```

SaveAs Example — VC++

Example 1

This example shows how to convert an image, burn in annotation marks, and save the image under a new file name.

```
void CImgEdit1Dlg::OnBurnin()
{
    // Converts an image, burns in annotation marks and saves it under a new
    // file name.
    #define ALL_MARKS  0
    #define WINDOWS_DEFAULT  2
    // If you wish to save color annotations on a black and white image you
    // must first convert the image to a non black and white type.
    VARIANT vRepaint; V_VT(&vRepaint) = VT_BOOL; V_BOOL(&vRepaint) = TRUE;
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.ConvertPageType(5,vRepaint);  // wiPageTypePal8, True
    // Save all marks in their original colors.
    ImgEdit1.BurnInAnnotations(ALL_MARKS, WINDOWS_DEFAULT,evt);
```

```
    // Color Burn in always converts to RGB24. To save disk space, save the
    // image as an 8 bit palettized image.
    VARIANT vFileType; V_VT(&vFileType) = VT_I2; // set FileType for saveas
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set to PageType for
                                                 // saveas
    V_I2(&vFileType) = 1;
    V_I2(&vPageType) = 5;
    // wiFileTypeTIFF, wiPageTypePal8
    ImgEdit1.SaveAs("C:\\imgsave\\annotate.tif",vFileType,vPageType,evt,
    ➥  evt,evt);
}
```

Example 2

This example shows the options that are available for file linking/copying when performing a SaveAs of an Imaging 1x server document.

```
void CImgEdit2Dlg::OnFilelinking()
{
    // An example of the options available for file linking/copying
    // when performing a SaveAs of a 1x server document.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default for saveas
// Option 1
    // Request files associated with the document being saved be copied to
    // new files in a location to be specified. This is the default and in
    // most cases the preferred method to use.
    ImgAdmin1.SetForceFileLinking1x(FALSE);
    // To specify location of the new files, use the Admin control
    // FileStgLoc1x property.
    ImgAdmin1.SetFileStgLoc1x("Image://servername\\new_images:");
    // Displayed 1x document is saved to a new document name Associated
    // image files will be copied to new randomly generated file names in
    // directory specified.
    ImgEdit1.SaveAs ("Image://servername\\doc_db:\\CABNAME\\DRAWER
    ➥   \\FOLDER\\NEW DOC",evt,evt,evt,evt,evt);
// Option 2
    // Request files associated with the document being saved reference the
    // original files. This results in multiple documents accessing the same
    // files -- could be useful to prevent rewrites of optical disk files.
    ImgAdmin1.SetForceFileLinking1x(TRUE);
    // Displayed 1x document is saved to a new document name
    // but original image files are linked to the new document.
    ImgEdit1.SaveAs ("Image://servername\\doc_db:\\CABNAME\\DRAWER
    ➥   \\FOLDER\\NEW DOC", evt,evt,evt,evt,evt);
}
```

SavePage Method

Description Saves the displayed image page or the entire image document using the fully-qualified name (see page 535) specified.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage

```
object.SavePage Image[,FileType,PageType,CompressionType,
                      CompressionInfo,SaveAtZoom,PageNumber]
```

Arguments The SavePage method has the following arguments:

Parameter	Data Type	Setting
Image	String	The fully-qualified name (see page 535) of the image document being saved.
FileType	Integer (enumerated)	The FileType (see page 530) the image is being saved as.
PageType	Integer (enumerated)	The PageType (see page 529) the image is being saved as.
CompressionType	Integer (enumerated)	The CompressionType (see page 531) at which the image is being saved.
CompressionInfo	Integer (enumerated)	The CompressionInfo (see page 531) setting of the image being saved.
SaveAtZoom	Boolean	Whether the image is being saved using the current or original scale percentage value: True — The image is being saved using the current scale percentage value. False — The image is being saved using the original scale percentage value.
PageNumber	Long	1 or greater — Saves the displayed image page to the name and PageNumber specified. -1 — Saves all pages in the displayed image document to the name specified. The default is -1. See “Special Note Regarding the PageNumber Parameter” for more information.

Remarks None.

Special Note Regarding the PageNumber Parameter

The following table describes what occurs when the SavePage method is used to save image page(s) to an existing image document.

PageNumber Value Entered	Image Page(s) to be Saved by the SavePage Method	Action Performed by the SavePage Method on the Existing Image Document
1 or greater	Image page currently being displayed	Overwrites the image page that corresponds to the value entered in the PageNumber parameter — provided the image page exists. If the image page does not exist, appends the displayed image page to the end of the existing image document.
-1	All image pages in the image document currently being displayed	Overwrites the entire image document; no existing image pages are retained.

Remarks The **Display** method must be invoked prior to calling this method.

You see an error message if you enter a PageNumber value of 0, or a PageNumber value less than -1. You also see an error message if you enter a PageNumber value greater than one when saving to a new image document.

Upon successful completion of the SavePage method, the image display window continues to display the original image; the **Image** property value remains unchanged. Continuing to display the original image ensures that the image on display is the same as the image specified in the Image property. For example, assume the currently displayed image is **payroll.tif** and the value of the Image property is also **payroll.tif**. When the SavePage method is invoked with its Image parameter set to **payroll98.tif**, the image is saved as **payroll98.tif**. The image display window continues to display **payroll.tif**, and the Image property value remains unchanged at **payroll.tif**.

If a multipage image file (AWD, DCX, TIFF, WIFF, or XIF) is saved as a file type that does not support multipage image files (for example, BMP), only the page currently being displayed is saved to the new file. Note that this restriction does not apply to Imaging 1x documents. The entire document is saved in the original file type — except for the displayed image, which is converted.

For the TIFF file type, annotations are saved as annotation data separate from the image data. For all other file types, annotations are lost unless the **BurnInAnnotations** method is invoked prior to calling the SavePage method. The BurnInAnnotations method burns annotations onto an image, causing them to be permanently incorporated into the image.

In addition, document properties are saved within TIFF images only.

If the FileType parameter is not entered, the SavePage method uses the original FileType setting of the image as a default. If the original FileType setting cannot be determined or is read-only, and the SavePage method is saving to a local drive or file server, the method throws an Invalid File Format error. If the SavePage method is saving to a document server, it saves the image as a TIFF file.

If the PageType and FileType parameters are not entered, the SavePage method uses the original PageType setting of the image as a default. However, if a FileType is entered, the SavePage method uses a PageType default setting that is associated with the FileType setting of the image. The following list describes the default PageType settings:

FileType Setting	Default PageType Setting
1 — TIFF	PageType setting of the displayed image
2 — AWD	1 — Black-and-white
3 — BMP	PageType setting of the displayed image

If the CompressionType parameter is Group3(2D), the SavePage method saves the image using the Group4(2D) compression type. If the CompressionType parameter is LZW, the SavePage method saves the image using no compression.

If the CompressionType parameter is not entered, the SavePage method uses a default CompressionType setting that is associated with the FileType and PageType settings of the image. The following list describes the default CompressionType settings:

FileType Setting	PageType Setting	Default CompressionType Setting
2 — AWD	1 — Black-and-white	1 — No compression
3 — BMP	1,5,7 — All supported PageTypes	1 — No compression
1 — TIFF	1 — Black-and-white	2 — Group3(1D)
1 — TIFF	2 — Gray4	1 — No compression
1 — TIFF	3 — Gray8	1 — No compression
1 — TIFF	4 — Palettized4	1 — No compression
1 — TIFF	5 — Palettized8	1 — No compression
1 — TIFF	6 — RGB24	1 — No compression

To use a PageType setting of 4 (Palettized4), the displayed image must already be a Palettized4 image.

Note: AWD is not available with Imaging for Windows NT 4.0.

If the CompressionInfo parameter is not entered, the SavePage method uses a default CompressionInfo setting that is associated with the CompressionType setting of the image. The following list describes the default CompressionInfo settings:

CompressionType Setting	CompressionInfo Default Setting
1 — No compression	0 — No compression
2 — Group3(1D)	8 — Compressed LTR, 16 — Expand LTR, 1 — EOL, and 4 — Prefixed EOL
3 — Group3(Modified Huffman)	8 — Compressed LTR and 16 — Expand LTR
4 — PackBits	16 — Expand LTR
5 — Group4(2D)	2 — Packed Lines, 8 — Compressed LTR, and 16 — Expand LTR
6 — JPEG	1024 — Medium Resolution/Medium Quality
9 — LZW	0 — No compression

When a CompressionInfo value of 0 is entered with the CompressionType parameter set to 1 (no compression), the file is saved using no compression.

When a CompressionInfo value of 0 is entered with the CompressionType parameter set to a value other than 1, the file is saved using the default CompressionInfo parameter settings that correspond to the setting of the CompressionType parameter (as explained in the preceding list).

For example, if the end user specifies a CompressionType value of 1 (no compression) and then a CompressionInfo value of 0, the file is saved using no compression. If the end user specifies a CompressionType value of 2 (Group3[1D]) and then a CompressionInfo value of 0, the file is saved using the default CompressionInfo values for Group3(1D):

8 — Compressed LTR,
16 — Expand LTR,
1 — EOL, and
4 — Prefixed EOL

If the SaveAtZoom parameter is not specified, the image is saved using the original zoom scale percentage value.

Invoking the SavePage method sets the **ImageModified** property to False.

Special Note Regarding Imaging 1.x/3.x Documents and Files

You cannot save image documents to Imaging 3.x Servers.

Imaging 1.x and 3.x Server access is available with Imaging for Windows Professional Edition V1.1 and V2.0 only.

Special Note Regarding URLs

The SavePage method does not function when a URL has been entered into the **Image** property.

See Also BadDocumentFileType event, BurnInAnnotations method, Close event, CompressionInfo property, CompressionType property, Display method, FileStgLoc1x property, FileType property, ImageModified property, PageType property, Save method, SaveAs method.

SavePage Example — VB

This example shows how to save a page to a new file of the same file type and page type. It also shows how to save the displayed page as a new, uncompressed BMP file at the displayed zoom factor.

```
Private Sub cmdSavePage_Click()
    'Save only the currently displayed page to a new file. Keep file type
    'and page type the same as they were originally. Use defaults for file
    'compression. Save at original zoom factor.
    ImgEdit1.SavePage "D:\image2\savepage.tif", , , , , False, 1
    'Save the currently displayed page to a new uncompressed color BMP file.
    'Save at the displayed zoom factor.
    ImgEdit1.SavePage "D:\image2\saved.bmp", wiFileTypeBMP,
    ➥   wiPageTypeBGR24, 1, 0, True, 1
End Sub
```

SavePage Example — VC++

This example shows how to save a page to a new file of the same file type and page type. It also shows how to save the displayed page as a new, uncompressed BMP file at the displayed zoom factor.

```
void CImgEdit1Dlg::OnSavePage()
{
    // Some examples of the SavePage method
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default for saveas
    VARIANT vFileType; V_VT(&vFileType) = VT_I2; // set FileType for saveas
    VARIANT vPageType; V_VT(&vPageType) = VT_I2; // set PageType for saveas
    // set CompressionType for saveas
    VARIANT vCompresionType; V_VT(&vCompresionType) = VT_I2;
    // set CompressionInfo for saveas
    VARIANT vCompresionInfo; V_VT(&vCompresionInfo) = VT_I2;
    // set SaveAtZoom for saveas
    VARIANT vSaveAtZoom; V_VT(&vSaveAtZoom) = VT_BOOL;
    // set CompressionInfo for saveas
    VARIANT vPageNumber; V_VT(&vPageNumber) = VT_I2;
```

```
    // Save only the currently displayed page to a new file. Keep file type
    // and page type the same as they were originally. Use defaults for file
    // compression. Save at original zoom factor.
    V_BOOL(&vSaveAtZoom) = FALSE;
    V_I2(&vPageNumber) = 1;
    ImgEdit1.SavePage ("D:\\image2\\savepage.tif", evt ,evt ,evt ,evt ,
    ➥   vSaveAtZoom, vPageNumber);
    // Save the currently displayed page to a new uncompressed color BMP
    // file. Save at the displayed zoom factor.
    V_I2(&vFileType) = 1;
    V_I2(&vPageType) = 1;
    V_I2(&vCompresionType) = 1;
    V_I2(&vCompresionInfo) = 0;
    V_BOOL(&vSaveAtZoom) = TRUE;
    V_I2(&vPageNumber) = 1;
    // wiFileTypeBMP, wiPageTypeBGR24, 1, 0, True, 1
    ImgEdit1.SavePage ("D:\\image2\\saved.bmp",vFileType,vPageType,
     ➥   vCompresionType,vCompresionInfo,vSaveAtZoom,vPageNumber);
}
```

ScrollImage Method

Description Scrolls a displayed image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ScrollImage** *Direction*,*ScrollAmount*

Arguments The ScrollImage method has the following arguments:

Parameter	Data Type	Setting
Direction	Integer (enumerated)	The direction in which to scroll the image: 0 — Down 1 — Up 2 — Right 3 — Left
ScrollAmount	Long	The number of pixels to scroll the image.

Remarks The **Display** method must be invoked prior to calling this method.

See Also Display method, Scroll event, ScrollBars property, ScrollPositionX property, ScrollPositionY property, ScrollShortcutsEnabled property.

SelectAnnotationGroup Method

Description Selects all annotation marks within a specific group on the image page.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SelectAnnotationGroup** *GroupName*

Arguments The SelectAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
GroupName	String	The name of the annotation group to select.

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

Selecting annotations enables many methods to perform functions on them.

See Also AddAnnotationGroup method, DeleteSelectedAnnotations method, Display method, GetAnnotationGroup method, GetCurrentAnnotationGroup method, MarkSelect event, SetCurrentAnnotationGroup method, ShowAnnotationGroup method.

SelectFirstOcrZone Method

Description Selects the first OCR zone on the displayed image page.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SelectFirstOcrZone** *ZoneType*

Arguments The SelectFirstOcrZone has the following argument:

Parameter	Data Type	Setting
ZoneType	Integer (enumerated)	0 — Select the first OCR Text zone. 1 — Select the first OCR Picture zone. 2 — Select the first OCR Text or Picture zone.

Returns Long (NextInfo, which is used by the **SelectNextOcrZone** method).

Remarks The **Display** method must be invoked prior to calling this method.

This method works in conjunction with the **SelectNextOcrZone** method.

See Also Display method, Image OCR control, SelectNextOcrZone method.

SelectNextOcrZone Method

Description Selects the next OCR zone on the displayed image page.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SelectNextOcrZone**(*NextInfo*)

Arguments The SelectNextOcrZone method has the following argument:

Parameter	Data Type	Setting
NextInfo	Long	Next information from the last invocation of the **SelectFirstOcrZone** or **SelectNextOcrZone** method.

Returns Long (NextInfo, which is used by the next invocation of this method).

Remarks The **Display** method must be invoked prior to calling this method.

See Also Display method, Image OCR control, SelectFirstOcrZone method.

SelectTool Method

Description Selects an annotation tool from the Annotation Tool Palette (see page 533).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SelectTool** *ToolID*

Arguments The SelectTool method has the following argument:

Parameter	Data Type	Setting	Description
ToolID	Integer (enumerated)	0	No Tool
		1	Selection
		2	Freehand Line
		3	Highlighter
		4	Straight Line
		5	Hollow Rectangle
		6	Filled Rectangle
		7	Text
		8	Attach-a-Note
		9	Text From File
		10	Text Stamp
		11	Hyperlink
		12	OCR Zones

Note: Hyperlink and OCR Zone ToolID settings are available with Imaging for Windows Professional Edition only.

Returns None.

Remarks The Highlighter annotation type is composed of a Filled Rectangle annotation type with a transparent fill style.

When a tool is selected from the tool palette, the previous tool is deselected automatically.

The Annotation Tool Palette does not have to be visible to use this method.

See Also ShowAnnotationToolPalette method, ToolSelected event.

SelectTool Example — VB

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
Private Sub cmdSelectTool_Click()
    'Uses the SelectTool method to select the Text from file annotation
    'type. Overlays text from a text file onto an image. If the annotation
    'tool palette is utilized within an application, SelectTool should be
    'used rather than setting the AnnotationType property for drawing marks
    'programatically.
    ImgEdit1.SelectTool 9 'Text from file
    ImgEdit1.AnnotationTextFile = "C:\text\disclaim.txt"
    ImgEdit1.Draw 20, 20
End Sub
```

SelectTool Example — VC++

This example uses the SelectTool method to select the Text From File annotation type. It then uses the content of the AnnotationTextFile property and the Draw method to overlay text from a text file onto an image.

```
void CImgEdit1Dlg::OnSelectTool()
{
    // Uses the SelectTool method to select the Text from file annotation
    // type. Overlays text from a text file onto an image.
    // If the annotation tool palette is utilized within an application,
    // SelectTool should be used rather than setting the AnnotationType
    // property for drawing marks programatically.
    ImgEdit1.SelectTool(9); // Text from file
    ImgEdit1.SetAnnotationTextFile("C:\\text\\disclaim.txt");
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    ImgEdit1.Draw(10, 10, evt, evt);
}
```

SetCurrentAnnotationGroup Method

Description Sets the annotation group to which subsequent annotations will belong.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetCurrentAnnotationGroup** *GroupName*

Arguments The SetCurrentAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
GroupName	String	The name of the annotation group

Returns None.

Remarks The SetCurrentAnnotationGroup method is valid only at run-time.

The names of the groups can be obtained from the **GetAnnotationGroup** method.

A new group can be created using the **AddAnnotationGroup** method.

See Also AddAnnotationGroup method, BurnInAnnotations method, GetAnnotationGroup method, GetCurrentAnnotationGroup method.

SetImagePalette Method

Description Sets the palette to be used for the image currently being displayed, or an image to be displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetImagePalette** *Option*

Arguments The SetImagePalette method has the following argument:

Parameter	Data Type	Setting	Description
Option	Integer (enumerated)	0 1	Foreground Background

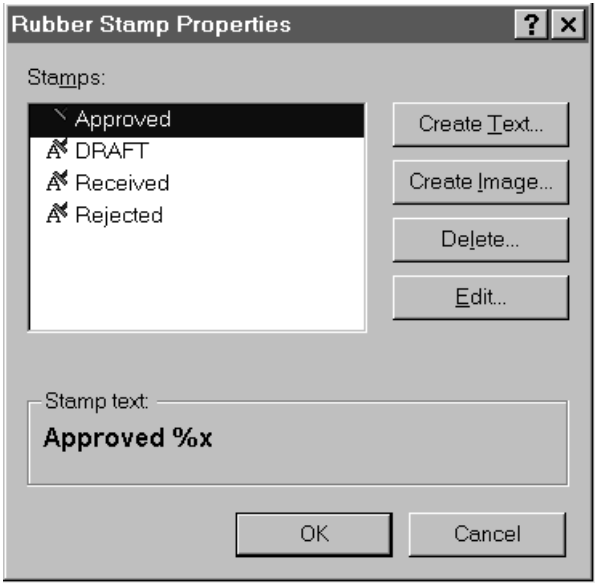
Returns None.

Remarks Images are displayed using the foreground palette by default. When the default foreground palette is used, it is loaded into the Windows palette, which can cause other images or other color-aware objects to become distorted. Setting the SetImagePalette Option value to 1 (Background) sets the existing palette, which has already been loaded by Windows.

See Also ImagePalette property.

SetRubberStampItem Method

Description Sets the item number for the rubber stamp according to its position on the Rubber Stamp Properties dialog box (shown here) and drop-down menu. Also activates the Rubber Stamp annotation.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetRubberStampItem** *ItemNumber*

Arguments The SetRubberStampItem method has the following argument:

Parameter	Data Type	Setting
ItemNumber	Integer	The desired rubber stamp item number

Returns None.

Remarks The item number is zero based. For example, an ItemNumber of 2 selects the Received stamp because it is the third item on the Rubber Stamp Properties dialog box and drop-down menu.

The Rubber Stamp Properties dialog box is accessible by right-clicking on the Rubber Stamp tool in the Annotation Tool Palette (see page 533). The drop-down menu is accessible by left-clicking on the tool.

See Also GetRubberStampItem method, GetRubberStampMenuItems method, ShowAnnotationToolPalette.

SetRubberStampItem Example — VB

This example lets users select the type of rubber stamp they want to use.

```
Private Sub cmdSetStamp_Click()
    'This allows a user to click on a stamp in order to select it as the
    'current stamp. See the GetRubberStampMenuItems method for an example of
    'how to create a list of stamps.
    Dim intNewStamp As Integer
    'Get the index number of the stamp that was selected in the list box.
    intNewStamp = frmStampList.List1.ListIndex
    'Set the selected stamp as the current stamp to be used.
    frmMain.ImgEdit1.SetRubberStampItem intNewStamp
    Unload frmStampList
End Sub
```

SetRubberStampItem Example — VC++

This example lets users select the type of rubber stamp they want to use.

```
void CImgEdit2Dlg::OnSetcurrentstamp()
{
    // This allows a user to click on a stamp in order to select it as
    // the current stamp. See GetRubberStampMenuItems for an example of how
    // to create a list of stamps.
    // Load the list box.
    CStampList frmStampList;
    frmStampList.m_pParent= this;
    frmStampList.DoModal();
    ImgEdit1.SetRubberStampItem(frmStampList.m_intNewStamp);
}
```

SetSelectedAnnotationBackColor Method

Description Sets the background color of a selected Attach-a-Note annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationBackColor** *Color*

Arguments The SetSelectAnnotationBackColor method has the following arguments:

Parameter	Data Type	Setting
Color	Long	The desired Attach-a-Note color

Returns None.

Remarks The SetSelectedAnnotationBackColor method is used with annotations of the Attach-a-Note type.

Use the RGB format (see page 540). To ensure the text in an Attach-a-Note annotation will be visible against the background, specify a background color that is significantly different from the font color set using the **SetSelectedAnnotationFontColor** method.

Invoking the SetSelectedAnnotationBackColor method sets the **ImageModified** property to True.

See Also AnnotationBackColor property, AnnotationType property, GetSelectedAnnotationBackColor method, ImageModified property, SetSelectedAnnotationFontColor method.

SetSelectedAnnotationBackColor Example — VB

This example uses the GetSelectedAnnotationBackColor method, the SetSelectedAnnotationBackColor method, and the Microsoft common dialog box to change the background color of a selected text annotation.

```
Private Sub cmdBackColor_Click()
    'This example uses the Microsoft color dialog box to change
    'a selected text attachment annotation mark's background color.
    Dim BackColor As Long
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    BackColor = ImgEdit1.GetSelectedAnnotationBackColor
    CommonDialog1.Color = BackColor
    CommonDialog1.Flags = cdlCCRGBInit
    'Display the dialog box, and set the annotation background color to the
    'color selected.
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationBackColor CommonDialog1.Color
End Sub
```

SetSelectedAnnotationBackColor Example — VC++

This example uses the GetSelectedAnnotationBackColor method, the SetSelectedAnnotationBackColor method, and the Microsoft common dialog box to change the background color of a selected text annotation.

```
void CImgEdit2Dlg::OnBackcolor()
{
    // This example uses the Microsoft color dialog box to change
    // a selected text attachment annotation mark's background color.
    long BackColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    BackColor = ImgEdit1.GetSelectedAnnotationBackColor();
    CommonDialog1.SetColor(BackColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationBackColor(CommonDialog1.GetColor());
}
```

SetSelectedAnnotationFillColor Method

Description Sets the color used to fill a selected Filled Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationFillColor** *Color*

Arguments The SetSelectedAnnotationFillColor method has the following argument:

Parameter	Data Type	Setting
Color	Long	The desired Filled Rectangle color

Remarks The SetSelectedAnnotationFillColor method is used with annotations of the Filled Rectangle annotation type.

Use the RGB format (see page 540).

Invoking the SetSelectedAnnotationFillColor method sets the **ImageModified** property to True.

See Also AnnotationFillColor property, AnnotationType property, GetSelectedAnnotationFillColor method, ImageModified property, SetSelectedAnnotationFontColor method.

SetSelectedAnnotationFillColor Example — VB

This example uses the GetSelectedAnnotationFillColor method to determine the fill color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFillColor method to change the fill color of a selected annotation.

```
Private Sub cmdFillColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's fill color.
    Dim FillColor As Long
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    FillColor = ImgEdit1.GetSelectedAnnotationFillColor
    CommonDialog1.Color = FillColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationFillColor CommonDialog1.Color
End Sub
```

SetSelectedAnnotationFillColor Example — VC++

This example uses the GetSelectedAnnotationFillColor method to determine the fill color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFillColor method to change the fill color of a selected annotation.

```
void CImgEdit2Dlg::OnFillcolor()
{
    // This example uses the Microsoft color dialog box to
```

```
    // change a selected annotation mark's fill color.
    long FillColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    FillColor = ImgEdit1.GetSelectedAnnotationFillColor();
    CommonDialog1.SetColor(FillColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationFillColor(CommonDialog1.GetColor());
}
```

SetSelectedAnnotationFillStyle Method

Description Sets the style used to fill a selected image or Filled Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationFillStyle** *Style*

Arguments The SetSelectedAnnotationFillStyle method has the following argument:

Parameter	Data Type	Constant	Setting	Description
Style	Integer (enumerated)	**wiTransparent**	0	Transparent — The underlying image data is visible.
		wiOpaque	1	Opaque — The underlying image data is obscured.

Returns None.

Remarks The SetSelectedAnnotationFillStyle method is used with annotations of the following types:

- Filled Rectangle
- Image Embedded
- Image Reference

Invoking the SetSelectedAnnotationFillStyle method sets the **ImageModified** property to True.

See Also AnnotationFillStyle property, AnnotationType property, GetSelectedAnnotationFillStyle method, ImageModified property.

SetSelectedAnnotationFillStyle Example — VB

This example uses the GetSelectedAnnotationFillStyle method to determine whether a selected image or filled rectangle annotation is opaque or transparent, and the SetSelectedAnnotationFillStyle method to change the fill style.

```
    Private Sub Form_Load()
    'This example would determine whether a selected annotation image or
    'filled rectangle is opaque or transparent and would allow that
    'attribute to be changed. The form used would have 2 option buttons --
    'one labeled "Opaque" and one labeled "Transparent".
    Dim FillStyle As AnnotationStyleConstants
    'Determine if the selected image annotation is opaque or transparent and
    'set the corresponding option button on the form.
    FillStyle = frmMain.ImgEdit1.GetSelectedAnnotationFillStyle
    If FillStyle = wiOpaque Then
      optOpaque.Value = True
    Else
      optTransparent.Value = True
    End If
    frmFillStyle.Show
End Sub
Private Sub optOpaque_Click()
     'If the user clicks on the Opaque option button the annotation is
     'changed to be opaque.
     frmMain.ImgEdit1.SetSelectedAnnotationFillStyle wiOpaque
End Sub
Private Sub optTransparent_Click()
     'If the user clicks on the Transparent option button the annotation
     'is changed to be transparent.
     frmMain.ImgEdit1.SetSelectedAnnotationFillStyle wiTransparent
End Sub
```

SetSelectedAnnotationFillStyle Example — VC++

This example uses the GetSelectedAnnotationFillStyle method to determine whether a selected image or filled rectangle annotation is opaque or transparent, and the SetSelectedAnnotationFillStyle method to change the fill style.

```
void CLineStyles::OnOK()
{
    if(m_Transparent.GetCheck() && m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(1);
    if(m_Transparent.GetCheck() && !m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(1);
    CDialog::OnOK();
}
```

```
BOOL CLineStyles::OnInitDialog()
{
    CDialog::OnInitDialog();
    short LineStyle;
    // Determine if the selected line or fill annotation is opaque or
    // transparent and set the corresponding option button on the form.
    if(m_bLineStyle)
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationLineStyle();
    else
     {
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationFillStyle();
      SetWindowText("Fill Styles");
     }
    if(LineStyle == 1)  //  wiOpaque
     {
      m_Opaque.SetCheck(1);
      m_Transparent.SetCheck(0);
     }
    else
     {
      m_Opaque.SetCheck(0);
      m_Transparent.SetCheck(1);
     }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

SetSelectedAnnotationFont Method

Description Sets a font object's properties. Applies to selected text annotations.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationFont** *Font*

Arguments The SetSelectedAnnotationFont method has the following argument:

Parameter	Data Type	Setting
Font	IFontDisp	The properties of the font object

Returns None.

Remarks Use the SetSelectedAnnotationFont method to identify the specific font object whose properties you want to set. The following list shows the properties you can set, along with their data types, descriptions, and defaults:

Property	Data Type	Description	Default
Bold	Boolean	True — Selects bold attribute. False — Deselects bold attribute.	False
Italic	Boolean	True — Selects italic attribute. False — Deselects italic attribute.	False
Name	String	The desired font name.	MS San Serif
Size	Single	The desired font size in points (0 to 2048).	12
Underline	Boolean	True — Selects underline attribute. False — Deselects underline attribute.	False
StrikeThrough	Boolean	True — Selects ~~strikethrough~~ attribute. False — Deselects strikethrough attribute.	False

The SetSelectedAnnotationFont method is used with annotations of the following types:

- Attach-a-Note
- Hyperlink
- Text
- Text From File
- Text Stamp

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

Invoking the SetSelectedAnnotationFont method sets the **ImageModified** property to True.

See Also AnnotationFont property, AnnotationType property, GetSelectedAnnotationFontColor method, ImageModified property.

SetSelectedAnnotationFont Example — VB

This example uses the GetSelectedAnnotationFont method to determine the font attributes of a selected annotation. It then updates the Microsoft common dialog box to indicate the attributes obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFont method to change the font attributes of a selected annotation.

```
Private Sub cmdGetAnnoFont_Click()
    'This example uses the Microsoft font dialog box to
    'change a selected text annotation mark's font.
    Dim AnnoFont As IFontDisp
    Dim FontColor As Long
    'Determine the current font used in the selected annotation
    'mark and init the dialog box font property to that font.
    Set AnnoFont = ImgEdit1.GetSelectedAnnotationFont
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor
    CommonDialog1.FontName = AnnoFont.Name
    CommonDialog1.FontBold = AnnoFont.Bold
    CommonDialog1.FontItalic = AnnoFont.Italic
    CommonDialog1.FontSize = AnnoFont.Size
    CommonDialog1.FontUnderline = AnnoFont.Underline
    CommonDialog1.FontStrikethru = AnnoFont.Strikethrough
    CommonDialog1.Color = FontColor
    CommonDialog1.Flags = cdlCFScreenFonts Or cdlCFEffects
    CommonDialog1.ShowFont
    'Take values set in the font dialog box and set these attributes in the
    'annotation font.
    AnnoFont.Name = CommonDialog1.FontName
    AnnoFont.Bold = CommonDialog1.FontBold
    AnnoFont.Italic = CommonDialog1.FontItalic
    AnnoFont.Size = CommonDialog1.FontSize
    AnnoFont.Underline = CommonDialog1.FontUnderline
    AnnoFont.Strikethrough = CommonDialog1.FontStrikethru
    'Apply the new font and font color to the selected annotation.
    ImgEdit1.SetSelectedAnnotationFont AnnoFont
    ImgEdit1.SetSelectedAnnotationFontColor CommonDialog1.Color
End Sub
```

SetSelectedAnnotationFont Example — VC++

This example uses the GetSelectedAnnotationFont method to determine the font attributes of a selected annotation. It then updates the Microsoft common dialog box to indicate the attributes obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFont method to change the font attributes of a selected annotation.

```
void CImgEdit2Dlg::OnAnnofont()
{
    // This example uses the Microsoft font dialog box to change a selected
    // text annotation mark's font.
```

```
COleFont AnnoFont;
AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
CY cy;
long FontColor;
// Determine the current font used in the selected annotation
// mark and init the dialog box font property to that font.
AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
FontColor = ImgEdit1.GetSelectedAnnotationFontColor();
CommonDialog1.SetFontName(AnnoFont.GetName());
CommonDialog1.SetFontBold(AnnoFont.GetBold());
CommonDialog1.SetFontItalic(AnnoFont.GetItalic());
cy = AnnoFont.GetSize();
CommonDialog1.SetFontSize((float)cy.Lo / 10000);
CommonDialog1.SetFontUnderLine(AnnoFont.GetUnderline());
CommonDialog1.SetFontStrikeThru(AnnoFont.GetStrikethrough());
CommonDialog1.SetColor(FontColor);
CommonDialog1.SetFlags(0x101); //cdlCFScreenFonts Or cdlCFEffects
CommonDialog1.ShowFont();
// Take values set in the font dialog box and set these attributes in
// the annotation font.
AnnoFont.SetName(CommonDialog1.GetFontName());
AnnoFont.SetBold(CommonDialog1.GetFontBold());
AnnoFont.SetItalic(CommonDialog1.GetFontItalic());
cy.Lo = (unsigned long)CommonDialog1.GetFontSize() * 10000;
AnnoFont.SetSize(cy);
AnnoFont.SetUnderline(CommonDialog1.GetFontUnderLine());
AnnoFont.SetStrikethrough(CommonDialog1.GetFontStrikeThru());
// Apply the new font and font color to the selected annotation.
ImgEdit1.SetSelectedAnnotationFont(AnnoFont);
ImgEdit1.SetSelectedAnnotationFontColor(CommonDialog1.GetColor());
```

SetSelectedAnnotationFontColor Method

Description Sets the font color used in a selected text annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationFontColor** *Color*

Arguments The SetSelectedAnnotationFontColor method has the following argument:

Parameter	Data Type	Setting
Color	Long	The desired font color

Returns None.

Remarks The SetSelectedAnnotationFontColor method is used with annotations of the following types:

- Attach-a-Note
- Hyperlink
- Text
- Text From File
- Text Stamp

Note: Hyperlink annotations are available with Imaging for Windows Professional Edition only.

Invoking the SetSelectedAnnotationFontColor method sets the **ImageModified** property to True.

Use the RGB format (see page 540). To ensure that the text in an Attach-a-Note annotation will be visible against the background, specify a font color that is significantly different from the background color set using the **SetSelectedAnnotationBackColor** method.

See Also AnnotationFontColor property, AnnotationType property, GetSelectedAnnotationFontColor method, ImageModified property, SetSelectedAnnotationBackColor method, SetSelectedAnnotationFillColor method.

SetSelectedAnnotationFontColor Example — VB

This example uses the GetSelectedAnnotationFontColor method to determine the font color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFontColor method to change the font color of a selected annotation.

```
Private Sub cmdFontColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's font color.
    Dim FontColor As Long
    'Determine the current color of the selected annotation
    'mark and init the dialog box color property to that color.
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor
    CommonDialog1.Color = FontColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationFontColor CommonDialog1.Color
End Sub
```

SetSelectedAnnotationFontColor Example — VC++

This example uses the GetSelectedAnnotationFontColor method to determine the font color of a selected annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationFontColor method to change the font color of a selected annotation.

```
void CImgEdit2Dlg::OnAnnofont()
{
    // This example uses the Microsoft font dialog box to
    // change a selected text annotation mark's font.
    COleFont AnnoFont;
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    CY cy;
    long FontColor;
    // Determine the current font used in the selected annotation
    // mark and init the dialog box font property to that font.
    AnnoFont = ImgEdit1.GetSelectedAnnotationFont();
    FontColor = ImgEdit1.GetSelectedAnnotationFontColor();
    CommonDialog1.SetFontName(AnnoFont.GetName());
    CommonDialog1.SetFontBold(AnnoFont.GetBold());
    CommonDialog1.SetFontItalic(AnnoFont.GetItalic());
    cy = AnnoFont.GetSize();
    CommonDialog1.SetFontSize((float)cy.Lo / 10000);
    CommonDialog1.SetFontUnderLine(AnnoFont.GetUnderline());
    CommonDialog1.SetFontStrikeThru(AnnoFont.GetStrikethrough());
    CommonDialog1.SetColor(FontColor);
    CommonDialog1.SetFlags(0x101); //cdlCFScreenFonts Or cdlCFEffects
    CommonDialog1.ShowFont();
    // Take values set in the font dialog box and set these attributes in
    // the annotation font.
    AnnoFont.SetName(CommonDialog1.GetFontName());
    AnnoFont.SetBold(CommonDialog1.GetFontBold());
    AnnoFont.SetItalic(CommonDialog1.GetFontItalic());
    cy.Lo = (unsigned long)CommonDialog1.GetFontSize() * 10000;
    AnnoFont.SetSize(cy);
    AnnoFont.SetUnderline(CommonDialog1.GetFontUnderLine());
    AnnoFont.SetStrikethrough(CommonDialog1.GetFontStrikeThru());
    // Apply the new font and font color to the selected annotation.
    ImgEdit1.SetSelectedAnnotationFont(AnnoFont);
    ImgEdit1.SetSelectedAnnotationFontColor(CommonDialog1.GetColor());
}
```

SetSelectedAnnotationLineColor Method

Description Sets the line color used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationLineColor** *Color*

Arguments The SetSelectedAnnotationLineColor method has the following argument:

Parameter	Data Type	Setting
Color	Long	The desired line color

Returns None.

Remarks The SetSelectedAnnotationLineColor method is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

Use the RGB format (see page 540).

Invoking the SetSelectedAnnotationLineColor method sets the **ImageModified** property to True.

See Also AnnotationLineColor property, AnnotationType property, GetSelectedAnnotationLineColor method, ImageModified property.

SetSelectedAnnotationLineColor Example — VB

This example uses the GetSelectedAnnotationLineColor method to determine the color of a selected line annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationLineColor method to change the color of a selected line annotation.

```
Private Sub cmdLineColor_Click()
    'This example uses the Microsoft color dialog box to
    'change a selected annotation mark's line color.
    Dim LineColor As Long
    'Determine the current color of the selected annotation
```

```
    'mark and init the dialog box color property to that color.
    LineColor = ImgEdit1.GetSelectedAnnotationLineColor
    CommonDialog1.Color = LineColor
    CommonDialog1.Flags = cdlCCRGBInit
    CommonDialog1.ShowColor
    ImgEdit1.SetSelectedAnnotationLineColor CommonDialog1.Color
End Sub
```

SetSelectedAnnotationLineColor Example — VC++

This example uses the GetSelectedAnnotationLineColor method to determine the color of a selected line annotation. It then updates the Microsoft common dialog box to indicate the color obtained.

This example also uses the Microsoft common dialog box and the SetSelectedAnnotationLineColor method to change the color of a selected line annotation.

```
void CImgEdit2Dlg::OnLinecolor()
{
    // This example uses the Microsoft color dialog box to
    // change a selected annotation mark's line color.
    long LineColor;
    // Determine the current color of the selected annotation
    // mark and init the dialog box color property to that color.
    LineColor = ImgEdit1.GetSelectedAnnotationLineColor();
    CommonDialog1.SetColor(LineColor);
    CommonDialog1.SetFlags(1); // cdlCCRGBInit
    CommonDialog1.ShowColor();
    ImgEdit1.SetSelectedAnnotationLineColor(CommonDialog1.GetColor());
}
```

SetSelectedAnnotationLineStyle Method

Description Sets the line style used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationLineStyle** *Style*

Arguments The SetSelectedAnnotationLineStyle method has the following argument:

Parameter	Data Type	Constant	Setting	Description
Style	Integer (enumerated)	**wiTransparent**	0	Transparent — The underlying image data is visible.
		wiOpaque	1	Opaque — The underlying image data is obscured.

Returns None.

Remarks The SetSelectedAnnotationLineStyle method is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

Invoking the SetSelectedAnnotationLineStyle method sets the **ImageModified** property to True.

See Also AnnotationLineStyle property, AnnotationType property, GetSelectedAnnotationLineStyle method, ImageModified property.

SetSelectedAnnotationLineStyle Example — VB

This example uses the GetSelectedAnnotationLineStyle method to determine whether a selected line or hollow rectangle annotation is opaque or transparent, and the SetSelectedAnnotationLineStyle method to change the line style.

```
Private Sub Form_Load()
    'This example determines whether a selected annotation line or
    'hollow rectangle is opaque or transparent and allows that
    'attribute to be changed. The form used would has 2 option buttons --
    'one labeled "Opaque" and one labeled "Transparent".
    Dim LineStyle As AnnotationStyleConstants
    'Determine if the selected line annotation is opaque or transparent and
    'set the corresponding option button on the form.
    LineStyle = frmMain.ImgEdit1.GetSelectedAnnotationLineStyle
    If LineStyle = wiOpaque Then
      optOpaque.Value = True
    Else
      optTransparent.Value = True
    End If
    frmLineStyle.Show
End Sub
Private Sub optOpaque_Click()
    'If the user clicks on the Opaque option button the annotation is
```

```
    'changed to be opaque.
    frmMain.ImgEdit1.SetSelectedAnnotationLineStyle wiOpaque
End Sub
 Private Sub optTransparent_Click()
    'If user clicks Transparent button the annotation becomes transparent.
    frmMain.ImgEdit1.SetSelectedAnnotationLineStyle wiTransparent
End Sub
```

SetSelectedAnnotationLineStyle Example — VC++

This example uses the GetSelectedAnnotationLineStyle method to determine whether a selected line or hollow rectangle annotation is opaque or transparent, and the SetSelectedAnnotationLineStyle method to change the line style.

```
void CLineStyles::OnOK()
{
    if(m_Transparent.GetCheck() && m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationLineStyle(1);
    if(m_Transparent.GetCheck() && !m_bLineStyle)
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationFillStyle(1);
    CDialog::OnOK();
}
BOOL CLineStyles::OnInitDialog()
{
    CDialog::OnInitDialog();
    short LineStyle;
    // Determine if the selected line or fill annotation is opaque or
    // transparent and set the corresponding option button on the form.
    if(m_bLineStyle)
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationLineStyle();
    else
     {
      LineStyle = m_pParent->ImgEdit1.GetSelectedAnnotationFillStyle();
      SetWindowText("Fill Styles");
     }
    if(LineStyle == 1)  //  wiOpaque
     {
      m_Opaque.SetCheck(1);
      m_Transparent.SetCheck(0);
     }
    else
     {
      m_Opaque.SetCheck(0);
      m_Transparent.SetCheck(1);
     }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

SetSelectedAnnotationLineWidth Method

Description Sets the line width used in a selected line or Hollow Rectangle annotation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationLineWidth** *Width*

Arguments The SetSelectedAnnotationLineWidth method has the following argument:

Parameter	Data Type	Setting
Width	Integer	The desired line width in pixels

Returns None.

Remarks The SetSelectedAnnotationLineWidth method is used with annotations of the following types:

- Freehand Line
- Hollow Rectangle
- Straight Line

Invoking the SetSelectedAnnotationLineWidth method sets the **ImageModified** property to True.

See Also AnnotationLineWidth property, AnnotationType property, GetSelectedAnnotationLineWidth method, ImageModified property.

SetSelectedAnnotationLineWidth Example — VB

Example 1 allows the user to change the line width value in a text box. It then updates a line control to illustrate the width, and uses the SetSelectedAnnotationLineWidth method to set the width of a selected line annotation to the value entered in the text box control.

Example 2 allows the user to change the line width value using an UpDown control. It then updates a line control to illustrate the width, and uses the SetSelectedAnnotationLineWidth method to set the width of a selected line annotation.

Example 1

```
Private Sub txtWidth_Change()
    'On a form containing a text box and a line control, this allows
    'the user to change the value in the text box and then updates the
    'line control (to illustrate the width) and also updates the width
```

```
    'of a selected line annotation.
    Global intLineWidth as integer
    'Determine the width entered by user in the text box.
    intLineWidth = Int(txtWidth.Text)
    'Set the width of the line control in the dialog box.
    Line1.BorderWidth = intLineWidth
    'Set the width of the line annotation on the displayed image.
    frmMain.ImgEdit1.SetSelectedAnnotationLineWidth intLineWidth
End Sub
```

Example 2

```
Private Sub UpDown1_Change()
    'On a form containing a text box that is linked to an UpDown control and
    'a line control, this allows the user to scroll the UpDown control
    'and changes the value in the text box, the width of the sample
    'line, and also updates the width of a selected line annotation.
    'Determine the line width by the value of the UpDown control.
    intLineWidth = UpDown1.Value
    'Set the width of the "sample" line in the dialog box.
    Line1.BorderWidth = intLineWidth
    'Set the width of the selected annotation line.
    frmMain.ImgEdit1.SetSelectedAnnotationLineWidth intLineWidth
End Sub
```

SetSelectedAnnotationLineWidth Example — VC++

This example uses the GetSelectedAnnotationLineWidth method to determine the width of a selected line annotation. It then displays the line width in a text box control.

It also uses the SetSelectedAnnotationLineWidth method to set the width of a selected line annotation to the value entered in the text box control.

```
BOOL CLineWidth::OnInitDialog()
{
    CDialog::OnInitDialog();
    int intLineWidth;
    long LineColor;
    // Get the line width and color to initialize the dialog box.
    intLineWidth = m_pParent->ImgEdit1.GetSelectedAnnotationLineWidth();
    LineColor = m_pParent->ImgEdit1.GetSelectedAnnotationLineColor();
    // Set the current line width in the text box.
    char tLineWidth [5];
    _itoa(intLineWidth,tLineWidth,10);
    m_Edit.SetWindowText(tLineWidth);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
void CLineWidth::OnOK()
{
    CString tLineWidth;
    int iLineWidth;
    m_Edit.GetWindowText(tLineWidth);
```

```
    iLineWidth = atoi(tLineWidth);
    m_pParent->ImgEdit1.SetSelectedAnnotationLineWidth(iLineWidth);
    CDialog::OnOK();
}
```

SetSelectedAnnotationOcrType Method

Description Changes the OCR type of a selected OCR Zones annotation.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**SetSelectedAnnotationOcrType** *OcrType*

Arguments The SetSelectedAnnotationOcrType method has the following parameter:

Parameter	Data Type	Setting
OcrType	Integer (enumerated)	0 — OCR Text zone 1 — OCR Picture zone

Returns None.

Remarks An OCR Zones annotation type must be selected prior to calling this method.

Invoking the SetSelectedAnnotationOcrType method sets the **ImageModified** property to True.

See Also AnnotationOcrType property, AnnotationType property, GetSelectedAnnotationOcrType method, Image OCR control, ImageModified property, OcrZoneVisibility property, SelectFirstOcrZone method, SelectNextOcrZone method.

SetSelectedAnnotationOcrType Example — VB

This example uses the GetSelectedAnnotationOcrType method to determine whether a selected OCR Zone annotation is a Picture zone or a Text zone. It then updates the appropriate option button on the form to indicate the OCR type.

This example also allows the user to change the OCR type of a selected OCR Zone annotation using the option buttons on the form. It uses the SetSelectedAnnotationOcrType method to set the OCR type.

```
Private Sub Form_Load()
    'This example determines whether a selected OCR recognition zone is
    'defined as a Picture Zone or a Text Zone and shows a dialog box to
    'allow the user to change the OCR zone type. Form used has 2
```

```
    'option buttons labeled Picture Zone and Text Zone
    Dim OCRType As AnnotationOcrTypeConstants
    'Determine if the selected OCR annotation mark is a Text Zone or a
    'Picture Zone and set the corresponding option button on the form.
    OCRType = frmMain.ImgEdit1.GetSelectedAnnotationOcrType
    If OCRType = wiOcrTypeText Then
      optTextZone.Value = True
    Else
      optPictureZone.Value = True
    End If
   frmOCRType.Show
End Sub
Private Sub optPictureZone_Click()
    'If the user clicks on the Picture Zone option button, the OCR
    'Zone type is changed to a Picture Zone.
    frmMain.ImgEdit1.SetSelectedAnnotationOcrType wiOcrTypePicture
End Sub
Private Sub optTextZone_Click()
    'If the user clicks on the Text Zone option button, the OCR
    'Zone type is changed to a Text Zone.
    frmMain.ImgEdit1.SetSelectedAnnotationOcrType wiOcrTypeText
End Sub
```

SetSelectedAnnotationOcrType Example — VB

This example uses the GetSelectedAnnotationOcrType method to determine whether a selected OCR Zone annotation is a Picture zone or a Text zone. It then updates the appropriate check box to indicate the OCR type. This example also allows the user to change the OCR type of a selected OCR Zone annotation using a check box. It uses the SetSelectedAnnotationOcrType method to set the OCR type.

```
void COCRZones::OnOK()
{
    if(m_TextZone.GetCheck())
      m_pParent->ImgEdit1.SetSelectedAnnotationOcrType(0);
    else
      m_pParent->ImgEdit1.SetSelectedAnnotationOcrType(1);
    CDialog::OnOK();
}
BOOL COCRZones::OnInitDialog()
{
    CDialog::OnInitDialog();
    short OCRType;
    // Determine if the selected OCR annotation mark is a Text Zone or a
    // Picture Zone and set the corresponding option button on the form.
    OCRType = m_pParent->ImgEdit1.GetSelectedAnnotationOcrType();
    if(OCRType == 0)   // wiOcrTypeText
    {
      m_TextZone.SetCheck(1);
      m_PictureZone.SetCheck(0);
    }
    else
```

```
    {
      m_TextZone.SetCheck(0);
      m_PictureZone.SetCheck(1);
    }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

ShowAnnotationGroup Method

Description Shows the specified annotation group.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowAnnotationGroup** [*GroupName*]

Arguments The ShowAnnotationGroup method has the following argument:

Parameter	Data Type	Setting
GroupName	String	The name of the desired annotation group

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

If the GroupName parameter is not entered, the ShowAnnotationGroup method shows all annotation groups.

This method does not show OCR Zone annotations. To show OCR Zone annotations, use the **OcrZoneVisibility** property.

Invoking the ShowAnnotationGroup method sets the **ImageModified** property to True.

Note: OCR Zone annotations are available with Imaging for Windows Professional Edition only.

See Also Display method, GetAnnotationGroup method, GetCurrentAnnotationGroup method, HideAnnotationGroup method, ImageModified property, OcrZoneVisibility property, SelectAnnotationGroup method.

ShowAnnotationGroup Example — VB

This example lets a user select an annotation group from a list and make it visible. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
Private Sub cmdShowGroup_Click()
    Dim strCurGroup As String
    'Determine which group the user selected from the listbox.
    strCurGroup = AnnoGroups.List(AnnoGroups.ListIndex)
    Form1.ImgEdit1.ShowAnnotationGroup (strCurGroup)
End Sub
```

ShowAnnotationGroup Example — VC++

This example lets a user select an annotation group from a list and make it visible. See the example for the GetAnnotationGroup method for instructions on how to list annotation groups.

```
void CFrmGroup::OnShowGroup()
{
    // This would allow a user to select a Group from a list and make it
    // visible. See example for GetAnnotationGroup method for how to list
    // annotation groups.
    CString szCurGroup;
    // Determine which group the user selected from the listbox.
    m_GroupList.GetText(m_GroupList.GetCurSel(),szCurGroup);
    VARIANT vCurGroup;
    V_VT(&vCurGroup) = VT_BSTR;
    V_BSTR(&vCurGroup) = szCurGroup.AllocSysString();
    if(pParentDlg)
    {
      pParentDlg->ImgEdit1.ShowAnnotationGroup(vCurGroup);
    }
}
```

ShowAnnotationToolPalette Method

Description Shows the Annotation Tool Palette (see page 533), which enables end users to add and edit annotations on the displayed image.

Available With

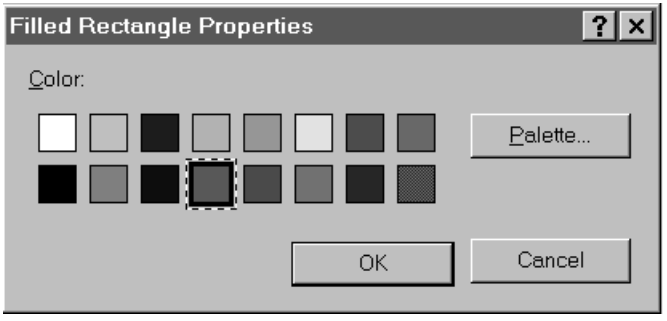
- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowAnnotationToolPalette** [*ShowAttributesDialog*, *Left*, *Top*, *ToolTipText*]

Arguments The ShowAnnotationToolPalette method has the following arguments:

Parameter	Data Type	Setting
ShowAttributesDialog	Boolean	Whether the Annotation Attributes dialog box is invoked: True (default) — When the end user clicks on a tool palette button using the right mouse button and selects properties from a pop-up menu, the appropriate Annotation Attributes dialog box appears (see below). When the end user clicks on the Rubber Stamp tool button, a pop-up menu appears that contains a list of previously defined stamps. False — The Annotation Attributes dialog box cannot be invoked.
Left	Long	The horizontal position of the Annotation Tool Palette in pixels, relative to the upper-left corner of the screen.
Top	Long	The vertical position of the Annotation Tool Palette in pixels, relative to the upper-left corner of the screen.
ToolTipText	String	Contains the tool tip text used for each Annotation Tool Palette button. Any number of strings may be entered; use the vertical bar (|) character to separate each string. If not specified, default strings are provided.



Returns None.

Remarks The ShowAnnotationToolPalette method can be invoked only once per control.

If the ShowAttributesDialog parameter is not entered, it is set to True by default.

If the Left and Top parameters are not entered, the Annotation Tool Palette is displayed at Position 10,10 relative to the Image Edit control.

ToolTip strings are displayed by default only as long as the tool palette control has focus. It is the responsibility of the calling application to insure that the control gets focus in the appropriate situations, either through explicit or implicit means.

To draw annotations using the Annotation Tool Palette, an annotation tool must be selected. This can be accomplished by the end user clicking on a button on the tool palette, or by invoking the **SelectTool** method. The annotation mark can then be drawn manually by the end user, or programatically by calling the **Draw** method. Changing the **AnnotationType** property with the Annotation Tool Palette displayed will not achieve the desired results.

Invoking the ShowAnnotationToolPalette method sets the **ImageModified** property to True.

Tab Characters

The Image Edit control does not support the use of Tab characters in any of the text-related annotation tools.

See Also Display method, Draw method, HideAnnotationGroup method, ImageModified property, SelectTool method, ShowAttribsDialog method, ToolSelected event, ToolTip event.

ShowAnnotationToolPalette Example — VB

This example shows how to invoke the annotation tool palette with user-defined tool tips. If tool tips are not specified, defaults are used.

```
Private Sub cmdToolPal_Click()
    'An example of showing the annotation tool palette with user defined
    'tooltips. If Tool tips are not specified default values are provided.
    'True parameter allows user to right click on a tool on the palette to
    'set attributes (color, size, etc) for that tool. If location is not
    'specified, palette is positioned relative to ImgEdit control.
    ToolTips$ = "Tool Tip #1|Tool Tip #2|Tool Tip #3|Tool Tip #4|
    ➥   Tool Tip #5|Tool Tip #6|Tool Tip #7|Tool Tip #8|Tool Tip #9|
    ➥   Tool Tip #10|Tool Tip #11"
    'Tool palette will appear in its default location since it is not
    'specified.
   ImgEdit1.ShowAnnotationToolPalette True, , , ToolTips$
End Sub
```

ShowAnnotationToolPalette Example — VC++

This example shows how to invoke the annotation tool palette with user-defined tool tips. If tool tips are not specified, defaults are used.

```
void CImgEdit1Dlg::OnToolPalette()
{
    // An example of showing the annotation tool palette with user defined
    // tooltips. If Tool tips are not specified default values are provided.
    // True parameter allows user to right click on a tool on the palette to
    // set attributes (color, size, etc) for that tool. If location is not
    // specified, palette is positioned relative to ImgEdit control.
    CString szToolTips;
    szToolTips = "Tool Tip #1|Tool Tip #2|Tool Tip #3|Tool Tip #4|
    ➥   Tool Tip #5|Tool Tip #6|Tool Tip #7|Tool Tip #8|Tool Tip #9|
    ➥   Tool Tip #10|Tool Tip #11";
    // Tool palette will appear in its default location since it is not
    // specified.
    VARIANT vToolTips;
    V_VT(&vToolTips) = VT_BSTR;
    V_BSTR(&vToolTips) = szToolTips.AllocSysString();
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default
    VARIANT vShowAttrDlg; V_VT(&vShowAttrDlg) = VT_BOOL;
    V_BOOL(&vShowAttrDlg) = TRUE;
    ImgEdit1.ShowAnnotationToolPalette(vShowAttrDlg, evt, evt, vToolTips);
}
```

ShowAttribsDialog Method

Description Shows an Annotation Attributes dialog box, which lets end users change the attributes of the selected annotation mark, or the attributes of the current type.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowAttribsDialog** *Option*

Arguments The ShowAttribsDialog method has the following arguments:

Parameter	Data Type	Setting
Option	Integer (enumerated)	0 — Show the Annotation Attributes dialog box (see page 490) for the currently selected annotation (default) ToolID - 1 — Show the attribute defaults for the given Tool ID number (see page 532) minus 1. 99 — Show the LinkTo dialog box (see page 543) for the selected annotation.

Note: The Option parameter is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0. With the exception of setting 99 (Link-To), it is also available with Imaging for Windows 98.

Returns None.

Remarks The type of dialog box invoked depends on the annotation type of the selected mark.

See Also The ToolID setting must be passed as the actual Tool ID number minus 1. For example, to show the attribute defaults for the Attach-a-Note tool, pass 7 (Attach-a-Note Tool ID of 8 minus 1).

An annotation must be selected if ToolID is not passed or is 0.

The Link To dialog box enables end users to make *any* annotation mark a Hyperlink annotation. A Hyperlink annotation can be linked to the following items:

- An image page in a TIFF image document stored on a local or network drive
- Universal Resource Locator (URL) on the World Wide Web

While users can create a Hyperlink annotation on any type of image, a Hyperlink annotation can be saved to disk only when the image upon which it is drawn is saved as a TIFF file. If end users burn-in a Hyperlink annotation, it no longer functions as a hypertext jump.

See Also AnnotationType property, SelectAnnotationGroup method, SelectTool method.

ShowAttribsDialog Example — VB

This example invokes the Annotation Attributes dialog box when the user right-clicks an annotation mark.

```
Private Sub ImgEdit1_MarkSelect(ByVal Button As Integer, ByVal Shift As
   ➥    Integer, ByVal Left As Long, ByVal Top As Long, ByVal Width As
   ➥    Long,ByVal Height As Long, ByVal MarkType As Integer, ByVal
   ➥    GroupName AsString)
   'If user right clicks on an annotation mark using the annotation select
   'tool, the attributes dialog box for that type of mark will be displayed
   'so the user could change the mark attributes.
   If Button = 2 Then     'User selected mark with a right mouse click.
```

```
      ImgEdit1.ShowAttribsDialog
    End If
End Sub
```

ShowAttribsDialog Example — VC++

This example invokes the Annotation Attributes dialog box when the user right-clicks an annotation mark.

```
void CImgEdit1Dlg::OnMarkSelect(short Button, short Shift, long Left,
   ➥    long Top, long Width, long Height, short MarkType, LPCTSTR
   ➥    GroupName)
{
    // If user right clicks on an annotation mark using the annotation
    // select tool, the attributes dialog box for that type of mark is
    // displayed so the user could change the mark attributes.
    VARIANT evt; V_VT(&evt) = VT_ERROR; // set to selected mark type
    if(Button == 2)     // User selected mark with a right mouse click
      ImgEdit1.ShowAttribsDialog(evt);
}
```

ShowMagnifier Method

Description Displays or hides the magnifier window. Can also optionally move and resize the magnifier window.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowMagnifier** *Show*[[,*Left*,*Top*][,*Width*,*Height*]]

Arguments The ShowMagnifier method has the following argument:

Parameter	Data Type	Setting
Show	Boolean	True — Display the magnifier window. False — Hide the magnifier window.
Left	Long	The horizontal position of the magnifier window in pixels, relative to the upper-left corner of the screen.
Top	Long	The vertical position of the magnifier window in pixels, relative to the upper-left corner of the screen.
Width	Long	The width of the magnifier window in pixels.

Parameter	Data Type	Setting
Height	Long	The height of the magnifier window in pixels.

Returns None.

Remarks If the position and size parameters are not specified, the magnifier window is shown at its previous location, in its previous size. If the Left parameter is specified, the Top parameter must also be specified. Likewise, if the Width parameter is specified, the Height parameter must also be specified.

The magnifier window can be resized by the user at run time; it does not have a caption bar.

See Also MagnifierStatus event, MagnifierZoom property.

ShowMagnifier Example — VB

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
Private Sub cmdMagnifier_Click()
    'Scroll the image to a specific horizontal/vertical pixel location.
    ImgEdit1.ScrollPositionX = 800
    ImgEdit1.ScrollPositionY = 1000
    'Magnifier window will zoom underlying image data to 4 times its size.
    ImgEdit1.MagnifierZoom = wi4XMagnifier  '1
    ImgEdit1.ShowMagnifier True
End Sub
```

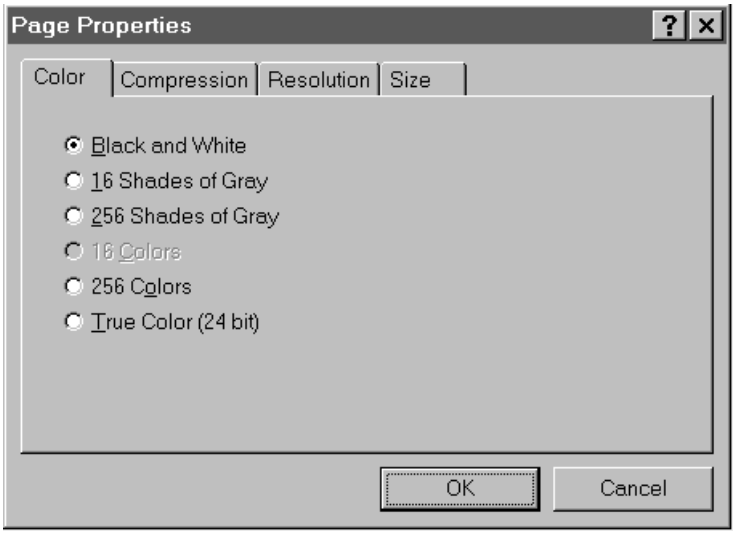
ShowMagnifier Example — VC++

This example scrolls the image to a specified location, sets the Magnifier zoom factor, and then displays the Magnifier window.

```
void CImgEdit1Dlg::OnMagnifier()
{
    // This example scrolls the image to a specific location and then
    // displays a magnifier window. Scroll the image to a specific
    // horizontal/vertical pixel location.
    ImgEdit1.SetScrollPositionX(800);
    ImgEdit1.SetScrollPositionY(1000);
    // Magnifier window zooms underlying image data to 4 times its size.
    ImgEdit1.SetMagnifierZoom(1); // wi4XMagnifier  // 1
    VARIANT evt; V_VT(&evt) = VT_ERROR;// set to default
    ImgEdit1.ShowMagnifier(TRUE,evt,evt,evt,evt);
}
```

ShowPageProperties Method

Description Shows the Page Properties dialog box (shown here), which enables end users to view or modify the properties of the displayed image page. The Page properties are: Color, Compression, Resolution, and Size.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowPageProperties**(*bReadOnly*)

Arguments The ShowPageProperties method has the following argument:

Parameter	Data Type	Setting
bReadOnly	Boolean	Determines read/write privileges: True — Read only. Users cannot alter property values. False — Read/write. Users can alter property values.

Returns Long.

Constant	Value	Description
vbOK	1	Returned when users click the OK button.
vbCancel	2	Returned when users click the Cancel button.

Remarks The **Display** method must be invoked prior to calling this method.

See Also Display method, PagePropertiesClose event.

ShowRubberStampDialog Method

Description Can be used in conjunction with the Annotation Tool Palette to show the Rubber Stamp Properties dialog box (shown here), which permits end users to create, delete, and edit Rubber Stamps.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- √ Imaging for Windows 95
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ShowRubberStampDialog**

Arguments None.

Returns None.

See Also SelectTool method, ShowAnnotationToolPalette method.

Undo Method

Description Undoes the last imaging operation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**Undo** [*Option*]

Arguments The Undo method has the following argument:

Parameter	Data Type	Setting
Option	Long	0 (default)

Returns None.

Remarks The **Display** method must be invoked prior to calling this method.

The Undo method undoes imaging operations only; it is limited to the number of undo levels specified in the **UndoLevels** property. If Undo is called when there is nothing to undo, an error is generated. Check the StatusCode property for the error.

Invoking the Undo method sets the **ImageModified** property to False if no other modifications were performed previously.

See Also StatusCode property, ContinueWithoutUndo property, Display method, ErrorSavingUndoInformation event, ImageModified property, Redo method, UndoLevels property.

Undo Example — VB

This example illustrates how to enable Undo functionality in an application.

```
Private Sub Form_Load()
    'UndoLevels must be set at form load to allocate undo buffers.
    ImgEdit1.UndoLevels = 1
End Sub
Private Sub cmdUndo_Click()
```

```
    'The Undo method will undo the last imaging operation performed.
    ImgEdit1.Undo
End Sub
```

Undo Example — VC++

This example illustrates how to enable Undo functionality in an application.

```
BOOL CImgEdit1Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    blnIsRectangle = FALSE;
    // UndoLevels must be set at form load to allocate undo buffers.
    ImgEdit1.SetUndoLevels(1);
    return TRUE;  // return TRUE  unless you set the focus to a control
}
void CImgEdit1Dlg::OnUndo()
{
    // The Undo method will undo the last imaging operation performed.
    VARIANT evt; V_VT(&evt) = VT_ERROR;  // set to default
    ImgEdit1.Undo(evt);
}
```

ZoomToSelection Method

Description Scales a selected portion of a displayed image so that it fits into the size of the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage *object*.**ZoomToSelection**

Arguments None.

Returns None.

Remarks A selection rectangle must be drawn on the image prior to calling this method.

The ZoomToSelection method updates the **Zoom** property with the zoom factor used.

See Also Display method, DrawSelectionRect method, FitTo method, SelectionRectDrawn event, Zoom property (Image Edit control).

ZoomToSelection Example — VB

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
Private Sub cmdZoomSelect_Click()
    'Enable rectangle drawing.
    ImgEdit1.SelectionRectangle = True
    'Draw rectangle.
    ImgEdit1.DrawSelectionRect 150, 200, 400, 400
    ImgEdit1.ZoomToSelection
End Sub
```

ZoomToSelection Example — VC++

This example draws a selection rectangle on an image and zooms in on the area bound by the rectangle.

```
void CImgEdit1Dlg::OnZoomto()
{
    // Enable rectangle drawing.
    ImgEdit1.SetSelectionRectangle(TRUE);
    // Draw rectangle.
    ImgEdit1.DrawSelectionRect( 150, 200, 400, 400);
    ImgEdit1.ZoomToSelection();
}
```

BadDocumentFileType Event

Description This event occurs when an image is being saved to an existing image document, and the current page attempting to be overwritten is read-only.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**BadDocumentFileType**(*Page*,*ErrorOut*,*Skip*,*Overwrite*)

Arguments The BadDocumentFileType event has the following arguments:

Parameter	Data Type	Setting
Page	Long	Specifies the page number being saved.
ErrorOut	Boolean	Determines whether the current operation terminates with an error when the event fires: True — The current operation terminates with an error (default). The Skip and Overwrite parameters are ignored. False — The current operation does not terminate with an error. Handling is determined by the Skip or Overwrite parameter setting.
Skip	Boolean	If the ErrorOut parameter is set to False, the Skip parameter determines whether to skip the file being processed: True — Skips the file being processed. The Overwrite parameter is ignored. False — Does not skip the file being processed (default). Handling is determined by the Overwrite parameter setting.
Overwrite	Boolean	If the ErrorOut and Skip parameters are set to False, the Overwrite parameter determines whether the current, read-only page in the document is overwritten: True — The current, read-only page in the document is overwritten. The overwritten image document retains the same name, which insures that any links to it are not broken. However, its extension may be changed to a read-only type, even though it contains read-write image data (the overwritten page). False — The current, read-only page in the document is not overwritten (default). Set Overwrite to False if you want to create a new, read-write file for the page being saved. (Keep in mind that other documents linking to this page will not be affected, but that the current document will no longer point to the same file for the current page).

Remarks This event is functional only when writing to AWD, BMP, and TIFF file types.

Links to image document pages are managed using Hyperlink annotations.

Note: AWD is not available with Imaging for Windows NT 4.0.

See Also SaveAs method, SavePage method.

BadDocumentFileType Example — VB

In this example, when the BadDocumentFileType event fires, the associated file is overwritten.

```
Private Sub ImgEdit1_BadDocumentFileType(ByVal Page As Long, ErrorOut As
    ➥    Boolean, SkipPage As Boolean, OverWritePage As Boolean)
    'If an attempt is made to rewrite a file type which is not supported for
    'write, the file is converted to TIFF and overwritten.
    ErrorOut = False
    Overwrite = True
End Sub
```

BadDocumentFileType Example — VC++

In this example, when the BadDocumentFileType event fires, the associated file is overwritten.

```
void CImgEdit2Dlg::OnBadDocumentFileTypeEditctrl1(long Page, BOOL FAR*
    ➥    ErrorOut, BOOL FAR* SkipPage, BOOL FAR* OverWritePage)
{
    // If an attempt is made to rewrite a file type which is not supported
    // for write, the file is converted to TIFF and overwritten.
    bool bErrorOut = FALSE;
    bool Overwrite = TRUE;
}
```

CheckContinuePrinting Event

Description This event occurs when a print operation requests whether it should cancel or continue printing. The value of the **ContinuePrinting** property provides the event with the desired response.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**CheckContinuePrinting**(*PagesPrinted*,*CurrentPage*,*Status*)

Arguments The CheckContinuePrinting event has the following arguments:

Parameter	Data Type	Setting
PagesPrinted	Long	The number of pages already printed.
CurrentPage	Long	The current page being printed.
Status	Integer	Not used currently.

Remarks The CheckContinuePrinting event fires at the beginning of a print operation and after each page is printed. The default is to continue printing.

Before you can use the ContinuePrinting property and the CheckContinuePrinting event, the **UseCheckContinuePrinting** property must be set to True.

See Also ContinuePrinting property, PrintImage method, UseCheckContinuePrinting property.

Close Event

Description This event occurs when one of the following methods is invoked:

- Display
- ClearDisplay
- DisplayBlankImage

The Close event can be used to indicate to end users that the displayed image is about to be closed.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 95 and 98

√ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_Close()`

Arguments None.

Remarks Once a Close event occurs, any changes made to displayed images are lost unless the **Save**, **SaveAs**, or **SavePage** method is invoked.

If desired, your program can check the **ImageModified** property to see if any changes have been made to the displayed image. If changes have been made, the program can be written so it prompts end users to save the changes.

See Also ClearDisplay method, Display method, DisplayBlankImage method, ImageModified property, Save method, SaveAs method, SavePage method.

Close Example — VB

In this example, when the Close event fires, the code evaluates the ImageModified property to see if changes were made to the image. If the value of the ImageModified property is True, the code prompts the user to save the image.

```
Private Sub ImgEdit1_Close()
    'This code could be used at the time a user attempts to open a new file
    'or clear a previously displayed image in order to prompt for changes to
    'be saved if the image has been modified.
    Dim intSaveImg As Integer
    If ImgEdit1.ImageModified = True Then
      intSaveImg = MsgBox("Do you want to save changes?", vbYesNoCancel,
      ➥    "Image has been modified")
      Select Case intSaveImg
        Case vbYes
          'Save the changes.
          ImgEdit1.Save
          'Drop down to the Open file code.
        Case vbNo
          'Don't save file, just drop down to the Open file code.
        Case vbCancel
          'User pressed cancel. Don't open a file, don't save changes.
          Exit Sub
      End Select
    End If
    'Open a new file at this point.
End Sub
```

Close Example — VC++

In this example, when the Close event fires, the code evaluates the ImageModified property to see if changes were made to the image. If the value of the ImageModified property is True, the code prompts the user to save the image.

```
void CImgEdit1Dlg::OnCloseEditctrl1()
{
    // This code could be used at the time a user attempts to open a new
    // file or clear a previously displayed image in order to prompt for
    // changes to be saved if the image has been modified.
    int iSaveImg;
    if(ImgEdit1.GetImageModified())
    {
      iSaveImg = AfxMessageBox("Do you want to save changes?",
      ➥     MB_YESNOCANCEL);
      switch (iSaveImg)
      {
        case IDYES:
          // Save the changes.
          VARIANT evt; V_VT(&evt) = VT_ERROR; // set to default for save at
                                              // zoom
          ImgEdit1.Save(evt);
```

```
            break;
            // Drop down to the Open file code.
          case IDNO:
            // Don't save file, just drop down to the Open file code.
            break;
          case IDCANCEL:
            // User pressed cancel. Don't open a file, don't save changes.
            return;
          }
      }
      // Open a new file at this point.
}
```

EditingTextAnnotation Event

Description This event occurs immediately after the Image Edit control enters or exits Text Edit mode.

When the Image Edit control is in Text Edit mode, it displays the Text Edit dialog box to let end users create or modify a Text, Attach-a-Note, or Hyperlink annotation.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**EditingTextAnnotation**(*Editing*)

Arguments The EditingTextAnnotation event has the following arguments:

Parameter	Data Type	Setting
Editing	Boolean	True — The Image Edit control is entering Text Edit mode. False — The Image Edit control is exiting Text Edit mode.

Remarks For this event to fire, an image must be displayed and the Image Edit control must be entering or exiting Text Edit mode.

The Image Edit control enters Text Edit mode when:

- An end user double-clicks on a selected Text, Attach-a-Note, or Hyperlink annotation.
- The **EditSelectedAnnotationText** method is invoked with a Text, Attach-a-Note, or Hyperlink annotation selected.
- A new Text, Attach-a-Note, or Hyperlink annotation is created.

The Image Edit control exits Text Edit mode when:

- An end user presses the Escape key (cancel).
- An end user clicks outside the Text Edit dialog box, but inside the client area.
- The **ExecuteTextEditCommand** method is invoked using the Finish command (value: 8 constant: wiFinishEditText) or the Cancel command (value: 9 constant: wiCancelEditText).
- Any other method is invoked.

> **Note:** When the Image Edit control is in Text Edit mode, the accelerator keys for Cut (Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), Undo (Ctrl+Z) and Backspace (Ctrl+H) are processed by the Text Edit box and are therefore not passed back to the application. In other words, an application that uses these keys as menu accelerators will not receive them when the Image Edit control is in Text Edit mode. Hyperlink annotations are available with Imaging for Windows Professional Edition only.

See Also EditSelectedAnnotationText method, ExecuteTextEditCommand method, SelectAnnotationGroup method.

EditingTextAnnotation Example — VB

In this example, when the EditingTextAnnotation event fires, the Undo selection on the Edit menu is updated so it indicates the action the Undo method will perform if the user invokes it.

```
Private Sub ImgEdit1_EditingTextAnnotation(ByVal Editing As Boolean)
    'When a text annotation is edited, this event could be used to change
    'the caption for the undo function to describe the function which would
    'be undone if the Undo menu pick was selected.
    mnuEditUndo.Caption = "Undo Annotation Edit"
End Sub
```

Error Event

Description This event occurs when a unexpected error is encountered (such as an error condition that occurs during painting, after a mouse click, etc.).

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit and Image Annotation controls.

Usage

```
Sub object_Error(Number,Description,Scode,Source,HelpFile,
                 HelpContext,CancelDisplay)
```

Arguments The Error event has the following arguments:

Parameter	Data Type	Setting
Number	Long	The native error number.
Description	String	The description of the error condition.
Scode	Long	The error return code.
Source	String	The source of the error condition.
HelpFile	String	The fully-qualified file name of the help system containing information about the error condition.
HelpContext	Long	The numeric help context ID that maps to the help topic that contains specific information about the error condition.
CancelDisplay	Boolean	Determines whether the error message is displayed: True — Display the error message (default). False — Continue processing without displaying the error message.

Remarks This error does not occur when a property is accessed or a method is invoked using invalid parameters.

ErrorSavingUndoInformation Event

Description This event occurs when an operation cannot save the information required to undo it.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**ErrorSavingUndoInformation**(*Error*)

Arguments The ErrorSavingUndoInformation event has the following argument:

Parameter	Data Type	Setting
Error	Long	The error that occurred.

Remarks The value of the **ContinueWithoutUndo** property provides the event with the desired processing alternative. The default is to continue the undo operation without saving the undo information.

See Also ContinueWithoutUndo property, Redo method, Undo method.

HyperlinkGoToDoc Event

Description This event occurs when end users click a Hyperlink annotation that is linked to an image page in an external image document.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**HyperlinkGoToDoc**(*Link*, *Page*, *Handled*)

Arguments The HyperlinkGoToDoc event has the following arguments:

Parameter	Data Type	Setting
Link	String	The fully-qualified name (see page 535) of the destination document.
Page	Long	The destination page number of the link.
Handled	Boolean	Determines the manner in which the link is handled: True — If the link is to the current document and page, it is not executed. This setting is recommended if the event is handled by the application. False — Forces the execution of the link.

Remarks Intradocument links are functional within TIFF documents only.

See Also AnnotationType property, HyperlinkGoToPage event.

HyperlinkGoToPage Event

Description This event occurs when end users click a Hyperlink annotation that is linked to an image page within the current image document.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98

  Imaging for Windows 95

  Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**HyperlinkGoToPage**(*Page*)

Arguments The HyperlinkGoToPage event has the following argument:

Parameter	Data Type	Setting
Page	Long	The destination page number of the link. If the link is to the current page, it is not executed.

Remarks Intradocument links are functional within TIFF documents only.

See Also AnnotationType property, HyperlinkGoToDoc event.

Load Event

Description This event occurs immediately after the **Display** method is invoked.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**Load**(*Zoom*)

Arguments The Load event has the following argument:

Parameter	Data Type	Setting
Zoom	Double	Returns the default zoom factor of 100. The Zoom parameter can also return the following values: -1 — Fit to width. The image was fit to the width of the window when it was displayed. -2 — Fit to height. The image was fit to the height of the window when it was displayed. -3 — No previously saved zoom factor.

Note: Zoom values of -1 and -2 may not be compatible with some applications.

Remarks Once a Load event occurs, other properties and methods can be set or invoked.

If desired, your program can use the returned zoom factor to set the **Zoom** property, or the Option parameter of the **FitTo** method, prior to displaying another image.

See Also Display method, FitTo method, Zoom property (Image Edit).

Load Example — VB

In this example, when the Load event fires, properties that control image display are set.

```
Private Sub ImgEdit1_Load(ByVal Zoom As Double)
    'Here are some examples of default settings you might
    'want to specify prior to displaying an image.
    'Repaint any changes immediately (ex. zoom or resolution changes, etc.)
    ImgEdit1.AutoRefresh = True
    'ScrollBars already default to true, but this is modifiable if desired.
    ImgEdit1.ScrollBars = True
    'Scale black and white images to grayscale. Keep color image displayed
    'as color images.
    ImgEdit1.DisplayScaleAlgorithm = wiScaleOptimize
    'Allow user to scroll image using keyboard shortcuts.
```

```
    ImgEdit1.ScrollShortcutsEnabled = True
    'Display all OCR zones.
    ImgEdit1.OcrZoneVisibility = wiOcrShowAllZones  '3
End Sub
```

Load Example — VC++

In this example, when the Load event fires, properties that control image display are set.

```
void CImgEdit1Dlg::OnLoadEditctrl1(double Zoom)
{
    // Here are some examples of default settings you might want to specify
    // prior to displaying an image. Repaint any changes immediately
    // (ex. zoom or resolution changes, etc.)
    ImgEdit1.SetAutoRefresh(TRUE);
    // ScrollBars already default to true, but this is modifiable if
    // desired.
    ImgEdit1.SetScrollBars(TRUE);
    // Scale black and white images to grayscale. Keep color image displayed
    // as color images.
    ImgEdit1.SetDisplayScaleAlgorithm(4);   // wiScaleOptimize
    // Allow user to scroll image using keyboard shortcuts.
    ImgEdit1.SetScrollShortcutsEnabled(TRUE);
    // Display all OCR zones.
    ImgEdit1.SetOcrZoneVisibility(3);  // wiOcrShowAllZones  // 3
}
```

MagnifierStatus Event

Description This event occurs when the status of the magnifier window has changed.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**MagnifierStatus**(*hWnd*, *Status*, *MagnifierZoom*, *Left*, *Top*, *Width*, *Height*)

Arguments The MagnifierStatus event has the following arguments:

Parameter	Data Type	Setting
hWnd	Stdole.OLE_HANDLE	The handle to the magnifier window
Status	Long	The current status of the magnifier window. Can be one of the following values: 0 — Primary magnifier window created. 1 — Primary magnifier window locked. No further status will be reported. 2 — Primary magnifier window closed.
MagnifierZoom	Long	The current setting of the **MagnifierZoom** property. Can be one of the following values: -1 — The displayed image in the magnifier window is locked at the current **Zoom** factor 0 — 2 times the current Zoom factor 1 — 4 times the current Zoom factor 2 — 8 times the current Zoom factor
Left	Long	The horizontal position of the magnifier window in pixels, relative to the upper-left corner of the screen.
Top	Long	The vertical position of the magnifier window in pixels, relative to the upper-left corner of the screen.
Width	Long	The width of the magnifier window in pixels.
Height	Long	The height of the magnifier window in pixels.

Remarks None.

See Also MagnifierZoom property, ShowMagnifier property.

MarkEnd Event

Description This event occurs immediately after the end user or program completes the drawing of an annotation mark.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**MarkEnd**(*Left*,*Top*,*Width*,*Height*,*MarkType*,*GroupName*)

Arguments The MarkEnd event has the following arguments:

Parameter	Data Type	Setting
Left	Long	The starting horizontal position of the annotation mark within the image in image pixels.
Top	Long	The starting vertical position of the annotation mark within the image in image pixels.
Width	Long	The ending horizontal position of the annotation mark within the image in image pixels.
Height	Long	The ending vertical position of the annotation mark within the image in image pixels.
MarkType	Integer	The AnnotationType (see page 528) of the annotation mark (except for the Select Annotations annotation type).
GroupName	String	The name of the annotation group to which the annotation belongs.

Remarks The MarkType parameter cannot be the Select Annotations annotation type.

A MarkEnd event does not occur if a Select Annotations rectangle is drawn (AnnotationType of 0).

See Also Draw method.

MarkEnd Example — VB

In this example, when the MarkEnd event fires, the Undo selection on the Edit menu is updated so it indicates the action the Undo method will perform if the user invokes it.

```
Private Sub ImgEdit1_MarkEnd(ByVal Left As Long, ByVal Top As Long, ByVal
    ➥    Width As Long, ByVal Height As Long, ByVal MarkType As Integer,
    ➥    ByVal GroupName As String)
    'This event could be used after an annotation mark is drawn to change
    'the caption on an undo menu selection to indicate what action undo
    'will take.
    mnuEditUndo.Caption = "Undo Annotation"
End Sub
```

MarkEnd Example — VC++

In this example, when the MarkEnd event fires, the Undo selection on the Edit menu is updated so it indicates the action the Undo method will perform if the user invokes it.

```
void CImgEdit2Dlg::OnMarkEnd(long Left, long Top, long Width, long Height,
    ➥    short MarkType, LPCTSTR GroupName)
{
    // This event could be used after an annotation mark is drawn to change
    // the caption on an undo menu selection to indicate what action undo
    // will take.
    mnuEditUndo.Caption = "Undo Annotation";
}
```

MarkMove Event

Description This event occurs immediately after the end user or program moves or resizes an annotation mark.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**MarkMove**(*MarkType*,*GroupName*)

Arguments The MarkMove event has the following arguments:

Parameter	Data Type	Setting
MarkType	Integer	The AnnotationType (see page 528) of the annotation mark (except for the Select Annotations annotation type).
GroupName	String	The name of the annotation group to which the annotation belongs, or an empty string if more than one annotation mark is selected.

Remarks The MarkType parameter cannot be the Select Annotations annotation type.

A MarkMove event does not occur if a Select Annotations rectangle is drawn (AnnotationType of 0).

See Also Draw method.

MarkMove Example — VB

In this example, when the MarkMove event fires, the Undo selection on the Edit menu is updated so it indicates the action the Undo method will perform if the user invokes it.

```
Private Sub ImgEdit1_MarkMove(Button As Integer, Shift As Integer, x As
    ➥    Single, y As Single)
    'This event could be used after an annotation mark is moved to change
    'the caption on an undo menu selection to indicate what action undo
    'will take.
    mnuEditUndo.Caption = "Undo Annotation Move"
End Sub
```

MarkMove Example — VC++

In this example, when the MarkMove event fires, the Undo selection on the Edit menu is updated so it indicates the action the Undo method will perform if the user invokes it.

```
void CImgEdit2Dlg::OnMarkMove(short Button, short Shift, long x, long y)
{
    // This event could be used after an annotation mark is moved to change
    // the caption on an undo menu selection to indicate what action undo
    // will take.
    mnuEditUndo.Caption = "Undo Annotation Move";
}
```

MarkSelect Event

Description This event occurs immediately after the end user or program selects an annotation mark.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**MarkSelect**(*Button*,*Shift*,*Left*,*Top*,*Width*,*Height*, *MarkType*,*GroupName*)

Arguments The MarkSelect event has the following arguments:

Parameter	Data Type	Setting
Button	Integer (enumerated)	Specifies the mouse button pressed: 1 — Left button 2 — Right button 4 — Middle button

Parameter	Data Type	Setting
Shift	Integer (enumerated)	Specifies whether the Shift, Ctrl, or Alt key is pressed: 1 — Shift key 2 — Ctrl key 4 — Alt key
Left	Long	The beginning horizontal position within the image in image pixels.
Top	Long	The beginning vertical position within the image in image pixels.
Width	Long	The ending horizontal position within the image in image pixels.
Height	Long	The ending vertical position within the image in image pixels.
MarkType	Integer	The AnnotationType (see page 528) of the annotation mark (except for the Select Annotations annotation type).
GroupName	String	The name of the annotation group to which the annotation belongs.

Remarks If multiple annotations are selected, a MarkType parameter value of 0 is returned. The MarkSelect event returns Left and Top coordinates that are relative to the Image Edit control. You need to add the values of the ScrollPositionX and ScrollPositionY properties to obtain the absolute position within the image.

The GroupName parameter is an empty string when multiple annotations are selected.

See Also SelectAnnotationGroup method.

MarkSelect Example — VB

In this example, when the MarkSelect event fires, the MarkLeft and MarkTop global variables are assigned the Left and Top coordinates of the selected annotation.

Then, the EditSelectedAnnotationText method in the cmdEditText_Click event procedure, permits the user to edit the text of the selected annotation.

```
Private Sub ImgEdit1_MarkSelect(ByVal Button As Integer, ByVal Shift As
➥    Integer, ByVal Left As Long, ByVal Top As Long, ByVal Width As
➥    Long, ByVal Height As Long, ByVal MarkType As Integer, ByVal
➥    GroupName As String)
```

```
    'Dimension MarkLeft and MarkTop as global variables so these values can
    'be passed back to the EditSelectedAnnotationText method.
    MarkLeft = Left
    MarkTop = Top
End Sub
Private Sub cmdEditText_Click()
    'This would allow the user to edit the text of an existing selected
    'annotation mark.
    On Error GoTo HandleIt
    'Coordinates of the mark are obtained from the MarkSelect event (see
    'code above).
    ImgEdit1.EditSelectedAnnotationText MarkLeft, MarkTop
    Exit Sub
HandleIt:
    'Handle error if no mark is selected or the wrong type of mark is
    'selected.
     MsgBox Err.Description
End Sub
```

MarkSelect Example — VC++

In this example, when the MarkSelect event fires, the m_MarkLeft and m_MarkTop variables are assigned the Left and Top coordinates of the selected annotation.

Then, the EditSelectedAnnotationText method in the CImgEdit2Dlg::OnAnnotext() procedure, permits the user to edit the text of the selected annotation.

```
void CImgEdit2Dlg::OnAnnotext()
{
    // This would allow the user to edit the text of an existing
    // selected annotation mark.
    TRY
     {
      // Coordinates of the mark are obtained from the MarkSelect event
      // (see code below).
      ImgEdit1.EditSelectedAnnotationText(m_MarkLeft, m_MarkTop);
    }
    CATCH (COleDispatchException, e)
    {
      // Handle error if no mark is selected or the wrong type of mark is
      // selected.
      AfxMessageBox(e->m_strDescription);
      // Err.Description
    }
    END_CATCH
}
void CImgEdit2Dlg::OnMarkSelect(short Button, short Shift, long Left,
➥ long Top, long Width, long Height, short MarkType, LPCTSTR GroupName)
{
    m_MarkLeft = Left;
    m_MarkTop = Top;
}
```

PagePropertiesClose Event

Description This event occurs when an end user clicks OK after having changed the compression type on the Page Properties dialog box (see page 496).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_PagePropertiesClose()`

Arguments None.

Remarks This event gives you the opportunity to execute code that saves page changes for undo purposes.

See Also ShowPageProperties method.

PasteClip Event

Description This event occurs when data being pasted onto a base image is too large (dimensionally) to within the confines of the image.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_PasteClip(Mode)`

Arguments The PasteClip event has the following argument:

Parameter	Data Type	Setting
Mode	Long	Determines the action to be performed when the event fires: vbYes — Enlarge the base image so it can accommodate the paste data (default). The base image is enlarged by adding white data around the image borders. vbNo — Do not enlarge the base image, clip the paste data to the current size of the base image. vbCancel — Cancel the Paste operation.

Remarks None.

See Also ClipboardPaste method, CompletePaste method, PasteCompleted event.

PasteCompleted Event

Description This event occurs immediately after pasted image or annotation data is committed to a location on an image. Pasted image or annotation data is committed when an end user clicks on a form, outside of a pasted image or annotation, or whenever the **CompletePaste** method is invoked.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_PasteCompleted()`

Arguments None.

Remarks The **ClipboardPaste** method must be invoked for this event to be triggered.

See Also ClipboardPaste method, CompletePaste method, PasteClip event.

ReadyStateChange Event

Description This event occurs when the state of the control has changed. When it fires, check the **ReadyState** property or the ReadyState parameter to ascertain the readiness state of the control.

You can use the ReadyState property and ReadyStateChange event to manage the asynchronous downloading of images from the World Wide Web.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Applies To Image Edit, Image Annotation, Image OCR, and Image Thumbnail controls.

Usage **Sub** *object*_**ReadyStateChange**(*ReadyState*)

Arguments The ReadyStateChange event has the following argument:

Parameter	Data Type	Setting
ReadyState	Long	The readiness state of the control: 0 — Control is initializing and retrieving properties; cannot perform methods. 2 — Control has initialized and downloading has started; cannot perform methods. 4 — Control is ready for all requests; can perform methods.

Remarks This event is useful when the **Image** property is set to a URL pathname.

The control can perform methods only when its ReadyState property is 4.

See Also Image property, ReadyState property.

Scroll Event

Description This event occurs immediately after the end user or program scrolls an image in the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**Scroll()**

Arguments None.

See Also ScrollBars property, ScrollImage method, ScrollPositionX property, ScrollPositionY property, ScrollShortcutsEnabled property.

SelectionRectDrawn Event

Description This event occurs immediately after the end user presses the left mouse button, or the **DrawSelectionRect** method is invoked.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**SelectionRectDrawn**(*Left*,*Top*,*Width*,*Height*)

Arguments The SelectRectDrawn event has the following arguments:

Parameter	Data Type	Setting
Left	Long	The beginning horizontal position within the image in image pixels relative to the upper-left corner of the Image Edit display window.
Top	Long	The beginning vertical position within the image in image pixels relative to the upper-left corner of the Image Edit display window.
Width	Long	The ending horizontal position within the image in image pixels relative to the upper-left corner of the Image Edit display window.
Height	Long	The ending vertical position within the image in image pixels relative to the upper-left corner of the Image Edit display window.

Remarks The **SelectionRectangle** property must be set to True for this event to be triggered.

If the end user draws a selection rectangle of 3 pixels or less, no selection rectangle is drawn. When this happens, the SelectionRectDrawn event is still triggered, but all of its parameters contain a value of 0. Your program can use these 0 values to determine whether cut, copy, or delete operations should be enabled.

See Also DrawSelectionRect method, SelectionRectangle property.

SelectionRectDrawn Example — VB

In this example, when the SelectionRectDrawn event fires, its parameters are evaluated. A state variable is then set to True or False to indicate whether a selection rectangle has been drawn by the user.

```
Private Sub ImgEdit1_SelectionRectDrawn(ByVal Left As Long, ByVal Top As
    ➥    Long, ByVal Width As Long, ByVal Height As Long)
    'If the parameters returned by this event are all 0, the user has
    'clicked on the image and removed a previously drawn rectangle. If
    'parameters have values, a rectangle has been drawn (note that
    'blnIsRectangle would need to be set to False if another image
    'is opened/created or another page is displayed).
    'This technique could also be used for enabling disable cut/copy paste
    'menu picks.
    Dim blnIsRectangle As Boolean
    If Width + Height = 0 Then
        blnIsRectangle = False
    Else
        blnIsRectangle = True
    End If
End Sub
```

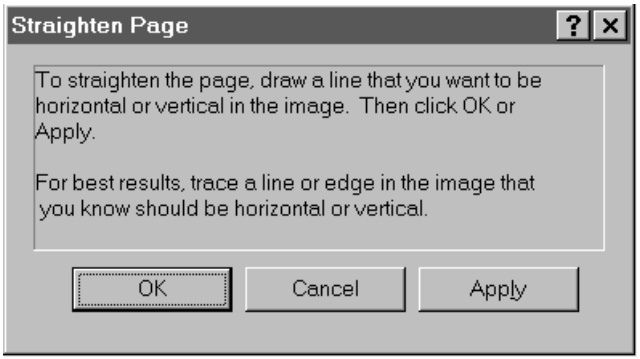
SelectionRectDrawn Example — VC++

In this example, when the SelectionRectDrawn event fires, its parameters are evaluated. A state variable is then set to True or False to indicate whether a selection rectangle has been drawn by the user.

```
void CImgEdit1Dlg::OnSelectionRectDrawn(long Left, long Top, long Width,
    ➥    long Height)
{
    // If the parameters returned by this event are all 0, the user has
    // clicked on the image and removed a previously drawn rectangle. If
    // parameters have values, a rectangle has been drawn IsRectangle
    // would need to be set to false if another image is opened/created
    // or another page is displayed. This technique could also be used
    // for enabling disable cut copy paste menu picks.
    if (Width + Height == 0)
      blnIsRectangle = FALSE;
    else
      blnIsRectangle = TRUE;
}
```

StraightenPage Event

Description This event occurs immediately after the end user straightens a page and then clicks OK on the Straighten Page dialog box (shown here).



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_StraightenPage()`

Arguments None.

Remarks This event works in conjunction with the **ManualDeSkew** method.

See Also ManualDeSkew method.

ToolPaletteHidden Event

Description This event occurs immediately after the Annotation Tool Palette (see page 533) has been hidden.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**ToolPaletteHidden**(*Left*,*Top*)

Arguments The ToolPaletteHidden event has the following arguments:

Parameter	Data Type	Setting
Left	Long	The horizontal position of the Annotation Tool Palette in pixels when it was closed, relative to the upper-left corner of the screen.
Top	Long	The vertical position of the Annotation Tool Palette in pixels when it was closed, relative to the upper-left corner of the screen.

Remarks The Annotation Tool Palette can be hidden by invoking the **HideAnnotationToolPalette** method or by clicking on the Close icon on the Annotation Tool Palette caption bar.

See Also HideAnnotationToolPalette method.

ToolPaletteHidden Example — VB

In this example, when the ToolPaletteHidden event fires, the TPLeft and TPTop global variables are assigned to the coordinates of the Annotation Tool Palette (before it was hidden).

These values can be used by the ShowAnnotationToolPalette method to have the tool palette reappear at the same coordinates.

```
Private Sub ImgEdit1_ToolPaletteHidden(ByVal Left As Long, ByVal Top As
    ➥    Long)
    'This event could be used to track the current location of the tool
    'palette when it is hidden so that it could be shown at the same
    'location when it is shown again. Dimension TPLeft and TPTop as
    'global long variables.
    TPLeft = Left
    TPTop = Top
    'Pass TPLeft and TPTop in to the ShowAnnotationToolPalette method.
End Sub
```

ToolPaletteHidden Example — VC++

In this example, when the ToolPaletteHidden event fires, the TPLeft and TPTop global variables are assigned to the coordinates of the Annotation Tool Palette (before it was hidden).

These values can be used by the ShowAnnotationToolPalette method to have the tool palette reappear at the same coordinates.

```
void CImgEdit2Dlg::OnToolPaletteHidden(long Left, long Top)
{
    // This event could be used to track the current location of the tool
    // palette when it is hidden so that it could be shown at the same
```

```
    // location when it is shown again. Dimension TPLeft and TPTop as
    // global long variables.
    long TPLeft = Left;
    long TPTop = Top;
    // Pass TPLeft and TPTop in to the ShowAnnotationToolPalette method.
}
```

ToolSelected Event

Description This event occurs immediately after the end user selects a tool from the Annotation Tool Palette (see page 533).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage **Sub** *object*_**ToolSelected**(*ToolID*)

Arguments The ToolSelected event has the following argument:

Parameter	Data Type	Setting
ToolID	Integer (enumerated)	Identifier of the tool, which corresponds to the Tool ID settings (see page 532).

Remarks The **ShowAnnotationToolPalette** method does not have to be invoked to trigger this event. The **SelectTool** method can be used to select the annotation tool in the Annotation Tool Palette.

See Also SelectTool method, ShowAnnotationToolPalette method, ToolTip event.

ToolSelected Example — VB

In this example, when the ToolSelected event fires, all of the annotation tool names that appear in the Tools menu are unchecked. (There is one menu item for each annotation tool in the Annotation Tool Palette.)

Then, according to the value of the ToolID parameter, the procedure checks the menu item that corresponds to the annotation tool selected by the user.

```
Private Sub ImgEdit1_ToolSelected(ByVal ToolId As Integer)
    'This event could be used to keep menu selections in sync with
    'selections made from the Annotation Tool Palette Initialize all menu
    'selections to unchecked.
    mnuToolsAnnoSelect.Checked = False
    mnuToolsFreeLine.Checked = False
```

```
    mnuToolsHighLine.Checked = False
    mnuToolsStraightLine.Checked = False
    mnuToolsHollowRect.Checked = False
    mnuToolsFilledRect.Checked = False
    mnuToolsText.Checked = False
    mnuToolsNote.Checked = False
    mnuToolsTextFile.Checked = False
    mnuToolsStamp.Checked = False
    mnuToolsHyperlinks.Checked = False
    'Use the ToolId parameter to determine which tool the user selected and
    'then check the corresponding menu item.
    Select Case ToolId
       Case 1
         mnuToolsAnnoSelect.Checked = True
       Case 2
         mnuToolsFreeLine.Checked = True
       Case 3
         mnuToolsHighLine.Checked = True
       Case 4
         mnuToolsStraightLine.Checked = True
       Case 5
         mnuToolsHollowRect.Checked = True
       Case 6
         mnuToolsFilledRect.Checked = True
       Case 7
         mnuToolsText.Checked = True
       Case 8
         mnuToolsNote.Checked = True
       Case 9
         mnuToolsTextFile.Checked = True
       Case 10
         mnuToolsStamp.Checked = True
       Case 11
         mnuToolsHyperlinks.Checked = True
    End Select
End Sub
```

ToolSelected Example — VC++

In this example, when the ToolSelected event fires, all of the annotation tool names that appear in the Tools menu are unchecked. (There is one menu item for each annotation tool in the Annotation Tool Palette.)

Then, according to the value of the ToolID parameter, the procedure checks the menu item that corresponds to the annotation tool selected by the user.

```
void CImgEdit2Dlg::OnToolSelected(short ToolId)
{
    // Demonstrates using this event to keep menu selections in sync with
    // selections made from the Annotation Tool Palette. Initialize all
    // menu selections to unchecked.
    mnuToolsAnnoSelect.Checked = False;
```

```
    mnuToolsFreeLine.Checked = False;
    mnuToolsHighLine.Checked = False;
    mnuToolsStraightLine.Checked = False;
    mnuToolsHollowRect.Checked = False;
    mnuToolsFilledRect.Checked = False;
    mnuToolsText.Checked = False;
    mnuToolsNote.Checked = False;
    mnuToolsTextFile.Checked = False;
    mnuToolsStamp.Checked = False;
    mnuToolsHyperlinks.Checked = False;
    // Use the ToolId parameter to determine which tool the user selected
    // and then check the corresponding menu item.
    switch (ToolId)
    {
      case 1:
        mnuToolsSelectArrow.Checked = True;
      case 2:
        mnuToolsSelectFreeLine.Checked = True;
      case 3:
        mnuToolsSelectHighLine.Checked = True;
      case 4:
        mnuToolsStraightLine.Checked = True;
      case 5:
        mnuToolsSelectHollowRect.Checked = True;
      case 6:
        mnuToolsSelectFilledRect.Checked = True;
      case 7:
        mnuToolsSelectText.Checked = True;
      case 8:
        mnuToolsSelectNote.Checked = True;
      case 9:
        mnuToolsSelectTextFile.Checked = True;
      case 10:
        mnuToolsSelectStamp.Checked = True;
      case 11:
        mnuToolsSelectHyperlinks.Checked = True;
    }
}
```

ToolTip Event

Description This event occurs immediately after a tool tip has been displayed on the Annotation Tool Palette (see page 533).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Applies To Image Edit control.

Usage `Sub object_ToolTip(Index)`

Arguments The ToolTip event has the following argument:

Parameter	Data Type	Setting
Index	Integer (enumerated)	Index value of the tool tip displayed, which corresponds to the Tool ID settings (see page 532).

Remarks The **ShowAnnotationToolPalette** method displays the Annotation Tool Palette. The **SelectTool** method selects the annotation tool in the Annotation Tool Palette.

See Also SelectTool method, ShowAnnotationToolPalette method, ToolSelected event.

PageType Settings

The following list shows the valid PageType settings:

Constant	Setting	Description
wiPageTypeBW	1	Black-and-white
wiPageTypeGray4	2	Gray scale, 4-bit
wiPageTypeGray8	3	Gray scale, 8-bit
wiPageTypePal4	4	Palettized, 4-bit
wiPageTypePal8	5	Palettized, 8-bit
wiPageTypeRGB24	6	RGB, 24-bit
wiPageTypeBGR24	7	BGR, 24-bit

Note: Palettized4 is not available as a parameter for the **ConvertPageType** method unless the image presently has a 4-bit palette.

PageType Settings for SaveAs and SavePage

The valid PageType settings for the **SaveAs** or **SavePage** method depend on the FileType setting entered. The following list shows the valid PageType settings:

Constant	Setting	Description	Prerequisite FileType
wiPageTypeBW	1	Black-and-white	1 — TIFF, 2 — AWD, or 3 — BMP
wiPageTypeGray4	2	Gray4	1 — TIFF
wiPageTypeGray8	3	Gray8	1 — TIFF
wiPageTypePal4	4	Palettized4	1 — TIFF or 3 — BMP
wiPageTypePal8	5	Palettized8	1 — TIFF or 3 — BMP
wiPageTypeRGB24	6	RGB24	1 — TIFF
wiPageTypeBGR24	7	BGR24	3 — BMP

Note: AWD is not available with Imaging for Windows NT 4.0.
To use a PageType setting of 4 (Palettized4), the displayed image must already be a Palettized4 image.

Annotation Type Settings

The following list shows the valid AnnotationType settings:

Constant	Setting	Description
wiNone	0 (default)	None
wiStraightLine	1	Straight Line
wiFreehandLine	2	Freehand Line
wiHollowRect	3	Hollow Rectangle
wiFilledRect	4	Filled Rectangle
wiImageEmbedded	5	Image Embedded
wiImageReference	6	Image Reference
wiText	7	Text
wiTextStamp	8	Text Stamp
wiTextFromFile	9	Text From File
wiTextAttachment	10	Attach-a-Note

Constant	Setting	Description
wiAnnotationSelection	11	Select Annotations
wiHyperlink	12	Hyperlink
wiOcrRegion	13	OCR Zone

Note: Hyperlink and OCR Zones are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

FileType Settings

The following list shows the valid FileType settings:

Constant	Setting	Description
wiFileTypeTIFF	1	TIFF
wiFileTypeAWD	2	AWD
wiFileTypeBMP	3	BMP
wiFileTypePCX	4	PCX
wiFileTypeDCX	5	DCX
wiFileTypeJPG	6	JPG
wiFileTypeXIF	7	XIF
wiFileTypeGIF	8	GIF
wiFileTypeWIFF	9	WIFF

Displayed file types of PCX, DCX, JPG, XIF, GIF, or WIFF must be saved using a FileType setting of 1 (TIFF), 2 (AWD), or 3 (BMP).

If a multipage image file (AWD, DCX, TIFF, WIFF, or XIF) is saved as a BMP file, only the page currently being displayed is saved to the new file.

A file with a .JFX extension is also a TIFF file type.

Note: AWD is not available with Imaging for Windows NT 4.0.

GIF and WIFF are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0; and Imaging for Windows 98.

CompressionType Settings

The following list shows the valid CompressionType settings:

Setting	Description
1	No compression
2	Group3(1D)
3	Group3(Modified Huffman)
4	PackBits
5	Group4(2D)
6	JPEG
7	Reserved
8	Group3(2D) (read only)
9	LZW

AWD and BMP files must be saved using a CompressionType setting of 1.

Note: AWD is not available with Imaging for Windows NT 4.0.

CompressionInfo Settings

The following list shows the valid CompressionInfo settings (bit-wise values):

Setting	Description
0	No compression — If CompressionType is set to 1 (no compression) Default compression — If CompressionType is set to a value other than 1
1	EOL
2	Packed Lines
4	Prefixed EOL
8	Compressed LTR
16	Expand LTR
32	Negate
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)

Setting	Description
4096	High Resolution/High Quality (JPEG compression only)
8192	High Resolution/Medium Quality (JPEG compression only)
16384	High Resolution/Low Quality (JPEG compression only)

AWD, BMP, and other uncompressed files must be saved using a CompressionInfo setting of 0. LZW-compressed files must be saved using a CompressionInfo setting of 8.

Note: AWD is not available with Imaging for Windows NT 4.0.

Tool ID Settings

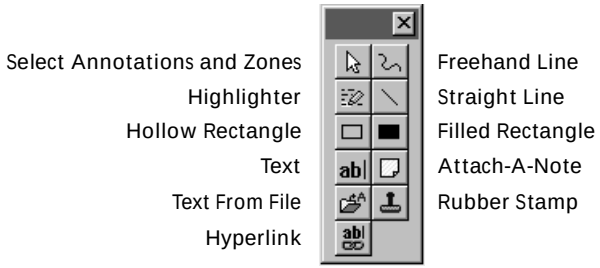
The following list shows the valid Tool ID settings:

Parameter	Data Type	Setting	Description
ToolID	Integer (enumerated)	0	No Tool
		1	Select Annotations
		2	Freehand Line
		3	Highlighter
		4	Straight Line
		5	Hollow Rectangle
		6	Filled Rectangle
		7	Filled RectangleText
		8	Attach-a-Note
		9	Text From File
		10	Rubber Stamp
		11	Hyperlink[a]
		12	OCR Zones[a]

a. Available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

Annotation Tool Palette

The following illustration shows the components of the Annotation Tool Palette. Refer to the appropriate section that follows to see an explanation of each one.



Select Annotations and Zones

The only tool that does not draw an annotation, the Select Annotations and Zones tool is used to select single or multiple annotation marks, including OCR zones.

End users can select annotation marks and zones by clicking on them, or by dragging a selection rectangle around them.

Note: OCR Zone annotations are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

Highlighter

The Highlighter tool is used to draw a transparent, filled rectangle for highlighting emphasis.

Note: End users can right-click on the tool to set the highlight color.

Hollow Rectangle

The Hollow Rectangle tool is used to place a border around areas of an image for emphasis.

Note: End users can right-click on the tool to set line properties, such as width, style, and color.

Text

The Text tool is used to enter text directly on an image.

Note: End users can right-click on the tool to set font properties, such as name, style, size, color, script, strikethrough, and underline.

Text From File

The Text From File tool is used to enter text from a file onto an image. When used, it invokes the Select Text File dialog box, which permits users to select the desired text file.

Note: End users can right-click on the tool to set font properties, such as name, style, size, color, script, strikethrough, and underline.

Freehand Line

The Freehand Line tool is used to draw a freehand line on a section of text or a portion of an image for emphasis.

Note: End users can right-click on the tool to set the line properties, such as width and color.

Straight Line

The Straight Line tool is used to underline text, demarcate a section of a page, or draw callout lines.

Note: End users can right-click on the tool to set the line properties, such as width and color.

Filled Rectangle

The Filled Rectangle tool is used to create a color background for text annotations or to hide a portion of an image.

Note: End users can right-click on the tool to set the fill color.

Attach-a-Note

The Attach-a-Note tool is used to enter text into a background rectangle on an image.

Note: End users can right-click on the tool to set the background color and font properties, such as name, style, size, color, script, strikethrough, and underline.

Rubber Stamp

The Rubber Stamp tool is used to place an image or text stamp onto an image.

Note: End users can right-click on the tool to format the stamp text.

Hyperlink

The Hyperlink tool is used to enter a hypertext link directly on an image. When used, it invokes the Link To dialog box, which permits users to specify the desired link.

Note: End users can right-click on the tool to set font properties, such as name, style, size, color, script, strikethrough, and underline.
The Hyperlink tool is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

Fully-qualified Image Document Names

A fully-qualified image document name includes the complete path to where the image document file or *managed* image document resides. There are several formats, depending on the file system or document management database in use. Here are some examples:

Windows — c:\insurance\claims\claim234.tif

UNC (Universal Naming Convention) — \\INSURANCE\CLAIMS\CLAIM234.TIF

URL (Uniform Resource Locator) — http://www.dxcessinsurance.com/auto/claims.tif or ftp://ftp.dxcess.ins.net/pub/access/claims.tif

Imaging 1.x Server Document Manager database — Image://DXCESS/ins_db1:/INSURANCE/AUTO/CLAIMS/CLAIM234

Imaging 1.x Server Image File volume — Image://DXCESS\images:\auto\claims.tif

Imaging 3.x Server Document Manager database — Imagex://claim234

Note: URLs can be either HTTP (HyperText Transfer Protocol) or FTP (File Transfer Protocol) addresses. Keep in mind that some properties and methods do not support URLs.
The file or document names of images saved to Imaging 1.x Servers must not contain any of the following characters: Angle brackets (< >), asterisks (*), colons (:), question marks (?), quotes ("), slashes (\ /), or vertical bars (|).
Imaging 3.x Server documents are read-only.
The use of the UNC and URL formats is available with Imaging for Windows Professional Edition V2.0 only.
Imaging 1.x and 3.x Server access is available with Imaging for Windows Professional Edition V1.1 and V2.0 only.

Extender Properties, Methods, and Events

Extender properties, methods, and events are provided by the container of the Imaging control, rather than by the control itself.

Examples of Extender properties are:

- Name
- Index
- Top

Examples of Extender methods are:

- Drag
- SetFocus
- ShowWhatsThis

Examples of Extender events are:

- Click
- DragDrop
- MouseMove

Some Extender properties, methods, and events are standard for all containers, others are provided only by specific containers.

Refer to the documentation that came with your development environment for more information about Extender properties, methods, and events.

Image Edit Extender Properties

This section lists the Extender properties that are available when the Image Edit control is drawn on a Visual Basic form.

Name	Description
Container	Returns the container of an object.
DataBindings	Returns the DataBindings collection object containing the available bindable properties.
DataChanged	Returns or sets a value indicating that the data in the bound Image Edit control has been changed by some process other than that of retrieving data from the current record.
DataField	Returns or sets a value that binds the Image Edit control to a field in the current record.
DragIcon	Returns or sets the mouse pointer icon in a drag-and-drop operation.
DragMode	Returns or sets the drag mode (manual or automatic).
Height	Returns or sets the height of an object.

Name	Description
HelpContextID	Returns or sets the Help context ID for an object.
Index	Returns or sets the subscript value of a control in an array of controls.
Left	Returns or sets the distance between the left edge of an object and the left edge of its container.
Name	Returns the name of an object.
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
TabIndex	Returns or sets the tab order of an object within its parent.
TabStop	Returns or sets whether the Tab key can be used to move focus to an object.
Tag	Returns or sets ancillary data.
ToolTipText	Returns or sets a tool tip.
Top	Returns or sets the distance between the top edge of an object and the top edge of its container.
Visible	Returns or sets whether an object is visible or hidden.
WhatsThisHelpID	Returns or sets the context-sensitive help ID for an object.
Width	Returns or sets the width of an object.

Refer to the Visual Basic on-line help for more information about these properties.

Image Edit Extender Methods

This section lists the Extender methods that are available when the Image Edit control is drawn on a Visual Basic form.

Name	Description
Drag	Begins, ends, or cancels a drag operation.
Move	Moves an object.
SetFocus	Gives focus to the object specified.
ShowWhatsThis	Displays the help topic corresponding to the value of the WhatsThisHelpID property.
Zorder	Places an object at front or back of the z-order within its graphical level.

Refer to the Visual Basic on-line help for more information about these methods.

Image Edit Extender Events

This topic lists the Extender events that are available when the Image Edit control is drawn on a Visual Basic form.

Name	Description
Click	Fires when a user presses and then releases a mouse button over an object
DblClick	Fires when a user presses and releases a mouse button twice over an object.
DragDrop	Fires when a drag-and-drop operation is completed.
DragOver	Fires when a drag-and-drop operation is in progress.
Error	Fires when an error occurs.
GotFocus	Fires when an object receives focus.
KeyDown	Fires when a user presses a key while the object has focus.
KeyPress	Fires when a user presses and then releases a key while the object has focus.
KeyUp	Fires when a user releases a key while the object has focus.
LostFocus	Fires when an object loses focus.
MouseDown	Fires when a user presses a mouse button.
MouseMove	Fires when a user moves the mouse.
MouseUp	Fires when a user releases a mouse button.

Refer to the Visual Basic on-line help for more information about these events.

IATB Extender Properties

This section lists the Extender properties that are available when the Image Annotation Tool Button control is drawn on a Visual Basic form.

Name	Description
Container	Returns the container of an object.
DataBindings	Returns the DataBindings collection object containing the available bindable properties.
DragIcon	Returns or sets the mouse pointer icon in a drag-and-drop operation.
DragMode	Returns or sets the drag mode (manual or automatic).
Height	Returns or sets the height of an object.
HelpContextID	Returns or sets the Help context ID for an object.
Index	Returns or sets the subscript value of a control in an array of controls.
Left	Returns or sets the distance between the left edge of an object and the left edge of its container.
Name	Returns the name of an object.

Name	Description
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
TabIndex	Returns or sets the tab order of an object within its parent.
TabStop	Returns or sets whether the Tab key can be used to move focus to an object.
Tag	Returns or sets ancillary data.
ToolTipText	Returns or sets a tool tip.
Top	Returns or sets the distance between the top edge of an object and the top edge of its container.
Visible	Returns or sets whether an object is visible or hidden.
WhatsThisHelpID	Returns or sets the context-sensitive help ID for an object.
Width	Returns or sets the width of an object.

Refer to the Visual Basic on-line help for more information about these properties.

IATB Extender Methods

Description This section lists the Extender methods that are available when the Image Annotation Tool Button control is drawn on a Visual Basic form.

Name	Description
Drag	Begins, ends, or cancels a drag operation.
Move	Moves an object.
SetFocus	Gives focus to the object specified.
ShowWhatsThis	Displays the help topic corresponding to the value of the WhatsThisHelpID property.
Zorder	Places an object at the front or back of the z-order within its graphical level.

Refer to the Visual Basic on-line help for more information about these methods.

IATB Extender Events

This section lists the Extender events that are available when the Image Annotation Tool Button control is drawn on a Visual Basic form.

Name	Description
Click	Fires when a user presses and then releases a mouse button over an object.
DragDrop	Fires when a drag-and-drop operation is completed.
DragOver	Fires when a drag-and-drop operation is in progress.
Error	Fires when an error occurs.
GotFocus	Fires when an object receives focus.
KeyDown	Fires when a user presses a key while the object has focus.
KeyPress	Fires when a user presses and then releases a key while the object has focus.
KeyUp	Fires when a user releases a key while the object has focus.
LostFocus	Fires when an object loses focus.
MouseDown	Fires when a user presses a mouse button.
MouseMove	Fires when a user moves the mouse.
MouseUp	Fires when a user releases a mouse button.

Refer to the Visual Basic on-line help for more information about these events.

RGB Format

RGB (Red Green Blue) values range from 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF).

Annotation Styles

You can assign the same annotation type to two or more IATB controls. When you do, you can set some of the properties of each control to different values to create distinctive annotation *styles*.

Example

You may want to include two buttons (controls) in your application that let users draw two different types of Attach-a-Note annotations.

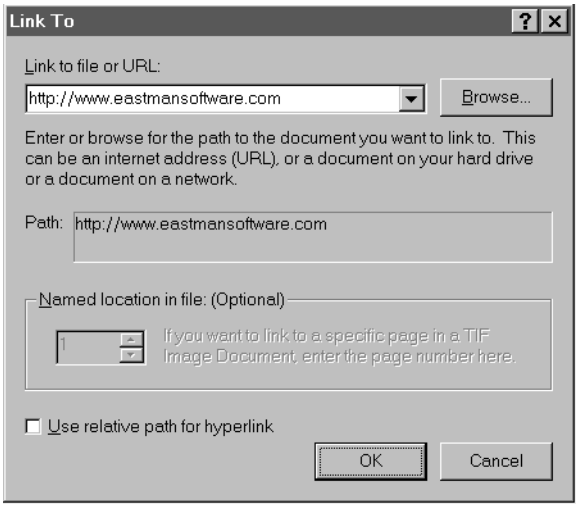
While both controls are assigned to the same annotation type, each is set to different AnnotationBackColor and AnnotationFontColor property values, resulting in two distinct annotation styles. One button lets users draw Attach-a-Note annotations with a yellow background and black text, while the other button lets users draw Attach-a-Note annotations with a red background and white text.

Annotation Types

The Image Edit and Image Annotation controls support the following annotation types:

Type	Description
Attach-a-Note	Enters text into a background rectangle on an image.
Filled Rectangle	Covers a portion of an image. Highlights 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 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 (see below) 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).
OCR Zones	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 lines Highlights text when drawn using the transparent line style.
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.

Note: Hyperlink and OCR Zones are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.



File Types

Imaging ActiveX controls support the following file types:

AWD — The AWD (At Work Document) file type is used by Microsoft Fax. AWD supports the black-and-white page type only; its image documents can contain multiple image pages. AWD image documents are compressed using Microsoft's RBA compression.

BMP — The BMP (bitmap) file type is used by many Windows applications. A Bitmap image is a representation of a graphic image, displayed by setting individual pixels to white, black, or a color. BMP supports all page types; its image documents can contain only a single image page. Compression of BMP image document files is not supported.

DCX (read-only) — The DCX file type was made popular by ZSoft Corporation's PC Paintbrush program and continues to be used by many graphics programs. DCX supports all page types; its image documents can contain multiple image pages. DCX image document files are compressed using PCX compression.

GIF (read-only) — The GIF (Graphics Interchange Format) file type was made popular by CompuServe and continues to be used by many graphics programs. GIF supports the following page types: Gray4, Gray8, Palettized4, and Palettized8. Its image documents can contain only a single image page. GIF image document files are compressed using LZW compression.

JPG (read-only) — The file format for Joint Photographics Experts Group (JPEG) images. JPG supports the following page types: Palettized8 and RGB24. Its image documents can contain only a single image page. JPG image document files are compressed using JPEG compression.

PCX (read-only) — The PCX file type was made popular by ZSoft Corporation's PC Paintbrush program and continues to be used by many graphics programs. PCX supports all page types; its image documents can contain only a single image page. PCX image document files are compressed using PCX compression.

TIFF — The TIFF (Tagged Image File Format) file type is used by many applications. TIFF supports all page types. Its image documents can contain multiple image pages, and can store Summary property information and image annotation data separate from the actual image data. TIFF supports several compression types.

WIFF (read-only) — The WIFF (Wang Image File Format) file type is used by Wang Integrated Image Systems. WIFF supports the black-and-white page type; its image documents can contain multiple image pages. WIFF supports several compression types.

XIF (read-only) — XIF is the file type for Xerox image documents. When read by Imaging, XIF supports the black-and-white page type; its image documents can contain multiple image pages. XIF image document files are compressed using Xerox-proprietary compression.

Note: AWD is not available with Imaging for Windows NT 4.0. GIF and WIFF are available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0, as well as with Imaging for Windows 98.

Pixel

Also known as a picture element, a pixel is a dot that represents the smallest graphic unit of measure on a screen. A pixel is a screen-dependent unit of measure, meaning that its size varies with the display system and resolution.

8

Image Admin Control

This chapter describes what developers and users can do using the Image Admin control, as well as the properties and methods that are available with this control. In addition to the properties and methods described in this chapter, the following properties and methods, which are described in Chapter 7, apply to the Image Admin control:

- StatusCode property
- AboutBox method
- GetVersion method

In This Chapter

What the Image Admin Control Lets You Do

The Image Admin control lets you — the application developer — add image file management functions to applications using the Imaging ActiveX controls.

The Image Admin control can manage files that reside locally or on a local area network (LAN), and documents that reside on Eastman Software's Imaging 1.x and 3.x servers. Read/Write access is supported for Imaging 1.x documents, and Read-only access for Imaging 3.x documents. File and Print common dialog boxes support specific image file options as well as general directory and file functions.

What the Image Admin Control Lets Your Users Do

Depending on how you design and code your application, the Image Admin control lets your users

- Create and delete directories.
- Create and delete image files.
- Open, save, and print image documents.
- Append, insert, and replace pages in multipage image documents and 1.x server documents.
- Manage File (Summary) properties.

Author Property

Description Returns or sets the Author property of the current image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 98 (read only)

Imaging for Windows 95

Imaging for Windows NT 4.0

Usage *object*.**Author**[=*value*]

Data Type String.

Remarks This property pertains to local/redirected TIFF files, and 1.x files (not documents).

Setting this property does not write the change to disk. To do so, you must call the **SetFileProperties** method, or, choose OK in the dialog box created by the **ShowFileProperties** method.

The current setting of this property can be displayed and changed from the Summary tab on the dialog box created by the **ShowFileProperties** method (see page 673 for an example of this dialog box).

See Also Comments property, Keywords property, SetFileProperties method, ShowFileProperties method, Subject property, Title property, Summary Properties.

Author Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image you are going to set properties for.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

Author Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image you are going to set properties for.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

Browse1xReturnedPath

Description Specifies the path returned by the **Browse1x** method.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Browse1xReturnedPath**[=*value*]

Data Type String.

Remarks This property is read-only.

See Also Browse1x method, Browse1xReturnedType property.

Browse1xReturnedType Property

Description Specifies the type of path returned by the **Browse1x** method.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Browse1xReturnedType**[=*value*]

Data Type String.

Remarks This property is read-only.

Path types are:

- IFS (imaging file system) that displays a directory structure.
- DM (document management system) that displays nodes such as database, cabinet, drawer, or folder.

See Also Browse1x method, Browse1xReturnedPath property.

CancelError Property

Returns or sets whether an error will be returned when a user chooses Cancel from the following dialog boxes: File, Print, LoginToServer, or 1.x Server Options.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**CancelError**[= {*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Causes an error to be returned when the user chooses Cancel from either the File or Print dialog boxes.
False	No error is returned when the user chooses Cancel.

Remarks When a user chooses Cancel from the File or Print dialog boxes, and this property is set to True, an error is returned by the **ShowFileDialog** or **ShowPrintDialog** methods, respectively.

See Also ShowFileDialog method, ShowPrintDialog method.

CancelError Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print
    'dialog box (for example, start page, end page, and so on).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥  ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
```

```
PrintErr:
    'User pressed the cancel button.
       Exit Sub
End Sub
```

CancelError Example – VC++

This example uses the **ShowPrintDialog** method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    // Use Try and Catch around ShowPrintDialog to catch Cancel error. If
    // Cancel is pressed then the Catch will capture the error exception.
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥  evt, evt);
}
```

Comments Property

Description Specifies the Comments property of the current image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98 (read only)
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**Comments**[=*value*]

Data Type String.

Remarks This property pertains to local/redirected TIFF files, and 1.x files (not documents).

Setting this property does not write the change to disk. To do so, you must call the **SetFileProperties** method, or, choose OK in the dialog box created by the **ShowFileProperties** method.

The current setting of this property can be displayed and changed from the Summary tab on the dialog box created by the **ShowFileProperties** method (see page 673 for an example of this dialog box).

See Also Author property, Keywords property, SetFileProperties method, ShowFileProperties method, Subject property, Title property, Summary Properties (see page 681).

Comments Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the ShowFileProperties method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image you are going to set properties for.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

Comments Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image you are going to set properties for.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

CompressionInfo Property

Description Returns the bitwise compression options set for the current image.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**CompressionInfo**[=*value*]

Data Type Long.

Setting	Description
Note:	Values 1 to 32 are not used with JPEG compression.
0	No flags or options are set. This is the default value if the Image property is empty.
1	EOL— include or expect standard end of line bit sequences.
2	Packed lines — data is not byte aligned.
4	Prefixed EOLs — include or expect prefixed end of line bit sequences.
8	Compressed bit order, left to right.
16	Expanded bit order, left to right.

Setting	Description
32	Negate — indicates data is stored with black bits equal to 0 (zero).
Note:	Values 64 to 16384 pertain only to JPEG compression.
64	Low resolution, high quality
128	Low resolution, medium quality
256	Low resolution, low quality
512	Medium resolution, high quality
1024	Medium resolution, medium quality
2048	Medium resolution, low quality
4096	High resolution, high quality
8192	High resolution, medium quality
16384	High resolution, low quality

Remarks The value returned by this property is for the image page specified by the **PageNumber** property. If no image is specified by PageNumber, the value returned is 0 (zero).

This property is read-only.

See Also CompressionType property, Image property (Admin control), PageNumber property.

CompressionType Property

Description Returns the compression type of the current image.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

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

Data Type CompressionTypeValue (Integer).

Constant	Setting	Description
CompTypeUnk	0	Unknown
CompTypeNone	1	No compression
CompTypeGroup3	2	Group 3 1D FAX
CompTypeGroup3Huff	3	Group 3 Modified Huffman

Constant	Setting	Description
CompTypePacked	4	PackBits
CompTypeGroup4	5	Group 4 2D FAX
CompTypeJPEG	6	JPEG
	7	Reserved
CompTypeGroup32DFax	8	Group 3 2D FAX
CompTypeLZW	9	LZW

Remarks The value returned by this property is for the image page specified by the **PageNumber** property. If the **Image** property is empty, the value returned is 0 (zero).

This property is read-only.

See Also CompressionInfo property, Image property (Admin control), PageNumber property.

CompressionType Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
        Case 5
            strImgFileType = "DCX"
        Case 6
            strImgFileType = "JPG"
        Case 7
            strImgFileType = "XIF"
        Case 8
            strImgFileType = "GIF"
```

```
            Case 9
                strImgFileType = "WIF"
        End Select
        frmInfo.lblImgInfo(0).Caption = strImgFileType
        'Read the CompressionType property and translate the value to a 'corresponding string.
        Select Case ImgAdmin1.CompressionType
            Case 0
                strImgCompType = "Unknown"
            Case 1
                strImgCompType = "No Compression"
            Case 2
                strImgCompType = "Group3(1D)"
            Case 3
                strImgCompType = "Group3(Modified Huffman)"
            Case 4
                strImgCompType = "PackBits"
            Case 5
                strImgCompType = "Group4(2D)"
            Case 6
                strImgCompType = "JPEG"
            Case 7
                strImgCompType = "RBA"
            Case 8
                strImgCompType = "Group3(2D)"
            Case 9
                strImgCompType = "LZW"
        End Select
        frmInfo.lblImgInfo(1).Caption = strImgCompType
        'Read the PageType property and translate the value to a
        'corresponding string.
        Select Case ImgAdmin1.PageType
            Case 0
                strImgPageType = "Unknown"
            Case 1
                strImgPageType = "Black and White"
            Case 2
                strImgPageType = "4 bit grayscale"
            Case 3
                strImgPageType = "8 bit grayscale"
            Case 4
                strImgPageType = "4 bit palettized"
            Case 5
                strImgPageType = "8 bit palettized"
            Case 6
                strImgPageType = "24 bit RGB"
            Case 7
                strImgPageType = "24 bit BGR"
    End Select
        frmInfo.lblImgInfo(6).Caption = strImgPageType
        'Determine the dimensions of the image page.
        frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
        frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
```

```
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

CompressionType Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
    case 6:
         strImgFileType = "JPG";
               break;
    case 7:
         strImgFileType = "XIF";
               break;
```

```
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
```

```
    // Read the PageType property and translate the value to a
    // corresponding string.
    switch (ImgAdmin1.GetPageType())
    {
    case 0:
         strImgPageType = "Unknown";
              break;
    case 1:
         strImgPageType = "Black and White";
              break;
    case 2:
         strImgPageType = "4 bit grayscale";
              break;
    case 3:
         strImgPageType = "8 bit grayscale";
              break;
    case 4:
         strImgPageType = "4 bit palettized";
              break;
    case 5:
         strImgPageType = "8 bit palettized";
              break;
    case 6:
         strImgPageType = "24 bit RGB";
              break;
    case 7:
         strImgPageType = "24 bit BGR";
              break;
    }
    frmInfo.m_PageType = strImgPageType;
    // Determine the dimensions of the image page.
    long lInfo = ImgAdmin1.GetImageHeight();
    frmInfo.m_Height.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageWidth();
    frmInfo.m_Width.Format("%i",lInfo);
    // Determine the X and Y resolution of the image page.
    lInfo = ImgAdmin1.GetImageResolutionX();
    frmInfo.m_XRes.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageResolutionY();
    frmInfo.m_YRes.Format("%i",lInfo);
    // Determine the number of pages in the file.
    lInfo = ImgAdmin1.GetPageCount();
    frmInfo.m_PageCount.Format("%i",lInfo);
    // Show the form with the image attributes.
    frmInfo.DoModal();
}
```

DefaultExt Property

Description Returns or sets the default file extension in the dialog box created by the **ShowFileDialog** method (see page 670 for an example of this dialog box).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DefaultExt**[=*extension*]

Data Type String.

Remarks The extension of the currently selected filter overrides the default extension.

The default extension is used only when there is no extension associated with the filter (for example, 'All files *.*').

The first three characters of the string you specify are used as the file extension when the user does not specify a file extension.

See Also ShowFileDialog method.

DialogTitle Property

Description Returns or sets the title used in the dialog box created by the **ShowFileDialog** method (see page 670 for an example of this dialog box).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DialogTitle**[=*title*]

Data Type String.

Remarks This property is used in conjunction with the **ShowFileDialog** method.

If a string is not specified, the default is used — either Open or Save As.

See Also ShowFileDialog method.

DialogTitle Example – VB

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
Private Sub cmdDisplay_Click()
    'Initialize the Image property to check for Cancel pressed. When the
    'dialog box is shown, check for a value in the Image property.
    ImgAdmin1.Image = ""
    'Specify a dialog title and a list filter rather than using the
    'defaults.
    ImgAdmin1.DialogTitle = "Select Image for Display"
    ImgAdmin1.Filter = "All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥    TIFF Files (*.tif)|*.tif|All Files (*.*)|*.*|"
    'Default filter will be TIFF (fourth item).
    ImgAdmin1.FilterIndex = 4
    'Set the InitDir property to the preferred directory.
    ImgAdmin1.InitDir = "C:\images"
    ImgAdmin1.CancelError = False
    'Note: If you have Server Access software installed this will
    'automatically be enabled in the Open dialog box after the user
    'authenticates.
    ImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd
    'Determine if a file was selected or cancel was pressed.
    If ImgAdmin1.Image = "" Then Exit Sub
    'Set the image properties in the Image Edit and Thumbnail
    'controls to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Display the image in the Image Edit and Thumbnail control.
    ImgEdit1.Display
End Sub
```

DialogTitle Example – VC++

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
void CAdmin1Dlg::OnFileopen()
{
    // Initialize Image property in order to check for Cancel pressed. After // the dialog
     box is shown, check for a value in the image property.
    ImgAdmin1.SetImage ("" );
    // Specify a dialog title and a list filter rather than using the
    // defaults.
    ImgAdmin1.SetDialogTitle ("Select Image for Display");
    ImgAdmin1.SetFilter ("All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥    Tiff Files (*.tif)|*.tif|All Files (*.*)|*.*|");
    // Default filter will be TIFF (fourth item).
    ImgAdmin1.SetFilterIndex (4);
    // Set the InitDir property to the preferred directory.
```

```
    ImgAdmin1.SetInitDir ("C:\\images");
    ImgAdmin1.SetCancelError (FALSE);
    // Note: If you have Server Access software installed this will
    // automatically be enabled in the
    // Open dialog box after the user authenticates.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);       // OpenDlg
    // Determine if a file was selected or cancel was pressed.
    if (ImgAdmin1.GetImage() =="")
            return;
    // Set the image properties in the ImgEdit and Thumbnail control
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage (ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage (ImgAdmin1.GetImage());
    // Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display();
}
```

Domain Property

Data Type Returns or sets the name of the server running 3.x services.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Domain**[=*name*]

Data Type String.

Remarks The name of the 3.x server last accessed is the default.

The **LoginToServer** method authenticates the user to the server specified by this property.

See Also LoginToServer method.

FileStgLoc1x Property

Description Returns or sets the location where new files will be placed when creating 1.x server documents.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**FileStgLoc1x**[=*string*]

Data Type String.

Remarks Depending on the application, this location can be used by the Image Edit Control's **SaveAs** and **SavePage** methods, and also by the Image Admin control's **Append**, **Replace**, and **Insert** methods. These methods will cause a new image file with a unique filename to be created in the following situations.

- A page from a local file is being inserted, appended, or saved to a 1.x document.
- The **ForceFileLinking1x** property is set to False, and a page from a 1.x document or file is being inserted, appended, replaced, or saved to another 1.x document.

Use this property to set the initial value in the "File location for document pages" edit box in the 1.x Server Options dialog box (see page 669). The user can accept or change this value.

The 1.x Server Options dialog box is created by the **Show1xServerOptDlg** method.

See Also Append method, ForceFileLinking1x property, Insert method, Replace method, SaveAs method, SavePage method, Show1xServerOptDlg method.

FileStgLoc1x Example – VB

This example shows how to make a copy of a file being inserted into a 1.x document.

```
Private Sub cmdLink_Click()
    'Request file being inserted into a document be copied to a new file in
    'a location to be specified.
    ImgAdmin1.ForceFileLinking1x = False
    'To specify location of the new files, use the FileStgLoc1x property.
    ImgAdmin1.FileStgLoc1x = "Image://srvrname\new_images:"
    'Second page of INSERTED DOC, will be inserted before page 3 of ORIG 'DOC. Associated
     image files will be copied to new randomly generated
    'file names in directory specified.
    ImgAdmin1.Image = "Image://srvrname\doc_db:\CAB\DRAWER\FOLDER\
    ➥   ORIG DOC"
```

```
    ImgAdmin1.Insert "Image://srvrname\doc_db:\CAB\DRAWER\FOLDER\
    ➥   INSERTED DOC", 2, 3
End Sub
```

FileStgLoc1x Example – VC++

This example makes a copy of a file being inserted into a 1.x document.

```
void CAdmin1Dlg::OnFilelink()
{
    // Request file being inserted into a document be copied to a new file
    // in a location to be specified.
    ImgAdmin1.SetForceFileLinking1x(FALSE);
    // To specify location of the new files, use the FileStgLoc1x property.
    ImgAdmin1.SetFileStgLoc1x("Image://srvrname\\new_images:");
    // Second page of INSERTED DOC, will be inserted before page 3 of ORIG
    // DOC. Associated image files will be copied to new randomly generated
    // file names in directory specified.
    VARIANT evt;
    evt.vt = VT_ERROR;    // set to error for optional parameter
    ImgAdmin1.SetImage("Image://srvrname\\doc_db:\\CAB\\DRAWER\\
    ➥   FOLDER\\ORIG DOC");
    ImgAdmin1.Insert("Image://srvrname\\doc_db:\\CAB\\DRAWER
    ➥   \\FOLDER\\INSERTED DOC", 2, 3,1,evt,evt);
}
```

FileType Property

Description Returns the file format of the image file. For more information on file types, see page 679.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**FileType**[=*value*]

Data Type FileTypeValue (Integer)

Constant	Setting	Description
FileTypeUnk	0	Unknown
FileTypeTIFF	1	TIFF
FileTypeAWD	2	AWD (Windows 95 and 98 only)
FileTypeBMP	3	Bitmap (BMP)

Constant	Setting	Description
FileTypePCX	4	PCX
FileTypeDCX	5	DCX
FileTypeJPEG	6	JPEG
FileTypeXIF	7	XIF
FileTypeGIF	8	GIF
FileTypeWIFF	9	WIF

Remarks The file format is determined by the **Image** property.

This property is read-only.

See Also Image property, File Types (see page 679).

FileType Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
        Case 5
            strImgFileType = "DCX"
        Case 6
            strImgFileType = "JPG"
        Case 7
            strImgFileType = "XIF"
        Case 8
            strImgFileType = "GIF"
```

```
            Case 9
                strImgFileType = "WIF"
        End Select
        frmInfo.lblImgInfo(0).Caption = strImgFileType
        'Read the CompressionType property and translate the value to a
        'corresponding string.
        Select Case ImgAdmin1.CompressionType
            Case 0
                strImgCompType = "Unknown"
            Case 1
                strImgCompType = "No Compression"
            Case 2
                strImgCompType = "Group3(1D)"
            Case 3
                strImgCompType = "Group3(Modified Huffman)"
            Case 4
                strImgCompType = "PackBits"
            Case 5
                strImgCompType = "Group4(2D)"
            Case 6
                strImgCompType = "JPEG"
            Case 7
                strImgCompType = "RBA"
            Case 8
                strImgCompType = "Group3(2D)"
            Case 9
                strImgCompType = "LZW"
        End Select
        frmInfo.lblImgInfo(1).Caption = strImgCompType
        'Read the PageType property and translate the value to a
        'corresponding string.
        Select Case ImgAdmin1.PageType
            Case 0
                strImgPageType = "Unknown"
            Case 1
                strImgPageType = "Black and White"
            Case 2
                strImgPageType = "4 bit grayscale"
            Case 3
                strImgPageType = "8 bit grayscale"
            Case 4
                strImgPageType = "4 bit palettized"
            Case 5
                strImgPageType = "8 bit palettized"
            Case 6
                strImgPageType = "24 bit RGB"
            Case 7
                strImgPageType = "24 bit BGR"
    End Select
        frmInfo.lblImgInfo(6).Caption = strImgPageType
        'Determine the dimensions of the image page.
        frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
```

```
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

FileType Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
    case 6:
         strImgFileType = "JPG";
               break;
    case 7:
         strImgFileType = "XIF";
               break;
```

```
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
case 0:
     strImgPageType = "Unknown";
          break;
case 1:
     strImgPageType = "Black and White";
          break;
```

```
        case 2:
             strImgPageType = "4 bit grayscale";
                  break;
        case 3:
             strImgPageType = "8 bit grayscale";
                  break;
        case 4:
             strImgPageType = "4 bit palettized";
                  break;
        case 5:
             strImgPageType = "8 bit palettized";
                  break;
        case 6:
             strImgPageType = "24 bit RGB";
                  break;
        case 7:
             strImgPageType = "24 bit BGR";
                  break;
        }
        frmInfo.m_PageType = strImgPageType;
        // Determine the dimensions of the image page.
        long lInfo = ImgAdmin1.GetImageHeight();
        frmInfo.m_Height.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageWidth();
        frmInfo.m_Width.Format("%i",lInfo);
        // Determine the X and Y resolution of the image page.
        lInfo = ImgAdmin1.GetImageResolutionX();
        frmInfo.m_XRes.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageResolutionY();
        frmInfo.m_YRes.Format("%i",lInfo);
        // Determine the number of pages in the file.
        lInfo = ImgAdmin1.GetPageCount();
        frmInfo.m_PageCount.Format("%i",lInfo);
        // Show the form with the image attributes.
        frmInfo.DoModal();
}
```

Filter Property

Description Returns or sets the list of file filters displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Filter**[=*filter*]

Data Type String.

Remarks Filters are displayed in the dialog box created by the **ShowFileDialog** method. See page 673 for an example of this dialog box.

Each filter type consists of two strings separated by, and ending with, a pipe symbol (|). The first string contains a description of the filter, such as 'Tiff files'. The second string contains the file extension, such as '*.tif'. Be sure not to include spaces immediately before and after the pipe symbol, or they become part of the string. For example:

All image files|*.tif;*.bmp|TIFF files|*.tif|BMP files|*.bmp|

If no filter type is provided, the system supplies a default filter.

See Also ShowFileDialog method.

Filter Example – VB

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
Private Sub cmdDisplay_Click )
    'Initialize the Image property to check for Cancel pressed.
    'When the dialog box is shown, check for a value in the Image property.
    ImgAdmin1.Image = ""
    'Specify a dialog title and a list filter rather than using the
    'defaults.
    ImgAdmin1.DialogTitle = "Select Image for Display"
    ImgAdmin1.Filter = "All Image Files|*.bmp; *.jpg; *.tif|
    ➥   Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥   TIFF Files (*.tif)|*.tif|All Files (*.*)|*.*|"
    'Default filter will be TIFF (fourth item).
    ImgAdmin1.FilterIndex = 4
    'Set the InitDir property to the preferred directory.
    ImgAdmin1.InitDir = "C:\images"
    ImgAdmin1.CancelError = False
    'Note: If you have Server Access software installed this will
    'automatically be enabled in the
    'Open dialog box after the user authenticates.
    ImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd
    'Determine if a file was selected or cancel was pressed.
    If ImgAdmin1.Image = "" Then Exit Sub
    'Set the image properties in the Image Edit and Thumbnail
    'controls to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Display the image in the Image Edit and Thumbnail control.
    ImgEdit1.Display
End Sub
```

Filter Example – VC++

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
void CAdmin1Dlg::OnFileopen()
{
    // Initialize Image property in order to check for Cancel pressed. After
    // the dialog box is shown, check for a value in the image property.
    ImgAdmin1.SetImage ("" );
    // Specify a dialog title and a list filter rather than using the
    // defaults.
    ImgAdmin1.SetDialogTitle ("Select Image for Display");
    ImgAdmin1.SetFilter ("All Image Files|*.bmp; *.jpg; *.tif|
    ➥   Bitmap Files (*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥   Tiff Files (*.tif)|*.tif|All Files (*.*)|*.*|");
    // Default filter will be TIFF (fourth item).
    ImgAdmin1.SetFilterIndex (4);
    // Set the InitDir property to the preferred directory.
    ImgAdmin1.SetInitDir ("C:\\images");
    ImgAdmin1.SetCancelError (FALSE);
    // Note: If you have Server Access software installed this will
    // automatically be enabled in the
    // Open dialog box after the user authenticates.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);       // OpenDlg
    // Determine if a file was selected or cancel was pressed.
    if (ImgAdmin1.GetImage() =="")
            return;
    // Set the image properties in the ImgEdit and Thumbnail control
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage (ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage (ImgAdmin1.GetImage());
    // Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display();
}
```

FilterIndex Property

Description Returns or sets which of the file filters is highlighted when the dialog box first opens.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**FilterIndex**[=*index*]

Data Type Long.

Remarks The file Open or Save As dialog box is created by the **ShowFileDialog** method. See page 670 for an example of this dialog box.

If no filters are specified by the **Filter** property and the **FilterIndex** property is set to 0 (zero), then a default filter will be highlighted. If a different filter is selected, the index still remains set to 0.

If a value was set for the Filter property, and a different filter is selected, the index will be updated to reflect the selected filter.

The default value is 0 (zero).

See Also Filter property, ShowFileDialog method.

FilterIndex Example – VB

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
Private Sub cmdDisplay_Click()
    'Initialize the Image property to check for Cancel pressed.
    'When the dialog box is shown, check for a value in the Image property.
    ImgAdmin1.Image = ""
    'Specify a dialog title and a list filter rather than using the
    'defaults.
    ImgAdmin1.DialogTitle = "Select Image for Display"
    ImgAdmin1.Filter = "All Image Files |*.bmp; *.jpg; *.tif|
    ➥   Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥   TIFF Files (*.tif)|*.tif|All Files (*.*)|*.*|"
    'The default filter will be TIFF (fourth item).
    ImgAdmin1.FilterIndex = 4
    'Set the InitDir property to the preferred directory.
    ImgAdmin1.InitDir = "C:\images"
    ImgAdmin1.CancelError = False
    'Note: If you have Server Access software installed this will
    'automatically be enabled in the Open dialog box after the user 'authenticates.
    ImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd
    'Determine if a file was selected or cancel was pressed.
    If ImgAdmin1.Image = "" Then Exit Sub
    'Set the image properties in the Image Edit and Thumbnail
    'controls to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Display the image in the Image Edit and Thumbnail control.
    ImgEdit1.Display
End Sub
```

FilterIndex Example – VC++

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
void CAdmin1Dlg::OnFileopen()
{
    // Initialize Image property in order to check for Cancel pressed.
    // After the dialog box is shown, check for a value in the
    // image property.
    ImgAdmin1.SetImage ("" );
    // Specify a dialog title and a list filter rather than using the
    // defaults.
    ImgAdmin1.SetDialogTitle ("Select Image for Display");
    ImgAdmin1.SetFilter ("All Image Files |*.bmp; *.jpg; *.tif|
    ➥   Bitmap Files (*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|
    ➥   TIFF Files (*.tif)|*.tif|All Files (*.*)|*.*|");
    // Default filter will be TIFF (fourth item).
    ImgAdmin1.SetFilterIndex (4);
    // Set the InitDir property to the preferred directory.
    ImgAdmin1.SetInitDir ("C:\\images");
    ImgAdmin1.SetCancelError (FALSE);
    // Note: If you have Server Access software installed this will
    // automatically be enabled in the
    // Open dialog box after the user authenticates.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);       // OpenDlg
    // Determine if a file was selected or cancel was pressed.
    if (ImgAdmin1.GetImage() =="")
            return;
    // Set the image properties in the ImgEdit and Thumbnail control
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage (ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage (ImgAdmin1.GetImage());
    // Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display();
}
```

Flags Property

Description Returns or sets initialization flags for the File (Open or Save As) dialog box (see page 670) or the Print dialog box (see page 675). These dialog boxes are created by the **ShowFileDialog** and **ShowPrintDialog** methods.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Flags**[=*value*]

Data Type Long.

Values for ShowFileDialog

Value	Description
&H400&	Indicates that the extension of the returned file name is different from the extension specified by the DefaultExt property. This flag is not set if the DefaultExt property is empty, if the extensions match, or if the file has no extension. This flag value can be checked upon returning from the dialog box.
&H4&	Hides the Read-Only check box.
&H8&	Forces the dialog box to set the current directory to what it was when the dialog box was invoked.
&H8000&	Specifies that the returned file will not have the Read-Only attribute set and will not be in a write-protected directory.
&H100&	Specifies that the common dialog box allows invalid characters in the returned file name.
&H1&	Causes the Read Only check box to be initially checked when the dialog box is created. This flag also indicates the state of the Read Only check box when the dialog box is closed.
&H10&	Causes the dialog box to display the Help button. If this option is specified, the parameter hParentWnd in the ShowFileDialog method must be set to null.
&H40&	Causes the dialog box to display the preview options and the Find button.

Values for ShowPrintDialog

Value	Description
&H80000&	Disables the Print to File check box.
&H100000&	The Print to File check box is not displayed.
&H80&	Prevents a warning message from being displayed when there is no default printer.
&H4&	Disables the Selection button.

Remarks The flags are bitwise and can be OR'd together.

The Help button in the dialog box is not shown unless a Help file is specified by the **HelpFile** property.

See Also HelpFile property, ShowFileDialog method.

Flags Example – VB

This example uses the Flags property to define options for the Open dialog box; hides the Read-Only check box and enables preview image window and the find button.

```
Private Sub cmdSetFlags_Click()
    'Private Sub mnuSetFlags_Click
    Const OFN_HIDEREADONLY = &H4&
    Const FINDANDPREVIEW = &H40&
    'Bitwise OR the flags together.
    ImgAdmin1.Flags = OFN_HIDEREADONLY + FINDANDPREVIEW
    ImgAdmin1.ShowFileDialog OpenDlg
End Sub
```

Flags Example – VC++

This example uses the Flags property to define options for the Open dialog box; hides the Read-Only check box and enables preview image window and the find button.

```
{
    // Private Sub mnuSetFlags_Click
    #define FINDANDPREVIEW 0x40
    ImgAdmin1.SetFlags (OFN_HIDEREADONLY | FINDANDPREVIEW);
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);    // OpenDlg
}
```

ForceFileDeletion1x Property

Description Returns or sets whether the file referenced by a 1x document page is deleted when the document page itself is deleted. This property is used in conjunction with the **Delete** and **DeletePages** methods.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ForceFileDeletion1x**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Source file pages linked to the 1x document are deleted when a document page is deleted.
False	Source file pages linked to the 1x document are not deleted when a document page is deleted.

Remarks True is ignored in cases where a document contains several pages that are linked to the same source file, and not all of the pages are being deleted. In this case, the linked source file isn't deleted because other pages in the document are linked to it.

For example, a document named `anydoc.tif` contains 5 pages, and pages 1 and 4 are linked to `imgfile.tif`. If you set this property to True, and delete page 1 of `anydoc.tif`, `imgfile.tif` is not deleted because it is still needed by page 4.

Note: This exception is only true for pages within the *same* document.

Use this property to set the state of the checkbox "Delete File with pages" in the 1.x Server Options dialog box (see page 669). The user can accept or change this value. The dialog box is created by the **Show1xServerOptDlg** method.

See Also Delete method, DeletePages method, Show1xServerOptDlg method.

ForceFileLinking1x Property

Description Returns or sets how file pages being added to a document residing in a 1.x repository are handled.

Depending on the setting of the **ForceFileLinking1x** property, the Admin control's **Append**, **Replace**, and **Insert** methods can create copies of the source file pages in the 1.x file repository, or create links between the source file pages and the destination document.

The Image Edit control's **SaveAs** and **SavePage** methods can also use this property if the destination file is a 1.x document.

> **Note:** This property setting is ignored in cases where the source document/file is located on a local or redirected drive, and the Append, Insert, Replace, SaveAs, or SavePage method is called. In these situations, the source file pages are copied to the 1.x repository, given unique file names, and referenced by the destination document.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ForceFileLinking1x**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	The source file pages are linked to the destination document.
False (default)	Copies of the source file pages are created in the 1.x file repository, given unique file names, and linked to the destination document.

Remarks The source file/document (for the pages being appended, replaced, or inserted) can be located either on a local or redirected drive, or in the 1.x file/document repository — the destination document must reside in a 1.x repository and be managed by the Document Manager.In cases where both the source and destination documents reside in a 1.x repository, the property settings work in the following manner:

Setting = True — Source file pages are *linked* to a document, which means each time the source file changes, the destination document reflects the changes. You can link the same set of source pages to several documents so that information can be easily updated.

Setting = False — Copies of the source file pages are created in the location specified by the **FileStgLoc1x** property, and given unique file names. Each time pages are appended to or inserted into a destination document, a new set of copies are made. Changing the original source pages has no effect on the destination document because it references the *copy* of the source page.

Use this property to provide the initial value for the checkbox “Link files on reference” in the 1.x Server Options dialog box (see page 669). The user can accept or change this value.

The 1.x Server Options dialog box is created by the **Show1xServerOptDlg** method.

See Also Append method, FileStgLoc1x property, Insert method, Replace method, SaveAs method, Show1xServerOptDlg method.

ForceFileLinking1x Example – VB

The following examples demonstrate options for file linking/copying when performing insert/append functions using 1x server files and documents.

```
Private Sub cmdLink_Click()
'Example 1
    'Request file being inserted into a document be copied to a new file in
    'a location to be specified.
    ImgAdmin1.ForceFileLinking1x = False
    'To specify location of the new files, use the FileStgLoc1x property.
    ImgAdmin1.FileStgLoc1x = "Image://srvrname\new_images:"
    'Second page of INSERTED DOC, will be inserted before page 3 of
    'ORIG DOC. Associated image files will be copied to new randomly
    'generated file names in directory specified.
    ImgAdmin1.Image = "Image://srvrname\doc_db:\CAB\DRAWER\FOLDER\ORIG DOC"
    ImgAdmin1.Insert "Image://srvrname\doc_db:\CAB\DRAWER\FOLDER\
    ➥    INSERTED DOC", 2, 3

'Example 2
    'Request files associated with the page being appended reference the
    'original files. This results in multiple documents accessing the same
    'files -- could be useful to prevent rewrites of optical disk files.
    ImgAdmin1.ForceFileLinking1x = True
    'Pages 2, 3 and 4 of 5page.tif will be appended to ORIG DOC but they are
    'actually linked back to the file.
    ImgAdmin1.Image = "Image://srvrname\doc_db:\CAB\DRAWER\FOLDER\ORIG DOC"
    ImgAdmin1.Append "Image://srvrname\images:\5page.tif", 2, 3
End Sub
```

ForceFileLinking1x Example – VC++

The following examples demonstrate options for file linking/copying when performing insert/append functions using 1x server files and documents.

```
void CAdmin1Dlg::OnFilelink()
{
// Example 1
    // Request file being inserted into a document be copied to a new file
    // in a location to be specified.
    ImgAdmin1.SetForceFileLinking1x(FALSE);
    // To specify location of the new files, use the FileStgLoc1x property.
    ImgAdmin1.SetFileStgLoc1x("Image://srvrname\\new_images:");
    // Second page of INSERTED DOC, will be inserted before page 3 of
    // ORIG DOC. Associated image files will be copied to new randomly
```

```
    // generated file names in directory specified.
    VARIANT evt;
    evt.vt = VT_ERROR;    // set to error for optional parameter
    ImgAdmin1.SetImage("Image://srvrname\\doc_db:\\CAB\\DRAWER\\FOLDER
    ➥   \\ORIG DOC");
    ImgAdmin1.Insert("Image://srvrname\\doc_db:\\CAB\\DRAWER
    ➥   \\FOLDER\\INSERTED DOC", 2, 3,1,evt,evt);

// Example 2
    // Request files associated with the page being appended reference the
    // original files. This results in multiple documents accessing the same
    // files -- could be useful to prevent rewrites of optical disk files.
    ImgAdmin1.SetForceFileLinking1x(TRUE);
    // Pages 2, 3 and 4 of 5page.tif will be appended to
    // ORIG DOC but they are actually linked back to the file.
    ImgAdmin1.SetImage("Image://srvrname\\doc_db:\\CAB\\DRAWER
    ➥   \\FOLDER\\ORIG DOC");
    ImgAdmin1.Append("Image://srvrname\\images:\\5page.tif", 2,3,evt,evt);
}
```

ForceLowerCase1x Property

Description Returns or sets whether file names are forced to lower case.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ForceLowerCase1x**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	File names are forced to lower case when reading or writing 1.x files.
False (default)	File name case is not altered.

Remarks This property is used for backward compatibility with older, 16-bit clients that force directory path and file names to lower case when communicating with 1.x servers.

Setting this property provides the initial value of the “Force lower-case file names” checkbox in the 1.x Server Options dialog box (see page 669). The user can accept or change this value.

The 1.x Server Options dialog box is created by the **Show1xServerOptDlg** method.

See Also Show1xServerOptDlg method.

HelpCommand Property

Description Returns or sets the type of Help requested.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**HelpCommand**[=*value*]

Data Type Integer.

Setting	Description
&H1&	Displays Help for a particular context. When using this setting, you must also specify a context using the HelpContextID property.
&H3&	Displays the index of the specified Help file. An application should use this value only for a Help file with a single index.
&H4&	Displays Help for using the Help application itself.
&H5&	Sets the context specified by the HelpContext property as the current index for the Help file specified by the HelpFile property. This index remains current until the user accesses a different Help file. Use this value only for Help files with more than one index.
&H101&	Displays Help for a particular keyword. When using this setting, you must also specify a keyword using the HelpKey property.
&H105&	Brings up a search box in Help, with the key specified by the HelpKey property.

Remarks This property is used in conjunction with other Admin control help properties — **HelpContextId**, **HelpKey**, and **HelpFile** — to enable an application to use a custom Help file for the dialog box displayed by the **ShowFileDialog** method. See page 670 for an example of this dialog box.

See Also HelpContextId property, HelpFile property, HelpKey property, ShowFileDialog method.

HelpContextId Property

Description Returns or sets a Help context Id. Context Ids are assigned to objects to provide context-sensitive Help to your application.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**HelpContextId**[=*value*]

Data Type Integer.

Remarks This property is used in conjunction with other Admin control help properties — **HelpCommand**, **HelpKey**, and **HelpFile** — to enable an application to use a custom Help file for the file Open/Save As dialog box. The Open/Save As dialog box is displayed by the **ShowFileDialog** method. See page 670 for an example of this dialog box.

If this property is set to a value other than 0 (zero), and the HelpFile property contains a valid value, Help will be displayed when the Help button in the dialog box is clicked.

See Also HelpCommand property, HelpFile property, HelpKey property, ShowFileDialog method.

HelpFile Property

Description Returns or sets the name of a Help file to be used by the file Open/Save As dialog box. The Open/Save As dialog box is displayed by the **ShowFileDialog** method. See page 674 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**HelpFile**[=*filename*]

Data Type String.

Remarks This property is used in conjunction with the other Admin control help properties — **HelpCommand**, **HelpContextId**, and **HelpKey**.

See Also HelpCommand property, HelpKey property, ShowFileDialog method.

HelpKey Property

Description Returns or sets the keyword that identifies the requested Help topic.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
√ Imaging for Windows 95 and 98
√ Imaging for Windows NT 4.0

Usage *object*.**HelpKey**[=*keyword*]

Data Type String.

Remarks This property is used in conjunction with the **HelpCommand** and **HelpFile** properties. Together, they enable an application to use a custom Help file for the file Open/Save As dialog box displayed by the **ShowFileDialog** method. See page 670 for an example of this dialog box.

See Also HelpCommand property, HelpContextId property, HelpFile property, ShowFileDialog method.

Image Property

Description Returns or sets the file name of the current image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
√ Imaging for Windows 95 and 98
√ Imaging for Windows NT 4.0

Usage *object*.**Image**[=*filename*]

Data Type String.

Remarks When a new image is specified, the PageNumber property is set to 1.

This property is used by the following Image Admin control methods:

- Append
- DeletePages
- Insert
- Replace
- SetFileProperties
- ShowFileDialog
- ShowFileProperties
- ShowPrintDialog
- VerifyImage

When the Image property changes, the following Admin control properties are updated:

- CompressionInfo
- CompressionType
- FileType
- Page Count
- ImageResolutionX
- ImageResolutionY
- ImageWidth
- PageNumber
- PageType
- ImageHeight

Also, the Admin control **ShowFileProperty** method is updated.

See Also Append method, CompressionInfo property, CompressionType property, DeletePages method, FileType property, ImageHeight property, ImageResolutionY property, ImageResolutionX property, ImageWidth property, Insert method, PageCount property, PageNumber property, PageType property, Replace method, SetFileProperties method, ShowFileDialog method, ShowFileProperties method, ShowPrintDialog method, VerifyImage method.

Image Example – VB

This example shows syntax for the Image property. Single slashes can be forward or back.

```
Private Sub cmdSetImage_Click()
    'Showing syntax for 1.x Server document.
    'Note: prefix Image:// denotes service name for 1.x server.
    ImgAdmin1.Image = "Image://srvrname\volname:\CAB\DRAWER\FOLDER\YOUR DOC"
    'Showing syntax for 1.x Server filename with a volume name mapped.
    ImgAdmin1.Image = "Image://srvrname/volname:/dirname/temp.tif"
    'Showing syntax for 3.x Server document.
    'Note: prefix Image:// denotes service name for 3.x server.
    ImgAdmin1.Image = "Imagex://Docid1234
    'URL syntax
    ImgAdmin1.Image = "http://www.eastmansoftware.com/sbu/sampimg.tif"
    'UNC filename
    ImgAdmin1.Image = "\\srvrname\shared3\images\bw\thisisa.tif"
End Sub
```

Image Example – VC++

The following examples show syntax for the Image property. Single slashes can be forward or back.

```
void CAdmin1Dlg::OnClickImgedit1()
{
    // Examples of syntax for the Image property. Single slashes can be
    // forward or back. Showing syntax for 1.x Server document.
    // Note: prefix Image:// denotes service name for 1.x server.
    ImgAdmin1.SetImage  ("Image://srvrname\\volname:\\CAB\\DRAWER\\FOLDER
    ➥   \\YOUR DOC");
    // Showing syntax for 1.x Server filename with a volume name mapped.
    ImgAdmin1.SetImage ("Image://srvrname/volname:/dirname/temp.tif");
```

```
    // Showing syntax for 3.x Server document.
    // Note: prefix Image:// denotes service name for 3.x server.
    ImgAdmin1.SetImage ("Imagex://srvrname\\docid1234;)
    // URL syntax
    ImgAdmin1.SetImage ("http://www.eastmansoftware.com/sbu/sampimg.tif");
    // UNC filename
    ImgAdmin1.SetImage ("\\\\srvrname\\shared3\\images\\bw\\thisisa.tif");
}
```

ImageHeight Property

Description Returns the height, in pixels, of the current page.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

√ Imaging for Windows 98

Imaging for Windows 95

Imaging for Windows NT 4.0

Usage *object*.**ImageHeight**[=*value*]

Data Type Long.

Remarks The page is part of the current image file, as specified by the **Image** property.

This property is read-only.

See Also Image property, ImageWidth property, PageNumber property.

ImageHeight Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
```

```
    Case 1
        strImgFileType = "TIF"
    Case 2
        strImgFileType = "AWD"
    Case 3
        strImgFileType = "BMP"
    Case 4
        strImgFileType = "PCX"
    Case 5
        strImgFileType = "DCX"
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
```

```
        Case 2
            strImgPageType = "4 bit grayscale"
        Case 3
            strImgPageType = "8 bit grayscale"
        Case 4
            strImgPageType = "4 bit palettized"
        Case 5
            strImgPageType = "8 bit palettized"
        Case 6
            strImgPageType = "24 bit RGB"
        Case 7
            strImgPageType = "24 bit BGR"
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

ImageHeight Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
```

```
case 3:
     strImgFileType = "BMP";
          break;
case 4:
     strImgFileType = "PCX";
          break;
case 5:
     strImgFileType = "DCX";
          break;
case 6:
     strImgFileType = "JPG";
          break;
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
```

```
    case 8:
         strImgCompType = "Group3(2D)";
               break;
    case 9:
         strImgCompType = "LZW";
               break;
    }
    frmInfo.m_CompType = strImgCompType;
    // Read the PageType property and translate the value to a
    // corresponding string.
    switch (ImgAdmin1.GetPageType())
    {
    case 0:
         strImgPageType = "Unknown";
               break;
    case 1:
         strImgPageType = "Black and White";
               break;
    case 2:
         strImgPageType = "4 bit grayscale";
               break;
    case 3:
         strImgPageType = "8 bit grayscale";
               break;
    case 4:
         strImgPageType = "4 bit palettized";
               break;
    case 5:
         strImgPageType = "8 bit palettized";
               break;
    case 6:
         strImgPageType = "24 bit RGB";
               break;
    case 7:
         strImgPageType = "24 bit BGR";
               break;
    }
    frmInfo.m_PageType = strImgPageType;
    // Determine the dimensions of the image page.
    long lInfo = ImgAdmin1.GetImageHeight();
    frmInfo.m_Height.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageWidth();
    frmInfo.m_Width.Format("%i",lInfo);
    // Determine the X and Y resolution of the image page.
    lInfo = ImgAdmin1.GetImageResolutionX();
    frmInfo.m_XRes.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageResolutionY();
    frmInfo.m_YRes.Format("%i",lInfo);
    // Determine the number of pages in the file.
    lInfo = ImgAdmin1.GetPageCount();
    frmInfo.m_PageCount.Format("%i",lInfo);
    // Show the form with the image attributes.
    frmInfo.DoModal();
}
```

ImageResolutionX Property

Description Returns the horizontal resolution, in dots-per-inch (dpi), of the current page.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

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

Data Type Long.

Remarks The page is part of the current image file, as specified by the **Image** property.

This property is read-only.

See Also Image property, ImageResolutionY property, PageNumber property.

ImageResolutionX Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
        Case 5
            strImgFileType = "DCX"
```

```
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
    Case 2
        strImgPageType = "4 bit grayscale"
    Case 3
        strImgPageType = "8 bit grayscale"
    Case 4
        strImgPageType = "4 bit palettized"
    Case 5
        strImgPageType = "8 bit palettized"
    Case 6
        strImgPageType = "24 bit RGB"
    Case 7
        strImgPageType = "24 bit BGR"
```

```
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

ImageResolutionX Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
    case 6:
         strImgFileType = "JPG";
               break;
```

```
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
               break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
               break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
```

```
        case 0:
             strImgPageType = "Unknown";
                  break;
        case 1:
             strImgPageType = "Black and White";
                  break;
        case 2:
             strImgPageType = "4 bit grayscale";
                  break;
        case 3:
             strImgPageType = "8 bit grayscale";
                  break;
        case 4:
             strImgPageType = "4 bit palettized";
                  break;
        case 5:
             strImgPageType = "8 bit palettized";
                  break;
        case 6:
             strImgPageType = "24 bit RGB";
                  break;
        case 7:
             strImgPageType = "24 bit BGR";
                  break;
        }
        frmInfo.m_PageType = strImgPageType;
        // Determine the dimensions of the image page.
        long lInfo = ImgAdmin1.GetImageHeight();
        frmInfo.m_Height.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageWidth();
        frmInfo.m_Width.Format("%i",lInfo);
        // Determine the X and Y resolution of the image page.
        lInfo = ImgAdmin1.GetImageResolutionX();
        frmInfo.m_XRes.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageResolutionY();
        frmInfo.m_YRes.Format("%i",lInfo);
        // Determine the number of pages in the file.
        lInfo = ImgAdmin1.GetPageCount();
        frmInfo.m_PageCount.Format("%i",lInfo);
        // Show the form with the image attributes.
        frmInfo.DoModal();
}
```

ImageResolutionY Property

Description Returns the vertical resolution, in dots-per-inch (dpi), of the current page.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

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

Data Type Long.

Remarks The page is part of the current image file, as specified by the **Image** property.

This property is read-only.

See Also Image property, ImageResolutionX property, PageNumber property.

ImageResolutionY Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
        Case 5
            strImgFileType = "DCX"
```

```
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
    Case 2
        strImgPageType = "4 bit grayscale"
    Case 3
        strImgPageType = "8 bit grayscale"
    Case 4
        strImgPageType = "4 bit palettized"
    Case 5
        strImgPageType = "8 bit palettized"
    Case 6
        strImgPageType = "24 bit RGB"
    Case 7
        strImgPageType = "24 bit BGR"
```

```
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

ImageResolutionY Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
    case 6:
         strImgFileType = "JPG";
               break;
```

```
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
```

```
    case 0:
         strImgPageType = "Unknown";
               break;
    case 1:
         strImgPageType = "Black and White";
               break;
    case 2:
         strImgPageType = "4 bit grayscale";
               break;
    case 3:
         strImgPageType = "8 bit grayscale";
               break;
    case 4:
         strImgPageType = "4 bit palettized";
               break;
    case 5:
         strImgPageType = "8 bit palettized";
               break;
    case 6:
         strImgPageType = "24 bit RGB";
               break;
    case 7:
         strImgPageType = "24 bit BGR";
               break;
    }
    frmInfo.m_PageType = strImgPageType;
    // Determine the dimensions of the image page.
    long lInfo = ImgAdmin1.GetImageHeight();
    frmInfo.m_Height.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageWidth();
    frmInfo.m_Width.Format("%i",lInfo);
    // Determine the X and Y resolution of the image page.
    lInfo = ImgAdmin1.GetImageResolutionX();
    frmInfo.m_XRes.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageResolutionY();
    frmInfo.m_YRes.Format("%i",lInfo);
    // Determine the number of pages in the file.
    lInfo = ImgAdmin1.GetPageCount();
    frmInfo.m_PageCount.Format("%i",lInfo);
    // Show the form with the image attributes.
    frmInfo.DoModal();
}
```

ImageWidth Property

Description Returns the width, in pixels, of the current page.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**ImageWidth**[=*value*]

Data Type Long.

Remarks The page is part of the current image file, as specified by the **Image** property.

This property is read-only.

See Also Image property, ImageHeight property, PageNumber property.

ImageWidth Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
        Case 5
            strImgFileType = "DCX"
```

```
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
    Case 2
        strImgPageType = "4 bit grayscale"
    Case 3
        strImgPageType = "8 bit grayscale"
    Case 4
        strImgPageType = "4 bit palettized"
    Case 5
        strImgPageType = "8 bit palettized"
    Case 6
        strImgPageType = "24 bit RGB"
```

```
        Case 7
            strImgPageType = "24 bit BGR"
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

ImageWidth Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
     // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
```

```
case 6:
     strImgFileType = "JPG";
          break;
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
```

```
        case 0:
             strImgPageType = "Unknown";
                  break;
        case 1:
             strImgPageType = "Black and White";
                  break;
        case 2:
             strImgPageType = "4 bit grayscale";
                  break;
        case 3:
             strImgPageType = "8 bit grayscale";
                  break;
        case 4:
             strImgPageType = "4 bit palettized";
                  break;
        case 5:
             strImgPageType = "8 bit palettized";
                  break;
        case 6:
             strImgPageType = "24 bit RGB";
                  break;
        case 7:
             strImgPageType = "24 bit BGR";
                  break;
        }
        frmInfo.m_PageType = strImgPageType;
        // Determine the dimensions of the image page.
        long lInfo = ImgAdmin1.GetImageHeight();
        frmInfo.m_Height.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageWidth();
        frmInfo.m_Width.Format("%i",lInfo);
        // Determine the X and Y resolution of the image page.
        lInfo = ImgAdmin1.GetImageResolutionX();
        frmInfo.m_XRes.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageResolutionY();
        frmInfo.m_YRes.Format("%i",lInfo);
        // Determine the number of pages in the file.
        lInfo = ImgAdmin1.GetPageCount();
        frmInfo.m_PageCount.Format("%i",lInfo);
        // Show the form with the image attributes.
        frmInfo.DoModal();
}
```

InitDir Property

Description Returns or sets the initial directory displayed in the dialog box created by the **ShowFileDialog** method. See page 670 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**InitDir**[=*directoryname*]

Data Type String.

Remarks If the specified directory is invalid, the current directory is displayed.

See Also ShowFileDialog method.

InitDir Example – VB

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
Private Sub cmdDisplay_Click()
    'Initialize the Image property to check for Cancel pressed.
    'When the dialog box is shown, check for a value in the Image property.
    ImgAdmin1.Image = ""
    'Specify dialog title and list filter rather than using defaults.
    ImgAdmin1.DialogTitle = "Select Image for Display"
    ImgAdmin1.Filter = "All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|TIFF Files
    ➥    (*.tif)|*.tif|All Files (*.*)|*.*|"
    'The default filter will be TIFF (fourth item).
    ImgAdmin1.FilterIndex = 4
    'Set the InitDir property to the preferred directory.
    ImgAdmin1.InitDir = "C:\images"
    ImgAdmin1.CancelError = False
    'Note: If you have Server Access software installed this will
    'automatically be enabled in the Open dialog box after the user
    'authenticates.
    ImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd
    'Determine if a file was selected or cancel was pressed
    If ImgAdmin1.Image = "" Then Exit Sub
    'Set the image properties in the Image Edit and Thumbnail
    'controls to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Display the image in the Image Edit and Thumbnail control.
    ImgEdit1.Display
End Sub
```

InitDir Example – VC++

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
void CAdmin1Dlg::OnFileopen()
{
    // Initialize Image property in order to check for Cancel pressed. After
     // the dialog box is shown, check for a value in the image property.
    ImgAdmin1.SetImage ("" );
    // Specify a dialog title and a list filter rather than using the
    // defaults.
    ImgAdmin1.SetDialogTitle ("Select Image for Display");
    ImgAdmin1.SetFilter ("All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files (*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|Tiff Files
    ➥    (*.tif)|*.tif|All Files (*.*)|*.*|");
    // Default filter will be TIFF (fourth item).
    ImgAdmin1.SetFilterIndex (4);
    // Set the InitDir property to the preferred directory.
    ImgAdmin1.SetInitDir ("C:\\images");
    ImgAdmin1.SetCancelError (FALSE);
    // Note: If you have Server Access software installed this will
    // automatically be enabled in the
    // Open dialog box after the user authenticates.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);       // OpenDlg
    // Determine if a file was selected or cancel was pressed.
    if (ImgAdmin1.GetImage() =="")
            return;
    // Set the image properties in the ImgEdit and Thumbnail control
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage (ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage (ImgAdmin1.GetImage());
    // Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display();
}
```

Init1xFindDir Property

Description Returns or sets the initial directory displayed in the dialog box created by the **ShowFindDialog** method. See page 679 for an example of this dialog box.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Init1xFindDir**[=*value*]

Data Type String.

Remarks If the specified directory is invalid, the first 1.x Document Manager volume found is displayed.

Keywords Property

Description Sets or returns the Keywords property of the current image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98 (read only)
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**Keywords**[=*value*]

Data Type String.

Remarks This property pertains to local/redirected TIFF files, and 1.x files/documents.

Setting this property does not write the change to disk. To do so, you must call the **SetFileProperties** method, or, choose OK in the dialog box created by the **ShowFileProperties** method. See page 673 for an example of this dialog box.

The current setting of this property can be displayed and changed from the Summary tab on the dialog box created by the **ShowFileProperties** method.

See Also Author property, Comments property, SetFileProperties method, ShowFileProperties method, Subject property, Title property, Summary Properties.

Keywords Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the ShowFileProperties method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image you are going to set properties for.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

Keywords Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image you are going to set properties for.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

NameServer Property

Description Returns or sets the address of the 1.x server being accessed.

Available With

√	Imaging for Windows Professional Edition V2.0
	Imaging for Windows Professional Edition V1.0 and V1.1
	Imaging for Windows 95 and 98
	Imaging for Windows NT 4.0

Usage *object*.**NameServer**[=*value*]

Data Type String.

Remarks Use the format “IP address/port number” (for example, 172.25.18.79/4010) to specify the address.

The default value is the name of the 1.x server last accessed. The first time this property is used, the name of the server provided during software installation is used.

The **LoginToServer** method authenticates the user to the server specified by this property.

See Also LoginToServer method.

PageCount Property

Description Returns the number of pages in the image file. For information on image pages, see page 680. This property is read-only.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**PageCount**[=*value*]

Data Type Long.

Remarks The image file is specified by the **Image** property.

The value is 0 (zero) if the image specified does not exist.

See Also Image property.

PageCount Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
```

```
    Case 5
        strImgFileType = "DCX"
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
    Case 2
        strImgPageType = "4 bit grayscale"
    Case 3
        strImgPageType = "8 bit grayscale"
    Case 4
        strImgPageType = "4 bit palettized"
    Case 5
        strImgPageType = "8 bit palettized"
```

```
        Case 6
            strImgPageType = "24 bit RGB"
        Case 7
            strImgPageType = "24 bit BGR"
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

PageCount Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
```

```
case 6:
     strImgFileType = "JPG";
          break;
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
```

```
    case 0:
         strImgPageType = "Unknown";
              break;
    case 1:
         strImgPageType = "Black and White";
              break;
    case 2:
         strImgPageType = "4 bit grayscale";
              break;
    case 3:
         strImgPageType = "8 bit grayscale";
              break;
    case 4:
         strImgPageType = "4 bit palettized";
              break;
    case 5:
         strImgPageType = "8 bit palettized";
              break;
    case 6:
         strImgPageType = "24 bit RGB";
              break;
    case 7:
         strImgPageType = "24 bit BGR";
              break;
    }
    frmInfo.m_PageType = strImgPageType;
    // Determine the dimensions of the image page.
    long lInfo = ImgAdmin1.GetImageHeight();
    frmInfo.m_Height.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageWidth();
    frmInfo.m_Width.Format("%i",lInfo);
    // Determine the X and Y resolution of the image page.
    lInfo = ImgAdmin1.GetImageResolutionX();
    frmInfo.m_XRes.Format("%i",lInfo);
    lInfo = ImgAdmin1.GetImageResolutionY();
    frmInfo.m_YRes.Format("%i",lInfo);
    // Determine the number of pages in the file.
    lInfo = ImgAdmin1.GetPageCount();
    frmInfo.m_PageCount.Format("%i",lInfo);
    // Show the form with the image attributes.
    frmInfo.DoModal();
}
```

PageNumber Property

Description Returns or sets a page number in the current image file.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**PageNumber**

Data Type Long.

Remarks The image file is specified by the **Image** property.

If the image file is not multi-page, this value must be 1.

This property is available only at run-time (read and write).

See Also Image property.

PageNumber Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
    'Load the form where we will display file attributes.
    Load frmInfo
    'Read the Filetype property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.FileType
        Case 0
            strImgFileType = "Unknown"
        Case 1
            strImgFileType = "TIF"
        Case 2
            strImgFileType = "AWD"
        Case 3
            strImgFileType = "BMP"
        Case 4
            strImgFileType = "PCX"
```

```
    Case 5
        strImgFileType = "DCX"
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
'Read the PageType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.PageType
    Case 0
        strImgPageType = "Unknown"
    Case 1
        strImgPageType = "Black and White"
    Case 2
        strImgPageType = "4 bit grayscale"
    Case 3
        strImgPageType = "8 bit grayscale"
    Case 4
        strImgPageType = "4 bit palettized"
    Case 5
        strImgPageType = "8 bit palettized"
```

```
        Case 6
            strImgPageType = "24 bit RGB"
        Case 7
            strImgPageType = "24 bit BGR"
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

PageNumber Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch  (ImgAdmin1.GetFileType())
    {
    case 0:
         strImgFileType = "Unknown";
               break;
    case 1:
         strImgFileType = "TIF";
               break;
    case 2:
         strImgFileType = "AWD";
               break;
    case 3:
         strImgFileType = "BMP";
               break;
    case 4:
         strImgFileType = "PCX";
               break;
    case 5:
         strImgFileType = "DCX";
               break;
```

```
case 6:
     strImgFileType = "JPG";
          break;
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
```

```
        case 0:
             strImgPageType = "Unknown";
                  break;
        case 1:
             strImgPageType = "Black and White";
                  break;
        case 2:
             strImgPageType = "4 bit grayscale";
                  break;
        case 3:
             strImgPageType = "8 bit grayscale";
                  break;
        case 4:
             strImgPageType = "4 bit palettized";
                  break;
        case 5:
             strImgPageType = "8 bit palettized";
                  break;
        case 6:
             strImgPageType = "24 bit RGB";
                  break;
        case 7:
             strImgPageType = "24 bit BGR";
                  break;
        }
        frmInfo.m_PageType = strImgPageType;
        // Determine the dimensions of the image page.
        long lInfo = ImgAdmin1.GetImageHeight();
        frmInfo.m_Height.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageWidth();
        frmInfo.m_Width.Format("%i",lInfo);
        // Determine the X and Y resolution of the image page.
        lInfo = ImgAdmin1.GetImageResolutionX();
        frmInfo.m_XRes.Format("%i",lInfo);
        lInfo = ImgAdmin1.GetImageResolutionY();
        frmInfo.m_YRes.Format("%i",lInfo);
        // Determine the number of pages in the file.
        lInfo = ImgAdmin1.GetPageCount();
        frmInfo.m_PageCount.Format("%i",lInfo);
        // Show the form with the image attributes.
        frmInfo.DoModal();
}
```

PageType Property

Description Returns the page type of the specified page.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**PageType**[=*value*]

Data Type PageTypeValue (Integer).

Constant	Setting	Description
PageTypeUnk	0	Unknown
PageTypeBW	1	Black and white
PageTypeGray4	2	4-bit gray scale
PageTypeGray8	3	8-bit gray scale
PageTypePal4	4	4-bit palettized
PageTypePal8	5	8-bit palettized
PageTypeRGB24	6	24-bit RGB
PageTypeBGR24	7	24-bit BGR

Remarks The Image file is specified by the **Image** property; the page is specified by the **PageNumber** property.

If the Image property is empty, the value is 0 (zero).

This property is read-only.

See Also Image property, PageNumber property, Display Types.

PageType Example – VB

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties. Attributes for specific page of a specific file will be displayed.

```
Private Sub cmdGetInfo_Click()
    Dim strImgFileType, strImgCompType, strImgPageType As String
    ImgAdmin1.Image = "D:\image2\4page.tif"
    'Must specify page number if other than page 1.
    ImgAdmin1.PageNumber = 2
```

```
'Load the form where we will display file attributes.
Load frmInfo
'Read the Filetype property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.FileType
    Case 0
        strImgFileType = "Unknown"
    Case 1
        strImgFileType = "TIF"
    Case 2
        strImgFileType = "AWD"
    Case 3
        strImgFileType = "BMP"
    Case 4
        strImgFileType = "PCX"
    Case 5
        strImgFileType = "DCX"
    Case 6
        strImgFileType = "JPG"
    Case 7
        strImgFileType = "XIF"
    Case 8
        strImgFileType = "GIF"
    Case 9
        strImgFileType = "WIF"
End Select
frmInfo.lblImgInfo(0).Caption = strImgFileType
'Read the CompressionType property and translate the value to a
'corresponding string.
Select Case ImgAdmin1.CompressionType
    Case 0
        strImgCompType = "Unknown"
    Case 1
        strImgCompType = "No Compression"
    Case 2
        strImgCompType = "Group3(1D)"
    Case 3
        strImgCompType = "Group3(Modified Huffman)"
    Case 4
        strImgCompType = "PackBits"
    Case 5
        strImgCompType = "Group4(2D)"
    Case 6
        strImgCompType = "JPEG"
    Case 7
        strImgCompType = "RBA"
    Case 8
        strImgCompType = "Group3(2D)"
    Case 9
        strImgCompType = "LZW"
End Select
frmInfo.lblImgInfo(1).Caption = strImgCompType
```

```
    'Read the PageType property and translate the value to a
    'corresponding string.
    Select Case ImgAdmin1.PageType
        Case 0
            strImgPageType = "Unknown"
        Case 1
            strImgPageType = "Black and White"
        Case 2
            strImgPageType = "4 bit grayscale"
        Case 3
            strImgPageType = "8 bit grayscale"
        Case 4
            strImgPageType = "4 bit palettized"
        Case 5
            strImgPageType = "8 bit palettized"
        Case 6
            strImgPageType = "24 bit RGB"
        Case 7
            strImgPageType = "24 bit BGR"
End Select
    frmInfo.lblImgInfo(6).Caption = strImgPageType
    'Determine the dimensions of the image page.
    frmInfo.lblImgInfo(2).Caption = ImgAdmin1.ImageHeight
    frmInfo.lblImgInfo(3).Caption = ImgAdmin1.ImageWidth
    'Determine the X and Y resolution of the image page.
    frmInfo.lblImgInfo(4).Caption = ImgAdmin1.ImageResolutionX
    frmInfo.lblImgInfo(5).Caption = ImgAdmin1.ImageResolutionY
    'Determine the number of pages in the file.
    frmInfo.lblImgInfo(7).Caption = ImgAdmin1.PageCount
    'Show the form with the image attributes.
    frmInfo.Show
End Sub
```

PageType Example – VC++

This example shows how you can use the CompressionType, FileType, ImageHeight, ImageResolutionX, ImageResolutionY, ImageWidth and PageCount properties to request information about a file. Attributes for specific page of a specific file will be displayed.

```
void CAdmin1Dlg::OnGetimageinfo()
{
    // Load the form where we will display file attributes.
    CFrmInfo frmInfo;
    CString strImgFileType, strImgCompType, strImgPageType;
    // Must specify page number if other than page 1.
    ImgAdmin1.SetPageNumber(2);
    // Read the Filetype property and translate the value to a
    // corresponding string.
    switch (ImgAdmin1.GetFileType())
    {
    case 0:
            strImgFileType = "Unknown";
                        break;
```

```
case 1:
     strImgFileType = "TIF";
          break;
case 2:
     strImgFileType = "AWD";
          break;
case 3:
     strImgFileType = "BMP";
          break;
case 4:
     strImgFileType = "PCX";
          break;
case 5:
     strImgFileType = "DCX";
          break;
case 6:
     strImgFileType = "JPG";
          break;
case 7:
     strImgFileType = "XIF";
          break;
case 8:
     strImgFileType = "GIF";
          break;
case 9:
     strImgFileType = "WIF";
          break;
}
frmInfo.m_FileType = strImgFileType;
// Read the CompressionType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetCompressionType())
{
case 0:
     strImgCompType = "Unknown";
          break;
case 1:
     strImgCompType = "No Compression";
          break;
case 2:
     strImgCompType = "Group3(1D)";
          break;
case 3:
     strImgCompType = "Group3(Modified Huffman)";
          break;
case 4:
     strImgCompType = "PackBits";
          break;
case 5:
     strImgCompType = "Group4(2D)";
          break;
```

```
case 6:
     strImgCompType = "JPEG";
          break;
case 7:
     strImgCompType = "RBA";
          break;
case 8:
     strImgCompType = "Group3(2D)";
          break;
case 9:
     strImgCompType = "LZW";
          break;
}
frmInfo.m_CompType = strImgCompType;
// Read the PageType property and translate the value to a
// corresponding string.
switch (ImgAdmin1.GetPageType())
{
case 0:
     strImgPageType = "Unknown";
          break;
case 1:
     strImgPageType = "Black and White";
          break;
case 2:
     strImgPageType = "4 bit grayscale";
          break;
case 3:
     strImgPageType = "8 bit grayscale";
          break;
case 4:
     strImgPageType = "4 bit palettized";
          break;
case 5:
     strImgPageType = "8 bit palettized";
          break;
case 6:
     strImgPageType = "24 bit RGB";
          break;
case 7:
     strImgPageType = "24 bit BGR";
          break;
}
frmInfo.m_PageType = strImgPageType;
// Determine the dimensions of the image page.
long lInfo = ImgAdmin1.GetImageHeight();
frmInfo.m_Height.Format("%i",lInfo);
lInfo = ImgAdmin1.GetImageWidth();
frmInfo.m_Width.Format("%i",lInfo);
// Determine the X and Y resolution of the image page.
lInfo = ImgAdmin1.GetImageResolutionX();
frmInfo.m_XRes.Format("%i",lInfo);
```

```
    lInfo = ImgAdmin1.GetImageResolutionY();
    frmInfo.m_YRes.Format("%i",lInfo);
    // Determine the number of pages in the file.
    lInfo = ImgAdmin1.GetPageCount();
    frmInfo.m_PageCount.Format("%i",lInfo);
    // Show the form with the image attributes.
    frmInfo.DoModal();
}
```

PrintAnnotations Property

Description Returns or sets the state of the Print Displayed Annotations check box in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PrintAnnotations**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	The check box is checked, and visible annotations are printed.
False	The check box is not checked and visible annotations are not printed.

Remarks This property sets the state of the check box that is displayed when the dialog box is first opened. The user can accept or change this setting.

The Print dialog box is created by the **ShowPrintDialog** method.

See Also ShowPrintDialog method.

PrintAnnotations Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
```

```
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print
    'dialog box (for example, start page, end page, and so on).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥     ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
    'User pressed the cancel button.
        Exit Sub
End Sub
```

PrintAnnotations Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥     evt, evt);
}
```

PrintCollate Property

Description Returns or sets the state of the Collate check box in the the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Usage *object*.**PrintCollate**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	The Collate check box is checked — multiple copies are printed and collated.
False (default)	The Collate check box is not checked — multiple copies are printed but not collated.

Remarks The setting of this property is indicated by the state of the Collate check box when the Print dialog box is first opened. The user can then accept or change the value.

The Print dialog box is created by the **ShowPrintDialog** method.

See Also ShowPrintDialog method.

PrintEndPage Property

Description Returns or sets the number of the last page to be printed, which is displayed in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**PrintEndPage**

Data Type Long.

Remarks The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

The default value is the value specified in the **PageCount** property.

The Print dialog box is created by the **ShowPrintDialog** method.

This property is available only at run-time (read and write).

See Also PageCount property, PrintStartPage property, ShowPrintDialog method.

PrintEndPage Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print
    'dialog box (for example, start page, end page, and so on).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
    'User pressed the cancel button.
        Exit Sub
End Sub
```

PrintEndPage Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
```

```
    // Set filename to be printed to the displayed file. If this property
    // is not set, the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥    evt, evt);
}
```

PrintNumCopies Property

Description Returns or sets the default number of copies to be printed, which is displayed in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PrintNumCopies**[=*value*]

Data Type Long.

Remarks The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

The Print dialog box is created by the **ShowPrintDialog** method.

The value entered by the user is returned by the **ShowPrintDialog** method.

The default is 1 copy.

See Also ShowPrintDialog method.

PrintNumCopies Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
```

```
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print dialog box
    '(e.g., start/end page).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
    'User pressed the cancel button.
        Exit Sub
End Sub
```

PrintNumCopies Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
     ➥    evt, evt);
}
```

PrintOrientation Property

Description Returns or sets the the orientation of the printed page(s), which is displayed in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Usage *object*.**PrintOrientation**[=*value*]

Data Type PrintOrientationValue (Integer).

Constant	Setting	Description
PrintPortrait	0	Portrait
PrintLandscape	1	Landscape
PrintAutomatic	2 (default)	Automatic — the best orientation for each page is determined by the **PrintImage** method (Image Edit control).

Remarks The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

The Print dialog box is created by the **ShowPrintDialog** method.

See Also ShowPrintDialog method.

PrintOutputFormat Property

Description Returns or sets the format of the printer output, which is displayed in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PrintOutputFormat**[=*value*]

Data Type PrintFormatValue (Integer).

Constant	Setting	Description
OutPixel	0	Pixel-to-pixel.
OutActualSize	1	Actual size.
OutFitPage	2	Fit to page.
OutBestFit	3 (default)	Prints actual size if the image fits on the paper; otherwise, prints Fit to page.

Remarks The Print dialog box is created by the **ShowPrintDialog** method.

The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

The OutBestFit value is only available in Imaging for Windows Professional Edition V2.0.

See Also ShowPrintDialog method.

PrintOutputFormat Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print
    'dialog box (for example, start page, end page, and so on).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥     ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
    'User pressed the cancel button.
        Exit Sub
End Sub
```

PrintOutputFormat Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥    evt, evt);
}
```

PrintRangeOption Property

Description Returns or sets the print range option in the Print dialog box. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PrintRangeOption**[=*value*]

Data Type RangeValue (Integer).

Constant	Setting	Description
PrintAll	0 (default)	All pages
PrintRange	1	Range of pages
PrintCurrent	2	Current page

Constant	Setting	Description
PrintSelection	3	Current selection

Remarks The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

If a value of 1 is set, the start and end pages are provided by the **PrintStartPage** and **PrintEndPage** properties.

The Print dialog box is created by the **ShowPrintDialog** method.

The PrintSelection value is only available in Imaging for Windows Professional Edition V2.0.

See Also PrintEndPage property, PrintStartPage property, ShowPrintDialog method.

PrintStartPage Property

Description Returns or sets, in the Print dialog box, the number of the first page to be printed. See page 675 for an example of this dialog box.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**PrintStartPage**

Data Type Long.

Remarks The value set by this property is displayed when the Print dialog box is first opened. The user can then accept or change that value.

The Print dialog box is created by the **ShowPrintDialog** method.

The default value is 1.

This property is available only at run-time (read and write).

See Also PrintEndPage property, ShowPrintDialog method.

PrintStartPage Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
```

```
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print dialog
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
PrintErr:
    'User pressed the cancel button.
        Exit Sub
End Sub
```

PrintStartPage Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print dialog
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;      // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥    evt, evt);
}
```

PrintToFile Property

Description Returns or sets the state of the Print to File check box in the Print dialog box. See page 675 for an example of this dialog box.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 95 and 98

√ Imaging for Windows NT 4.0

Usage *object*.**PrintToFile**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	The check box is checked, and data is to be printed to a file.
False (default)	The check box is not checked, and data is not to be sent to a file.

Remarks The Print dialog box is created by the **ShowPrintDialog** method.

This property sets the state of the check box that is displayed when the dialog box is first opened. The user can accept or change this setting.

The state of the check box is returned by the **ShowPrintDialog** method.

See Also ShowPrintDialog method.

SaveAsName Property

Description Sets the unqualified or qualified file namestring to be used in the Filename edit field of the SaveAs dialog box.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SaveAsName**[=*value*]

Data Type String.

Remarks The value set by this property is displayed when the SaveAs dialog (ShowFileDialog method) is first opened. The user can accept or change this value. The default value is the name of the image defined in the Image property. If none is defined, the filename field is left blank. This property's value is automatically cleared after the SaveAs dialog is completed.

See Also ShowPrintDialog method.

Subject Property

Description Returns or sets the Subject property of the current image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98 (read only)
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**Subject**[=*value*]

Data Type String.

Remarks This property pertains to local/redirected TIFF files, and 1.x files (not documents).

Setting this property does not write the change to disk. To do so, you must call the **SetFileProperties** method, or, choose OK in the dialog box created by the **ShowFileProperties** method. See page 673 for an example of this dialog box.

The current setting of this property can be displayed and changed from the Summary tab on the dialog box created by the **ShowFileProperties** method.

See Also Author property, Comments property, Keywords property, SetFileProperties method, ShowFileProperties method, Title property, Summary Properties.

Subject Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the ShowFileProperties method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image you are going to set properties for.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

Subject Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image you are going to set properties for.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

Title Property

Description Returns or sets the Title property of the current image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 98 (read only)

Imaging for Windows 95

Imaging for Windows NT 4.0

Usage *object*.**Title**[=*value*]

Data Type String.

Remarks Microsoft Internet Explorer 4.0 reserves for its own use a property named Title. Therefore, you must use the following syntax if you use the Title property of the Image Admin control on an HTML page in Internet Explorer 4.0.

ImgAdmin1.Object.Title = “My Title”

If you do not include the Object property node, the Image Admin control will not read or write the Title property from the TIFF image nor will an error be reported. Note, however, that this syntax will generate an error in non-HTML environments or with Internet Explorer 3.0.

This property pertains to local/redirected TIFF files, and 1.x files (not documents).

Setting this property does not write the change to disk. To do so, you must call the **SetFileProperties** method, or, choose OK in the dialog box created by the **ShowFileProperties** method. See page 673 for an example of this dialog box.

The current setting of this property can be displayed and changed from the Summary tab on the dialog box created by the **ShowFileProperties** method.

See Also Author property, Comments property, Keywords property, SetFileProperties method, ShowFileProperties method, Subject property, Summary Properties.

Title Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the ShowFileProperties method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image you are going to set properties for.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

Title Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image you are going to set properties for.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

Append Method

Description Appends one or more pages to the current image.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Append** *Source,SourcePage,NumPages* [,*CompressionType, CompressionInfo*]

Arguments The Append method has the following arguments:

Parameter	Data Type	Description
Source	String	Specifies the image file that contains the page(s) to be appended. The file can reside on a ▪ local or redirected drive (desktop). ▪ 1.x server. ▪ 3.x server.
SourcePage	Long	Specifies the first page in the source image file to be appended.
NumPages	Long	Specifies the number of pages from the source image file to append to the current image file.
CompressionType	Integer	(Optional) Specifies the compression type to apply to the appended pages. If a type is not specified, the compression type of the source page(s) is used. See CompressionType property for allowable values.
CompressionInfo	Long	(Optional) Specifies which compression options to apply to the appended pages. If options are not specified, the compression options of the source page(s) are used. See CompressionInfo property for allowable values.

Remarks The current image file is specified by the **Image** property. If an image does not exist, one will be created.

The current image, if one exists, must be in TIFF or AWD format, as they support multiple pages. If the image pages added are of another type, they will be converted to the file type format used by the current image.

You can append a single BMP file to a new file. If you do, the only compression type allowed is None, with no options.

If the CompressionType and CompressionInfo values specified are not compatible with a source page's page type (color), the source file's compression type and compression information are used in place of those arguments.

If JPEG compression type is specified, and the CompressionInfo argument is 0 (zero), a default JPEG value is used. The default value is 4096 (high resolution, high quality).

If the destination is a 1.x document, and the source page does not reside in the 1.x repository, the **FileStgLoc1x** property must be set.

For other operations with 1.x documents, the **ForceFileLinking1x** property can be used.

See Also CompressionInfo property, CompressionType property, FileStgLoc1x property, ForceFileLinking1x property, Insert method, Replace method.

Append Example – VB

This example demonstrates how to append 3 image pages to a file.

```
Private Sub cmdAppend_Click()
    Dim strOrigFile As String, strAppendFile As String
    Dim lngPgCount As Long
    'This is the displayed image to which we are appending.
    ImgAdmin1.Image = "D:\image2\original.tif"
    'Because we are using the dialog box to pick the file for append, we
    'need to save the original file.
    strOrigFile = ImgAdmin1.Image
    ImgAdmin1.DialogTitle = "Select the file to be appended to the
    ➥    displayed image"
    ImgAdmin1.ShowFileDialog OpenDlg
    strAppendFile = ImgAdmin1.Image
    'Restore the original filename to the Image property because
    'the property must contain the destination file name.
    ImgAdmin1.Image = strOrigFile
    'Append 3 pages starting with page 1.
    ImgAdmin1.Append strAppendFile, 1, 3
End Sub
```

Append Example – VC++

This example demonstrates how to append 3 image pages to a file.

```
void CAdmin1Dlg::OnAppend()
{
    CString strOrigFile, strAppendFile;
    // This is the displayed image to which we are going to append.
    ImgAdmin1.SetImage ("D:\\image2\\original.tif");
    // We will use the dialog box to pick the file for append so we need to
     // save the original file.
    strOrigFile = ImgAdmin1.GetImage();
    ImgAdmin1.SetDialogTitle ("Select the file to be appended to the
    ➥    displayed image");
```

```
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);   // OpenDlg
    strAppendFile = ImgAdmin1.GetImage();
    // Restore image property to the original filename because this property
    // must contain the destination file.
    ImgAdmin1.SetImage (strOrigFile);
    // User wants to append 3 pages starting with page 1.
    VARIANT evt;
    evt.vt = VT_ERROR;   // Set to error for optional parameter
    ImgAdmin1.Append (strAppendFile, 1, 3, evt, evt);
}
```

Browse1x Method

Description Displays a dialog box (refer to Chapter 5) that lets the end user select the desired Imaging 1.x path.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Browse1x** *Browse1xScope,Title,Caption,hParentWnd*

Arguments The Browse1x method has the following arguments:

Parameter	Data Type	Description
Browse1xScope	Integer	Indicates the hierarchy displayed in the Browse dialog box. BrowseFiles (Literal 0) — Browse files (IFS - imaging file system) by displaying directories and subdirectories. BrowseDocuments (Literal 1) — Browse documents (DM - document management system) by displaying database, cabinet, drawer, or folder. BrowseBoth (Literal 2) — Browse both files and documents.
Title	String	Specifes the name of the string that appears in the title bar of the Browse dialog box.
Caption	String	Specifies a string that is used as the prompt that appears above the tree view in the Browse dialog box.
hParentWnd	hWnd	(Optional) Assigns a parent window handle to the displayed dialog box.

Remarks The path selected by the user is set in the **Browse1xReturnedPath** property when this method returns. The path type selected by the user is set in the **Browse1xReturnedType** property when this method returns.

See Also Browse1xReturnedPath property, Browse1xReturnedType property.

ConvertDate Method

Description Converts conventional (Gregorian) dates to Julian dates to be used when you call the **ImgQuery** method.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Usage *object*.**ConvertDate**(*Date* [*, nOption*])

Arguments The ConvertDate method has the following arguments:

Parameter	Data Type	Description
Date	String	Specifies the date to be converted. Acceptable formats include: "1/21/98", "January 21, 1998", or "1/21/1998".
nOption	Long	Specifies the type of conversion to perform: GregorianToJulian (Literal 1) — Gregorian to Julian (default).

See Also ImgQuery method.

CreateDirectory Method

Description Creates a user-defined directory.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**CreateDirectory** *Path*

Arguments The CreateDirectory method has the following argument:

Parameter	Data Type	Description
Path	String	Specifies the path to be created.

Remarks If the Path contains more than one subdirectory that does not exist, an error is returned.

Delete Method

Description Deletes a user-specified object.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Delete** *Object*

Arguments The Delete method has the following argument:

Parameter	Data Type	Description
Object	String	Specifies the object to be deleted.

Remarks The Object specified can be a file name or a path.

A file can't be deleted if it is in use by any Eastman Software/Wang Imaging application or Imaging ActiveX control. Also, a directory or folder cannot be deleted unless it is empty.

See Also ForceFileDeletion1x property.

DeletePages Method

Description Deletes a user-specified range of pages from the image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DeletePages** *StartPage,NumPages*

Arguments The DeletePages method has the following arguments:

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

Remarks The image file is specified by the **Image** property.

Pages cannot be deleted if they are being used by any Eastman Software/Wang Imaging application or Imaging ActiveX control.

If StartPage is within the image, and NumPages is greater than the number of pages in the image, all the pages after StartPage are deleted.

See Also ForceFileDeletion1x property, Image property.

DeletePages Example – VB

This example demonstrates how to delete a single page in a file.

```
Private Sub cmdDeletePage_Click()
    'Delete 1 page, starting with page 4.
    ImgAdmin1.DeletePages 4, 1
End Sub
```

DeletePages Example – VC++

This example demonstrates how to delete a single page in a file.

```
void CAdmin1Dlg::OnDeletepages()
{
    // Delete 1 page, starting with page 4.
    ImgAdmin1.DeletePages(4, 1 );
}
```

GetSysCompressionInfo Method

Description Retrieves the system default compression information for a given page type.

> **Note:** This method is obsolete, and should no longer be used. Use the CompressionInfo property instead. The following information is provided for older applications that may have used it.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetSysCompressionInfo**(*PageType*)

Arguments The GetSysCompressionInfo method has the following argument:

Parameter	Data Type	Description
PageType	Integer	Specifies a page type: PageTypeUnk (Literal 0) — Unknown PageTypeBW (Literal 1) — Black and white PageTypeGray4 (Literal 2) — 4-bit gray scale PageTypeGray8 (Literal 3) — 8-bit gray scale PageTypePal4 (Literal 4) — 4-bit palettized PageTypePal8 (Literal 5) — 8-bit palettized PageTypeRGB24Unk (Literal 6) — 24-bit RGB PageTypeBGR24 (Literal 7) — 24-bit BGR

Data Type Long.

Returns Returns the following bitwise compression information:

Setting	Description
1	EOL — include or expect standard end of line bit sequences
2	Packed lines—data is not byte aligned
4	Prefixed EOLs — include or expect prefixed end of line bit sequences
8	Compressed bit order, left to right
16	Expanded bit order, left to right
32	Negate — reverses black and white on expansion
64	Low resolution, high quality
128	Low resolution, medium quality
256	Low resolution, low quality
512	Medium resolution, high quality
1024	Medium resolution, medium quality
2048	Medium resolution, low quality
4096	High resolution, high quality

Setting	Description
8192	High resolution, medium quality
16384	High resolution, low quality

GetSysCompressionType Method

Description Retrieves the system default compression type for a given page type.

> **Note:** This method is obsolete, and should no longer be used. Use the CompressionType property instead. The following information is provided for older applications that may have used it.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetSysCompressionType**(*PageType*)

Arguments The GetSysCompressionType method has the following argument:

Parameter	Data Type	Description
PageType	Integer	Specifies a page type: PageTypeUnk (Literal 0) — Unknown PageTypeBW (Literal 1) — Black and white PageTypeGray4 (Literal 2) — 4-bit gray scale PageTypeGray8 (Literal 3) — 8-bit gray scale PageTypePal4 (Literal 4) — 4-bit palettized PageTypePal8 (Literal 5) — 8-bit palettized PageTypeRGB24Unk (Literal 6) — 24-bit RGB PageTypeBGR24 (Literal 7) — 24-bit BGR

Data Type Integer (enumerated).

Returns Returns one of the following compression types:

Constant	Value	Description
CompTypeUnk	0	Unknown
CompTypeNone	1	No compression
CompTypeGroup3	2	Group 3 1D FAX

Constant	Value	Description
CompTypeGroup3Huff	3	Group 3 Modified Huffman
CompTypePacked	4	PackBits
CompTypeGroup4	5	Group 4 2D FAX
CompTypeJPEG	6	JPEG
Reserved	7	Reserved
CompTypeGroup32DFax	8	Group 3 2D FAX
CompTypeLZW	9	LZW

GetSysFileType Method

Description Retrieves the system default file type for a given page type.

> **Note:** This method is obsolete, and should no longer be used. The following information is provided for older applications that may have used it.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetSysFileType**(*PageType*)

Arguments The GetSysFileType method has the following argument:

Parameter	Data Type	Description
PageType	Integer	Specifies a page type: PageTypeUnk (Literal 0) — Unknown PageTypeBW (Literal 1) — Black and white PageTypeGray4 (Literal 2) — 4-bit gray scale PageTypeGray8 (Literal 3) — 8-bit gray scale PageTypePal4 (Literal 4) — 4-bit palettized PageTypePal8 (Literal 5) — 8-bit palettized PageTypeRGB24Unk (Literal 6) — 24-bit RGB PageTypeBGR24 (Literal 7) — 24-bit BGR

Data Type Integer (enumerated).

Returns One of the following file types is returned:

Constant	Setting	Description
FileTypeUnk	0	Unknown
FileTypeTIFF	1	TIFF
FileTypeAWD	2	AWD (Windows 95 and 98 only)
FileTypeBMP	3	Bitmap (BMP)
FileTypePCX	4	PCX
FileTypeDCX	5	DCX
FileTypeJPEG	6	JPEG
FileTypeXIF	7	XIF
FileTypeGIF	8	GIF
FileTypeWIFF	9	WIFF

GetUniqueName Method

Description Generates a unique file name.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetUniqueName**(*Path,Template,Extension*)

Arguments The GetUniqueName method has the following arguments:

Parameter	Data Type	Description
Path	String	The name of the existing path to a local or redirected drive, or to a 1.x repository where a unique file name is generated.
Template	String	The initial characters of the generated file name. Up to 4 characters can be used.
Extension	String	The extension added to the generated file name. Up to 3 characters can be used.

Returns A string that contains the file name.

Remarks The file name is based on user-supplied path, template, and file extension information.

GetUniqueName Example – VB

This example generates a unique filename by using the Admin control for an unattended "Save As" operation in the Image Edit control.

```
Private Sub cmdUniqueFile_Click()
    Dim strNewfile As String
    Dim strPath as As String
    'Pass the directory, template, and file extension for the new file.
    strPath = "D:\image2"
    strNewfile = ImgAdmin1.GetUniqueName(strPath, "test", "tif")
    ImgEdit1.SaveAs strPath & "\" & strNewfile
End Sub
```

GetUniqueName Example – VC++

This example generates a unique filename by using the Admin control for an unattended "Save As" operation in the Image Edit control.

```
void CAdmin1Dlg::OnUniquefilename()
{
    CString strNewfile;
    Cstring strPath;
    // Pass in the directory, template, and file extension for the new file.
    strPath = "D:\\image2"
    strNewfile = ImgAdmin1.GetUniqueName(strPath, "test", "tif");
    VARIANT evt;
    evt.vt = VT_ERROR;   // set to error for optional parameter
    StrPath + = "\\";
    StrPath + = StrNewFile;
    ImgEdit1.SaveAs (strPath, evt, evt, evt, evt, evt);
}
```

GetVolumeType Method

Description Enables you to determine the type of a 1.x server volume.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**GetVolumeType**(*Server,Volume*)

Arguments The GetVolumeType method has the following arguments:

Parameter	Data Type	Description
Server	String	The name of the 1.x server that contains the volume.
Volume	String	The name of the volume. Volume names are always terminated with a colon (**:**).

Returns VOLUMETYPE

Constant	Value	Description
typeFILE	0	Volume is a 1.x IFS (imaging file system) volume.
typeDOCUMENT	1	Volume is a 1.x DM (document management) volume or database that can contain cabinets, drawers, and folders.
typeBADSPEC	2	Reserved
typeUNKNOWN	3	The volume type cannot be determined.
typeCANTINT	4	The function cannot initialize due to low memory or because it cannot access the 1.x server.

ImgQuery Method

Description Performs a query on a 1.x or 3.x document repository.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ImgQuery**(*vScope,szQueryTerms,iDispatch*)

Arguments The ImgQuery method has the following arguments:

Parameter	Data Type	Description
vScope	Variant	For a 1.x server, vScope is: ▪ a string that identifies a 1.x server and virtual volume. ▪ "DMVOLUMES", if all Document Manager (DM) volume names in the 1.x domain are to be returned. ▪ "IFSVOLUMES", if all Imaging File System (IFS) volume names in the 1.x domain are to be returned. For a 3.x server, vScope is a Boolean value that indicates the scope of the query. **False** = only the Domain will be queried. **True** = both the Domain and the Archive will be queried.
szQueryTerms	String	A pointer to a string containing the query terms, expressed in syntax appropriate to the server being queried.
iDispatch	Object	The address of an iDispatch pointer that points to a Collection object which will contain the query results.

Remarks The method **ImgQueryEnd** must be called after **ImgQuery** to free up associated resources.

See Also ConvertDate method, ImgQueryEnd method.

1.x Query Forms

Valid 1.x server query term string forms are:

null or null string

"findIFS *path*"

where *path* is a fully qualified path name from the virtual volume to the container to be enumerated

"findcabinets"

"finddrawers cabinet=*cabinetname*"

where *cabinetname* is the name of a cabinet

"findfolders cabinet=*cabinet*"

where *cabinet* is the name of a cabinet

"findfolders cabinet=*cabinetname;*drawer=*drawername*"

lists folders that are resident in a drawer, which is contained in a cabinet. You can supply names for the variables *cabinetname* and *drawername.*

Querying 1.x documents

You can query 1.x documents by name, date modified, location, and/or keywords. The syntax for this type of query is:

```
"finddocs [NAME_TERM] [DATE_EXPR] [KEYWORD_TERM]"
```

The syntax used by each parameter is shown below. Reserved words appear in **bold**.

```
[NAME_TERM] = <name rel_op qstring | name like qstring |
➥     name between qstring and qstring>
[DATE_EXPR] = <date_name date_rel_op qdate | date_name between qdate
➥     and qdate>
[KEYWORD_TERM] = <keyword_expr bool_op keyword_expr>
```

Variable definitions for these parameters are:

```
rel_op:  = | eq | < | lt | | gt | <= | le | >= | ge
date_time:  created | modified
date_rel_op:  before | after | on | rel_op
bool_op:  and | or
qdate:  Julian date in the format AYYYDDD
qstring: ASCII string
keyword_expr: keyword rel_op qstring | keyword like qstring |
    keyword between qstring and qstring
```

The 1.x reserved words — **cabinet**, **drawer**, **folder**, **document**, and **modified** — can be used only once in a querystring. The reserved word **keyword** can be used up to two times.

Sample 1.x DM queries

To find all documents with the keyword “insurance”:

```
finddocs keyword=insurance
```

To find all documents in a folder named “ROOT DRAWER”:

```
finddocs drawer="ROOT DRAWER"
```

Note that ROOT DRAWER must be enclosed by quotation marks to preserve the space between the two words.

3.x Query Forms

Valid 3.x server query term string forms are:

null or null string

returns the names of all documents

NAME=*name*

where *name* is the name of the document to be found.

OBJECT_CLASS=*class*

where *class* is a valid class name.

FIELDS=fieldname:value,fieldname:value,…;

where *fieldname* is the name of a field defined in the class's form, and *value* is the value of the field in the documents to be found.

Query Results

Query results are BSTR variants that take different forms, depending on the type of query made. In the sample forms shown below, generic names (such as server, cabinet, drawer, and so on) are used as placeholders.

1.x DM document query

If a query is made to a folder contained by a drawer, the result will be in the form:

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

If a query is made to a folder contained by a cabinet, the result will be in the form:

Image://server\volume:\cabinet\folder\document

If a 1.x cabinet, drawer, or folder enumeration is requested, the result will be a fully qualified path describing the item. For example,

Image://server\volume:\cabinet

or

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

or

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

1.x Filesystem (IFS) query

A 1.x IFS query result is returned in the form:

[f]Image://server\volume:/directory/directory/.../file

or

[d]Image://server\volume:/directory/directory/.../directory

Where [f] means the item found was a file, and [d] means that it is a directory.

3.x document query

A document query to a 3.x server is returned in the form:

Imagex://document

ImgQuery Example – VB

The following example performs a query of documents located on a 1.x Server. The query is based upon keywords assigned to the document.

```
Private Sub cmdQuery_Click()
    Dim objResults As Object
    Dim varItem As Variant
    Dim intRet As Integer
```

```
    'Perform the query and put results in objResults.
    intRet = ImgAdmin1.ImgQuery("srvrname\test_db:", "finddocs
    ➥    keyword = insurance", objResults)
    'Write results of the query to a listbox.
    If intRet = 0 Then
        For Each varItem In objResults
            lb_results.AddItem varItem
        Next
        'Release memory and resources.
        Set objResults = Nothing
        ImgAdmin1.ImgQueryEnd
    End If
End Sub
```

ImgQuery Example – VC++

The following example performs a query of documents located on a 1.x Server. The query is based upon keywords assigned to the document.

```
void CAdmin1Dlg::OnImgquery()
{
    int iReturn;
    // Perform the query and put results in objResults.
        CString szDB = "test_db:";
        VARIANT vDB; vDB.vt = VT_BSTR; V_BSTR(&vDB) =  szDB.AllocSysString();
        lbResults.ResetContent();
        // Call ImgQuery(); get a CStringCollect's IDispatch*.
        LPDISPATCH idispStringCollect=NULL;
        iReturn = ImgAdmin1.ImgQuery(vDB, "finddocs keyword =
        ➥    insurance",&idispStringCollect);
        if (iReturn == 0)
        {
           IStringCollect* pIStringCollect;
           TRY
           {
               // Make new IStringCollect* and attach the Dispatch pointer.
               pIStringCollect = new IStringCollect;
               pIStringCollect->AttachDispatch(idispStringCollect, TRUE);
               // Call CStringCollect's Get_NewEnum();
               // get CEnumVARIANT's IUnknown*
              IUnknown* pEnumVarIUnknown = 0;
              pEnumVarIUnknown = pIStringCollect->Get_NewEnum();
              if (pEnumVarIUnknown)
              {
                  // Call CEnumVARIANT IUnknown's QueryInterface;
                  // get an IEnumVARIANT*
                  IEnumVARIANT* pIEnumVariant = 0;
                  pEnumVarIUnknown->QueryInterface(IID_IEnumVARIANT,
                  ➥    (void**)&pIEnumVariant);
                  if(pIEnumVariant)
                  {
```

```
                        // Call the IEnumVARIANT's Next() till all strings are
                        // retrieved.
                        VARIANT vName;
                        while (S_OK == pIEnumVariant->Next(1, &vName, NULL))
                        {
                               lbResults.AddString((CString)V_BSTR(&vName));
                        }
                        iReturn = 0;
                        pIEnumVariant->Release();
                      }
                      pEnumVarIUnknown->Release();
                   }
                   pIStringCollect->ReleaseDispatch();
                   delete pIStringCollect;
                   ImgAdmin1.ImgQueryEnd();
                }
                CATCH_ALL (e)
                {
                 CString msg;
                 pIStringCollect->ReleaseDispatch();
                 delete pIStringCollect;
                 ImgAdmin1.ImgQueryEnd();
                 msg.Format("Unexpected error 0x%lX", ((COleException*)e)->m_sc);
                 MessageBox(msg);
                }
                END_CATCH_ALL
             }
}
```

ImgQueryEnd Method

Description Completes a query and frees any associated resources.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ImgQueryEnd ()**

Arguments None.

Remarks Use this method with the **ImgQuery** method.

See Also ImgQuery method.

ImgQueryEnd Example – VB

The following example performs a query of documents located on a 1.x Server. The query is based upon keywords assigned to the document.

```
Private Sub cmdQuery_Click()
    Dim objResults As Object
    Dim varItem As Variant
    Dim intRet As Integer
    'Perform the query and put results in objResults.
    intRet = ImgAdmin1.ImgQuery("srvrname\test_db:", "finddocs
    ➥    keyword = insurance", objResults)
    'Write results of the query to a listbox.
    If intRet = 0 Then
        For Each varItem In objResults
            lb_results.AddItem varItem
        Next
        'Release memory and resources.
        Set objResults = Nothing
        ImgAdmin1.ImgQueryEnd
    End If
End Sub
```

ImgQueryEnd Example – VC++

The following example performs a query of documents located on a 1.x Server. The query is based upon keywords assigned to the document.

```
void CAdmin1Dlg::OnImgquery()
{
    int iReturn;
    // Perform the query and put results in objResults.
        CString szDB = "test_db:";
        VARIANT vDB; vDB.vt = VT_BSTR; V_BSTR(&vDB) = szDB.AllocSysString();
        lbResults.ResetContent();
        // Call ImgQuery(); get a CStringCollect's IDispatch*.
        LPDISPATCH idispStringCollect=NULL;
        iReturn = ImgAdmin1.ImgQuery(vDB, "finddocs keyword = insurance",
        ➥    & idispStringCollect);
        if (iReturn == 0)
        {
            IStringCollect* pIStringCollect;
            TRY
            {
               // Make new IStringCollect* and attach the Dispatch pointer.
               pIStringCollect = new IStringCollect;
               pIStringCollect->AttachDispatch(idispStringCollect, TRUE);
               // Call CStringCollect's Get_NewEnum(); get CEnumVARIANT's
               // IUnknown*
               IUnknown* pEnumVarIUnknown = 0;
               pEnumVarIUnknown = pIStringCollect->Get_NewEnum();
               if (pEnumVarIUnknown)
               {
```

```
                    // Call CEnumVARIANT IUnknown's QueryInterface; get an
                    // IEnumVARIANT*
                    IEnumVARIANT* pIEnumVariant = 0;
                    pEnumVarIUnknown->QueryInterface(IID_IEnumVARIANT,
                    ➥    (void**)&pIEnumVariant);
                    if(pIEnumVariant)
                    {
                    // Call the IEnumVARIANT's Next() till all x strings are
                    // retrieved.
                      VARIANT vName;
                      while (S_OK == pIEnumVariant->Next(1, &vName, NULL))
                        {
                             lbResults.AddString((CString)V_BSTR(&vName));
                         }
                         iReturn = 0;
                         pIEnumVariant->Release();
                    }
                    pEnumVarIUnknown->Release();
                }
                pIStringCollect->ReleaseDispatch();
                delete pIStringCollect;
                ImgAdmin1.ImgQueryEnd();
    }
    CATCH_ALL (e)
    {
            CString msg;
            pIStringCollect->ReleaseDispatch();
            delete pIStringCollect;
            ImgAdmin1.ImgQueryEnd();
            msg.Format("Unexpected error 0x%1X", ((COleException*)e)->m_sc);
            MessageBox(msg);
    }
    END_CATCH_ALL
  }
}
```

Insert Method

Description Inserts one or more user-specified pages into the current image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Insert** *Source,SourcePage,DestinationPage,NumPages*
[,*CompressionType*,*CompressionInfo*]

Arguments The Insert method has the following arguments:

Parameter	Data Type	Description
Source	String	Specifies the image file that contains the page(s) to be inserted. The file can reside on a ▪ local or redirected drive (desktop). ▪ 1.x server. ▪ 3.x server.
SourcePage	Long	Specifies which page in the source image file will be inserted.
DestinationPage	Long	Specifies the page in the current image file the selected pages will be inserted before.
NumPages	Long	Specifies the number of pages from the source image file to insert into the current image file.
CompressionType	Integer	(Optional) Specifies the compression type to apply to the inserted pages. If a type is not specified, the compression type of the source image is used. See the CompressionType property for allowable values.
CompressionInfo	Long	(Optional) Specifies which compression options to apply to the inserted pages. If options are not specified, the compression options of the source image are used. See the CompressionInfo property for allowable values.

Remarks The current image is specified by the **Image** property. If an image does not exist, one is created, and the destination page will be 1.

The current image must be a TIFF, AWD, or 1.x server document, because they support multiple pages. If the image pages added are of another type, they will be converted to the file type used by the current image.

You can insert a single BMP page into a new image. If you do, the only compression type allowed is None, with no options.

If the CompressionType and CompressionInfo values specified are not compatible with a source page's page (color) type, the source file's compression type and compression information is used in place of the arguments.

If JPEG compression is specified, and the CompressionInfo argument is 0 (zero), a default JPEG value is used. The default value is 4096 (high resolution, high quality).

If the destination is a 1.x document, and the source page does not reside in the 1.x repository, the **FileStgLoc1x** property must be set.

For other operations with 1.x documents, the **ForceFileLinking1x** property can be used.

See Also CompressionInfo property, CompressionType property, FileStgLoc1x property, ForceFileLinking1x property, Append method, Replace method.

Insert Example – VB

This example demonstrates how to insert pages into a file.

```
Private Sub cmdInsert_Click()
    Dim strOrigFile As String, strInsertFile As String
    Dim lngPgCount As Long
    'This is the displayed image into which we insert the page.
    ImgAdmin1.Image = "D:\image2\original.tif"
    'Because we will use the dialog box to pick the file to be
    'inserted, we need to save the original file.
    strOrigFile = ImgAdmin1.Image
    ImgAdmin1.DialogTitle = "Select the file you want to insert into the
    ➥    displayed image"
    ImgAdmin1.ShowFileDialog OpenDlg
    strInsertFile = ImgAdmin1.Image
    'Restore the original filename to the Image property because
    'the property must contain the destination file name.
    ImgAdmin1.Image = strOrigFile
    'Insert 2 pages starting with page 1 of the source file, which will be
    'placed before page 3 in the original file. The compression type of the
    'source file will be retained since a different compression type is not
    'specified.
    ImgAdmin1.Insert strInsertFile, 1, 3, 2
End Sub
```

Insert Example – VC++

This example demonstrates how to insert pages into a file.

```
void CAdmin1Dlg::OnInsert()
{
    CString strOrigFile, strInsertFile;
    // This is the displayed image into which we insert the page.
    ImgAdmin1.SetImage ("D:\\image2\\original.tif");
    // Because we will use the dialog box to pick the file to be
    // inserted, we need to save the original file.
    strOrigFile = ImgAdmin1.GetImage();
    ImgAdmin1.SetDialogTitle ("Select the file you want to insert into the
    ➥    displayed image");
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);   // OpenDlg
    strInsertFile = ImgAdmin1.GetImage();
    // Restore image property to the original filename because
    // this property must contain destination file.
    ImgAdmin1.SetImage(strOrigFile);
    // Insert 2 pages starting with page 1 of the source file, which will be
     // placed before page 3 in the original file. The compression type of
```

```
    // the source file will be retained since a different compression type
    // is not specified.
    VARIANT evt;
    evt.vt = VT_ERROR;  // set to error for optional parameter
    ImgAdmin1.Insert (strInsertFile, 1, 3, 2 , evt, evt);
}
```

LoginToServer Method

Description Provides the authentication interface to a 1.x or 3.x server, enabling a user to log on.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**LoginToServer** *nServerType*,*bEnableUI*,[*lpszUserId*],[*lpszPassWd*]

Arguments The LoginToServer method has the following arguments:

Parameter	Data Type	Description
nServerType	ServerType (Integer)	Indicates the type of server to be accessed.

Constant	Setting	Description
OneXUnix	1	1.x UNIX server
ThreeX	2	3.x server

Parameter	Data Type	Description
bEnableUI	AuthUI (Boolean)	Determines if the logon dialog box is displayed.

Constant	Setting	Description
DisableUI	False	Disables the dialog box.
EnableUI	True	Enables the dialog box.

lpszUserId	String	(Optional) User logon ID (up to 20 characters for a 1.xserver or 80 characters for 3.x server).
lpszPasswd	String	(Optional) User logon password (up to 36 characters for 1.x server or 80 characters for a 3.x server).

Remarks If nServerType is set to 2, the user is logged on to the 3.x server specified by the **Domain** property. If nServerType is set to 1, the user is logged on to the Imaging 1.x server specified by the NameServer property.

> **Note:** Multiple users cannot log on to a 3.x server using the same ID.

When a user is logged on to a 3.x server, authentication remains current until the application is closed. On a 1.x server, however, authentication remains valid after the application is closed.

To log a user off, use the **LogOffServer** method.

See Also Domain property, NameServer property, LogOffServer method.

LogOffServer Method

Description Logs a user off the server.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**LogOffServer** *nServerType*

Arguments The LogOffServer method has the following arguments:

Parameter	Data Type	Description
nServerType	ServerType (Integer)	Indicates the server type currently being accessed.

Constant	Setting	Description
OneXUnix	1	1.x server
ThreeX	2	3.x server

See Also LoginToServer method.

Rename Method

Description Renames a file, a document, or a directory in an Eastman Imaging 1.x Server file or document manager namespace.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.1
- Imaging for Windows Professional Edition V1.0
- Imaging for Windows 95 and 98
- Imaging for Windows NT 4.0

Usage *object*.**Rename** *Path,CurrentName,NewName*

Arguments The Rename method has the following arguments:

Parameter	Data Type	Description
Path	String	Specifies the path to the object to be renamed.
CurrentName	String	Specifies the current name of the object.
NewName	String	Specifies the new name of the object.

Replace Method

Description Replaces one or more user-specified pages in a file specified by the **Image** property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Replace** *Source,SourcePage,DestinationPage,NumPages*
[,*CompressionType,CompressionInfo*]

Arguments The Replace method has the following parameters:

Parameter	Data Type	Description
Source	String	Specifies the image file that contains the page(s) to be inserted. The file can reside on a ▪ local or redirected drive (desktop). ▪ 1.x server. ▪ 3.x server.

Parameter	Data Type	Description
SourcePage	Long	Specifies the first page in the source image file that will be inserted.
DestinationPage	Long	Specifies the first page in the current image file that will be replaced.
NumPages	Long	Specifies the number of pages in the current image file to replace with pages from the source image file.
CompressionType	Integer	(Optional) Specifies the compression type to apply to the replaced pages. If a type is not specified, the compression type of the source image is used. See the CompressionType property for allowable values.
CompressionInfo	Long	(Optional) Specifies which compression options to apply to the replaced pages. If options are not specified, the compression options of the source page are used. See the CompressionInfo property for allowable values.

Remarks The current image is specified by the **Image** property.

The replaced pages are converted to the file type of the current image (TIFF, AWD, or BMP).

If you are replacing the single page in a BMP file, the only compression type allowed is None, with no options.

If the CompressionType and CompressionInfo values specified are not compatible with a source page's page (color) type, the source file's compression type and compression information is used in place of the arguments.

If JPEG compression is specified, and the CompressionInfo argument is 0 (zero), a default JPEG value is used. The default value is 4096 (high resolution, high quality).

If the destination is a 1.x document, and the source page does not reside in the 1.x repository, the **FileStgLoc1x** property must be set.

For other operations with 1.x documents, the **ForceFileLinking1x** property can be used.

See Also CompressionInfo property, CompressionType property, FileStgLoc1x property, ForeFileLinking1x property, Append method, Insert method.

Replace Example – VB

This example demonstrates how to replace a single page in a file.

```
Private Sub cmdReplace_Click()
    Dim strOrigFile As String, strReplaceFile As String
    Dim lngPgCount As Long
    'Use the dialog box to pick the original file.
    ImgAdmin1.DialogTitle = "Select the original file"
```

```
    ImgAdmin1.ShowFileDialog OpenDlg
    'Because we are using the dialog box to pick the file to
    'be used for replace, we need to save the original file.
    strOrigFile = ImgAdmin1.Image
    ImgAdmin1.DialogTitle = "Select the file to be used for the replace"
    ImgAdmin1.ShowFileDialog OpenDlg
    strReplaceFile = ImgAdmin1.Image
    'Restore the original filename to the Image property because
    'the property must contain the destination file name.
    ImgAdmin1.Image = strOrigFile
    'You are going to replace 1 page – page 4 in original file will be
    'replaced by page 2 of the source file. The compression type of the
    'source file will be retained since a different compression type is
    'not specified.
    ImgAdmin1.Replace strReplaceFile, 2, 4, 1
End Sub
```

Replace Example – VC++

This example demonstrates how to replace a single page in a file.

```
void CAdmin1Dlg::OnReplace()
{
    CString strOrigFile, strReplaceFile;
    // Use the dialog box to pick the original file.
    ImgAdmin1.SetDialogTitle ("Select the original file");
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);   // OpenDlg
    // Because we are using the dialog box to pick the file to
    // be used for replace, we need to save the original file.
    strOrigFile = ImgAdmin1.GetImage();
    ImgAdmin1.SetDialogTitle ("Select the file to be used for the replace");
    ImgAdmin1.ShowFileDialog (0,vhWnd);   // OpenDlg
    strReplaceFile = ImgAdmin1.GetImage();
    // Restore the original filename to the Image property because
    // the property must contain the destination file name.
    ImgAdmin1.SetImage (strOrigFile);
    // You are going to replace 1 page – page 4 in original file will be
    // replaced by page 2 of the source file. The compression type of the
    // source file will be .0retained since a different compression type is
    // not specified.
    VARIANT evt;
    evt.vt = VT_ERROR; // set to error for optional parameter
    ImgAdmin1.Replace (strReplaceFile, 2, 4, 1, evt, evt);
}
```

SetFileProperties Method

Description Writes the Title, Subject, Author, Comments, and Keywords properties for the current image to disk.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetFileProperties**

Arguments None.

Remarks First, you assign the appropriate values to the Title, Subject, Author, Comments, and Keywords properties. Then you call the SetFileProperties method to write them to disk. If the current image is a 1.x document, only the **Keywords** property can be set.

> **Note:** Keywords assigned to 1.x documents using this method are automatically converted to uppercase characters.

The **Image** property determines which document is updated.

If the current image is a local or redirected TIFF file, or a 1.x file (not a document) the **Title**, **Subject**, **Author**, **Comments**, and **Keywords** properties can be set.

After setting file properties, you can use the Find button on the Open dialog box (see page 670) to search for local and redirected files, or the **ImgQuery** method to search for documents residing on a 1.x server.

Use the **ShowFileProperties** method to display the dialog box that shows the settings for these properties on the Summary tab. See page 673 for an example of this dialog box.

See Also Author property, Comments property, Image property, Keywords property, ShowFileProperties method, Subject property, Title property, ImgQuery method, ShowFileDialog method.

SetFileProperties Example – VB

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the ShowFileProperties method.

```
Private Sub cmdSetProperties_Click()
    'Specify the image for which you are going to set properties.
    ImgAdmin1.Image = "D:\image2\hopolicy.tif"
    ImgAdmin1.Title = "HomeOwners Policy"
    ImgAdmin1.Author = "John Q. Agent"
    'Add multiple keywords.
    ImgAdmin1.Keywords = "insurance important house"
```

```
    ImgAdmin1.Comments = "1997 Homeowners Policy"
    ImgAdmin1.Subject = "Insurance"
    'Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties
End Sub
```

SetFileProperties Example – VC++

This example shows how you can set several file properties. These properties can also be set by the user at runtime from the UI dialog box created by the **ShowFileProperties** method.

```
void CAdmin1Dlg::OnFileproperties()
{
    // Specify the image for which you are going to set properties.
    ImgAdmin1.SetImage ("D:\\image2\\hopolicy.tif");
    ImgAdmin1.SetTitle ("HomeOwners Policy");
    ImgAdmin1.SetAuthor ("John Q. Agent");
    // Adding multiple keywords.
    ImgAdmin1.SetKeywords ("insurance important house");
    ImgAdmin1.SetComments ("1997 Homeowners Policy");
    ImgAdmin1.SetSubject ("Insurance");
    // Call SetFileProperties to actually write the properties to the file.
    ImgAdmin1.SetFileProperties();
}
```

SetSystemFileAttributes Method

Description Allows the user to set how image files will be saved for a given type.

> **Note:** This method is obsolete, and should no longer be used. The following information is provided for older applications that may have used it.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**SetSystemFileAttributes** *PageType,FileType,CompressionType, CompressionInfo*

Arguments The SetSystemFileAttributes method has the following arguments:

Parameter	Data Type	Description
PageType	Integer	Specifies a page type: 0 — Unknown 1 — Black and white 2 — 4-bit gray scale 3 — 8-bit gray scale 4 — 4-bit palettized 5 — 8-bit palettized 6 — 24-bit RGB 7 — 24-bit BGR
FileType	Integer	Specifies a file type value: 0 — Unknown 1 — TIF 2 — AWD (Windows 95 and 98 only) 3 — Bitmap (BMP) 4 — PCX 5 — DCX 6 — JPEG 7 — XIF 8 — GIF
CompressionType	Integer	Specifies compression information: 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

Parameter	Data Type	Description
CompressionInfo	Long	Specifies compression information: 1 — EOL - inlcude or expect standard end of line bit sequences 2 — Packed lines - data is not byte aligned 4 — Prefixed EOLs - include or expect prefixed end of line bit sequences 8 — Compressed bit order, left to right 16 — Expanded bit order, left to right 32 — Negate - reverse black and white on expansion 64 — Low resolution, high quality 128 — Low resolution, medium quality 256 — Low resolution, low quality 512 — Medium resolution, high quality 1024 — Medium resolution, medium quality 2048 — Medium resolution, low quality 4096 — High resolution, high quality 8192 — High resolution, medium quality 16384 — High resolution, low quality

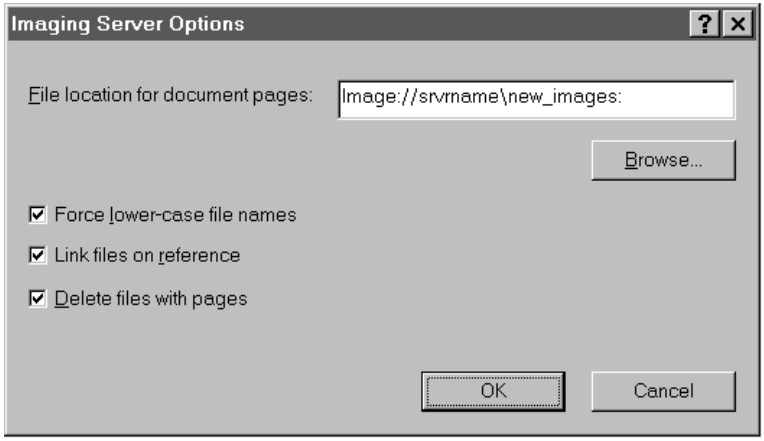
Remarks These settings will be used when scanning files with the Image Scan control.

Defaults are set for a classification of the page type into one of three categories:

- Black and White
- Gray scale
- Color

Show1xServerOptDlg Method

Description Displays the 1.x Server Options dialog box (shown here).



Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Show1xServerOptDlg** [*hParentWnd*]

Arguments The Show1xServerOptDlg method has the following argument:

Parameter	Data Type	Description
hParentWnd	hWnd	(Optional) Assigns a parent window handle to the displayed dialog box.

Remarks The following values are set by the specified properties. A user can accept or change these settings from the dialog box.

- The initial value for the "File location for document pages" edit box is set by the **FileStgLoc1x** property.
- The initial state of the checkbox "Force lower-case file names" is set by the **ForceLowerCase1x** property.
- The initial state of the checkbox "Link files on reference" is set by the **ForceFileLinking1x** property.

- The initial state of the checkbox “Delete files with pages” is set by the **ForceFileDeletion1x** property.

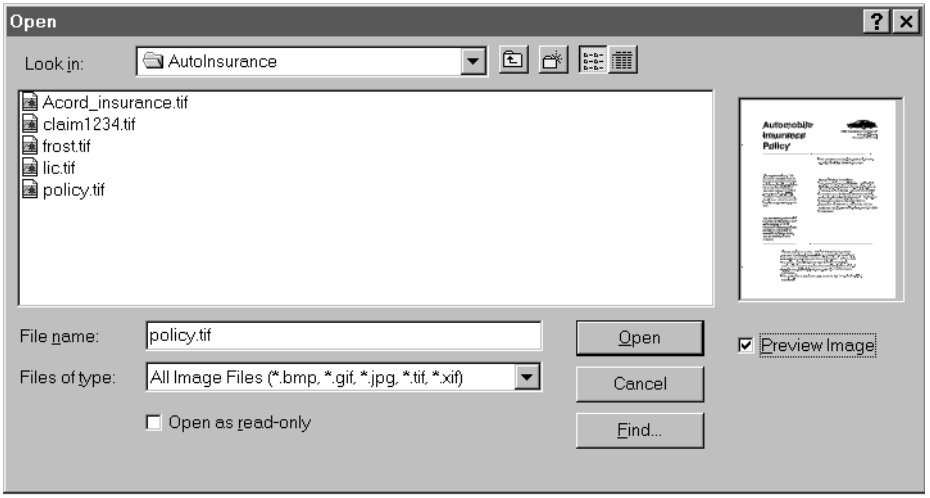
The Browse button displays a list of folders in the Browse dialog box. If a user is not already logged on to a 1.x server, he/she will be prompted to do so before the list is displayed.

See Also FileStgLoc1x property, ForceFileDeletion1x property, ForceFileLinking1x property, ForceLowerCase1x property.

ShowFileDialog Method

Description Enables a user to select a file from a local or redirected drive. Version 2.0 of Imaging for Windows Professional Edition supports 1.x server file selection, and queries a 3.x server document repository.

Depending on the setting of the DialogOption parameter, either an Open or a Save As dialog box is displayed. The Open dialog box is shown here.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowFileDialog** (*DialogOption*[,*hParentWnd*])

Arguments The ShowFileDialog method has the following arguments:

Parameter	Data Type	Description
DialogOption	Integer	Indicates what type of file dialog box will be displayed.
hParentWnd	hWnd	(Optional) Assigns a parent window handle to the displayed dialog box.

Constant	Setting	Description
OpenDlg	0	Open or Select
SaveDlg	1	Save As or Save Page

Remarks The file specification selected by the user from the dialog box is set in the **Image** property when the method returns the value.

The value set by the **SaveAsName** property is displayed when the SaveAs dialog is first opened. The user can accept or change this value. The default value is the name of the image defined in the Image property. If none is defined, the filename field is left blank. The SaveAsName property value is automatically cleared after the SaveAs dialog is completed.

If a help file is not specified in the **HelpFile** property, a Help button will not be shown. This is true even if the show help flag is set in the **Flags** property.

If a Parent window is not specified, the dialog box will behave as a modeless dialog box.

See Also CancelFrom property, Flags property, HelpFile property, Image property, SaveAsName property.

ShowFileDialog Example – VB

Example 1 — Open

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
Private Sub cmdDisplay_Click()
    'Initialize the Image property to check for Cancel pressed. When the
    'dialog box is shown, check for a value in the Image property.
    ImgAdmin1.Image = ""
    'Specify a dialog title and a list filter rather than using the defaults.
    ImgAdmin1.DialogTitle = "Select Image for Display"
    ImgAdmin1.Filter = "All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files(*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|TIFF Files
    ➥    (*.tif)|*.tif|All Files (*.*)|*.*|"
    'Default filter will be TIFF (fourth item).
    ImgAdmin1.FilterIndex = 4
    'Set the InitDir property to the preferred directory.
    ImgAdmin1.InitDir = "C:\images"
    ImgAdmin1.CancelError = False
    'Note: If you have Server Access software installed this will automatically
    'be enabled in the Open dialog box after the user authenticates.
```

```
    ImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd
    'Determine if a file was selected or cancel was pressed.
    If ImgAdmin1.Image = "" Then Exit Sub
    'Set the image properties in the Image Edit and Thumbnail
    'controls to the name of the file selected in the dialog box.
    ImgEdit1.Image = ImgAdmin1.Image
    ImgThumbnail1.Image = ImgAdmin1.Image
    'Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display
End Sub
```

Example 2 — SaveAs

This example saves the displayed image with a new filename using the Admin Save dialog and the Image Edit control SaveAs method.

```
Private Sub cmdSaveAs_Click()
    ImgAdmin1.ShowFileDialog SaveDlg
    ImgEdit1.SaveAs ImgAdmin1.Image
End Sub
```

ShowFileDialog Example – VC++

Example 1 — Open

This example shows how to use the ImgAdmin Open dialog box to display a file in the Image Edit and the Thumbnail controls.

```
void CAdmin1Dlg::OnFileopen()
{
    // Initialize Image property in order to check for Cancel pressed. After
     // the dialog box is shown, check for a value in the image property.
    ImgAdmin1.SetImage ("" );
    // Specify a dialog title and a list filter rather than using the defaults.
    ImgAdmin1.SetDialogTitle ("Select Image for Display");
    ImgAdmin1.SetFilter ("All Image Files |*.bmp; *.jpg; *.tif|
    ➥    Bitmap Files (*.bmp)|*. bmp|JPG Files (*.jpg)|*.jpg|TIFF Files
    ➥    (*.tif)|*.tif|All Files (*.*)|*.*|");
    // Default filter will be TIFF (fourth item).
    ImgAdmin1.SetFilterIndex (4);
    // Set the InitDir property to the preferred directory.
    ImgAdmin1.SetInitDir ("C:\\images");
    ImgAdmin1.SetCancelError (FALSE);
    // Note: If you have Server Access software installed this will
    // automatically be enabled in the Open dialog box after the user
    // authenticates.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (0,vhWnd);       // OpenDlg
    // Determine if a file was selected or cancel was pressed.
    if (ImgAdmin1.GetImage() =="")
            return;
    // Set the image properties in the ImgEdit and Thumbnail control
```

```
    // to the name of the file selected in the dialog box.
    ImgEdit1.SetImage (ImgAdmin1.GetImage());
    ImgThumbnail1.SetImage (ImgAdmin1.GetImage());
    // Display the image in the ImgEdit and Thumbnail control.
    ImgEdit1.Display();
}
```

Example 2 — SaveAs

This example saves the displayed image with a new filename using the Admin Save dialog and the Image Edit control SaveAs method.

```
void CAdmin1Dlg::OnSaveas()
{
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (1,vhWnd);   // OpenDlg
    VARIANT evt;
    evt.vt = VT_ERROR;     // set to error for optional parameter
    ImgEdit1.SaveAs (ImgAdmin1.GetImage(), evt, evt, evt, evt, evt);
}
```

ShowFileProperties Method

Description Displays a dialog box (see page 673) from which a user can view General file properties, and view and change Summary properties for TIFF files.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage

```
object.ShowFileProperties OriginalName
            [,bShowImageTabForOriginalFile = {True|False},
            bIsReadOnly = {True|False}]
```

Arguments The ShowFileProperties method has the following arguments:

Parameter	Data Type	Description
OriginalName	String	Name of the file to appear in the dialog box's caption bar. The value must be a fully qualified path name for the file.
BShowImageTab ForOriginalFile	Boolean	Optional. Shows or hides the image tab. True — Preview tab is displayed. False — (default) Preview tab is hidden.

Parameter	Data Type	Description
bIsReadOnly	Boolean	Optional. Bring up the dialog box in read-only mode. True — Dialog box comes up in read-only mode. False — (default) Dialog box comes up in normal mode.

Remarks Properties can be modified on TIFF files only, and only when using Imaging for Windows Professional Edition. Properties are read-only when using Imaging for Windows 98 (the blsReadOnly parameter is always True).

If the current image is a local or redirected TIFF file, or a 1.x file (not a document) the user can modify the **Title**, **Subject**, **Author**, **Comments**, and **Keywords** properties by typing new information into the corresponding fields.

If the current image is a 1.x document, only the **Keywords** property can be set.

When any of these property values are changed, the **FilePropertiesClose** event is fired, which enables the application to save the original image.

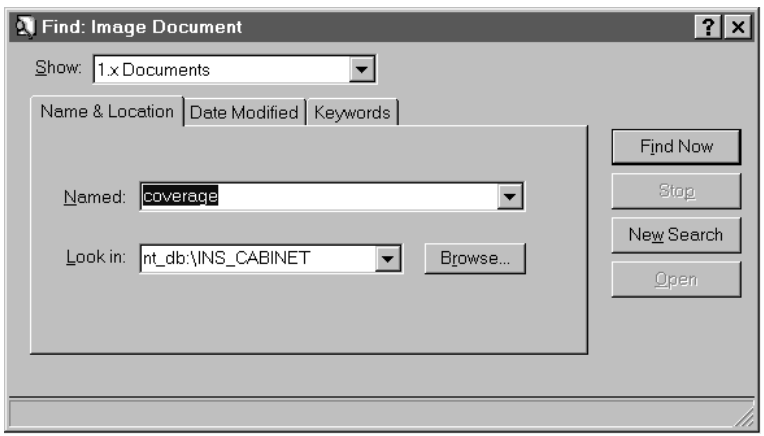
The modified property values are not saved to disk until the OK button in the dialog box is clicked.

Changes to the **Image** property update the property values displayed in the dialog box.

See Also Author property, Comments property, FilePropertiesClose event, Keywords property, Subject property, Title property.

ShowFindDialog Method

Description Displays the Find dialog box (shown here), from which a user specifies search parameters to 1.x and 3.x documents.



Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ShowFindDialog**([*hParentWnd*])

Arguments The ShowFindDialog method has the following argument:

Parameter	Data Type	Description
hParentWnd	hWnd	(Optional) Assigns a parent window handle to the displayed dialog box.

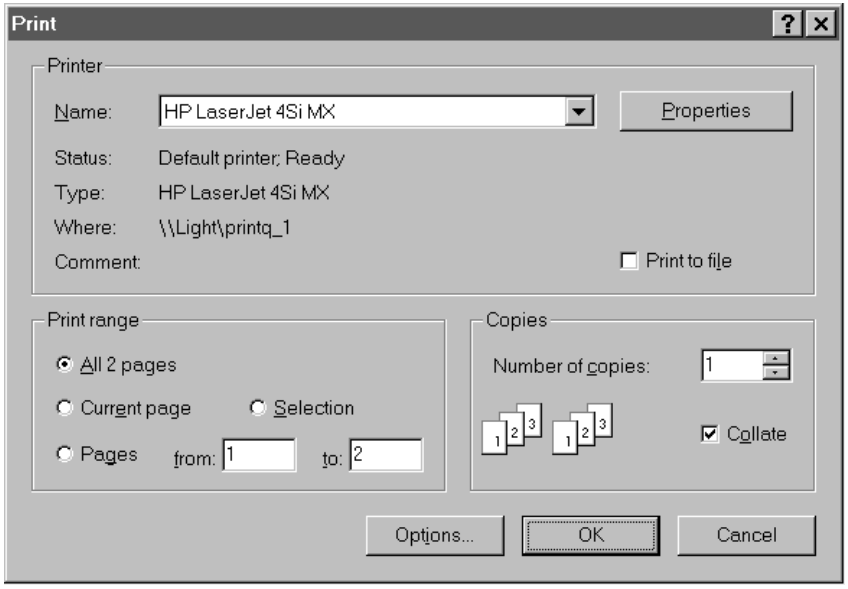
Remarks If a Parent window is not specified, the dialog box will behave as a modeless dialog box.

The document selected by the user from the dialog box is set in the **Image** property when the method returns the value.

See Also Image property.

ShowPrintDialog Method

Description Presents a dialog box (shown here) from which a user sets various print options.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowPrintDialog** [*hParentWnd*]

Arguments The ShowPrintDialog method has the following argument:

Parameter	Data Type	Description
hParentWnd	hWnd	(Optional) Assigns a parent window handle to the displayed dialog box.

Remarks The image property must be set to an existing image or an error is returned.

This function returns any parameters set by the user to the application via the corresponding properties.

If a Parent window is not specified, the dialog box will behave as a modeless dialog box.

See Also CancelError property, Flags property, PrintAnnotations property, PrintCollate property, PrintEndPage property, PrintNumCopies property, PrintOutputFormat property, PrintRangeOption property, PrintStartPage property, PrintToFile property.

ShowPrintDialog Example – VB

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
Private Sub cmdPrint_Click()
    On Error GoTo PrintErr
    'Display an image.
    ImgEdit1.Image = "D:\image2\4page.tif"
    ImgEdit1.Display
    'Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.PrintNumCopies = 1
    'If CancelError is true, an error is generated if user presses
    'cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.CancelError = True
    'Set filename to be printed to the displayed file. If this
    'property is not set the dialog box will not display.
    ImgAdmin1.Image = ImgEdit1.Image
    ImgAdmin1.ShowPrintDialog Form1.hWnd
    'Print the image using the parameters obtained from the print
    'dialog box (for example, start page, end page, and so on).
    ImgEdit1.PrintImage ImgAdmin1.PrintStartPage, ImgAdmin1.PrintEndPage,
    ➥    ImgAdmin1.PrintOutputFormat, ImgAdmin1.PrintAnnotations
```

```
PrintErr:
    'User pressed the cancel button.
       Exit Sub
End Sub
```

ShowPrintDialog Example – VC++

This example uses the ShowPrintDialog method to enable the user to specify printing parameters. The image displayed in the Image Edit control is then printed.

```
void CAdmin1Dlg::OnPrint()
{
    // Display an image.
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();
    // Reset NumCopies in case user printed multiple copies last time.
    ImgAdmin1.SetPrintNumCopies (1);
    // If CancelError is true, an error is generated if user presses
    // cancel. Trap the error to avoid trying to print the file.
    ImgAdmin1.SetCancelError (TRUE);
    // Set filename to be printed to the displayed file. If this
    // property is not set the dialog box will not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowPrintDialog (vhWnd);
    // Print the image using the parameters obtained from the print
    // dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;    // set to error for optional parameter
    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) = ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat, vAnnotations, evt,
    ➥    evt, evt);
}
```

VerifyImage Method

Description Determines if the image specified in the Image property exists, and verifies level of access.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**VerifyImage**(*Option*)

Arguments The VerifyImage method has the following argument:

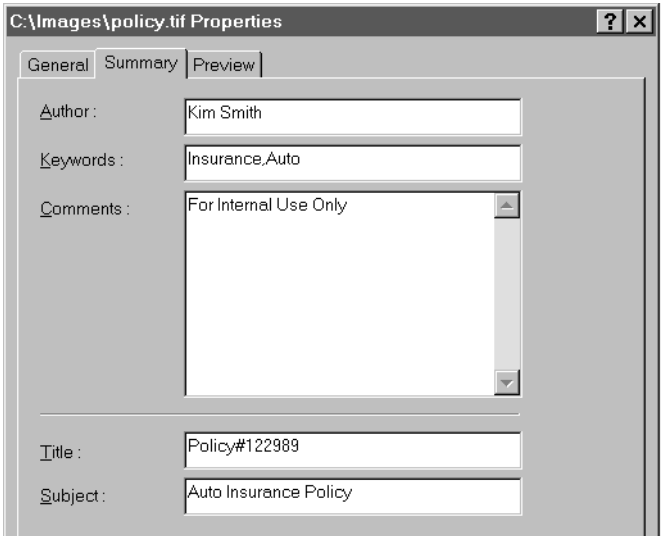
Parameter	Data Type	Description
Option	Long	Specifies the level of access to verify: VerifyExists (Literal 0) = Verify existence VerifyRead (Literal 0) = Verify read access VerifyWrite (Literal 2) = Verify write access VerifyReadWrite (Literal 3) = Verify read/write access

Returns Boolean.

Setting	Description
True	The file exists and has the level of access specified by the Option argument.
False	The file does not exist, or does not have the level of access specified in the Option argument.

FilePropertiesClose Event

Description This event fires when the end user changes properties on the File Properties dialog box and then closes it (subject to the restrictions in the Remarks section).



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage **Sub** *object*_**FilePropertiesClose**

Arguments None.

Returns You must set the bIsReadOnly parameter in the **ShowFileProperties** method to False for this event to fire. The end user must also modify one or more of the properties on the File Properties dialog box.

See Also ShowFileProperties method

Image Admin Extender Properties

This topic lists the Extender properties that are available when the Image Admin control is drawn on a Visual Basic form. Extender properties are provided by the container of the control, rather than by the control itself. Some Extender properties are standard for all containers, while others provided only by specific containers. For more information about Extender properties, refer to the documentation that came with your development environment.

Name	Description
Index	Returns or sets the subscript value of a control in an array of controls.
Name	Returns the name of an object.
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
Tag	Returns or sets ancillary data.

Refer to the Visual Basic on-line help for more information about these properties.

Image Page

Image pages provide a way to organize multiple images in an image file. Each page contains or stores one image. The following file types support multiple pages:

- AWD
- DCX
- TIFF
- WIFF
- XIF

Note: AWD is not available with Imaging for Windows NT 4.0. WIFF is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0, as well as with Imaging for Windows 98.

Display Types

The display type pertains to the way in which data is presented to the monitor. It does not alter the original image in any way. Imaging ActiveX controls support the following display types:

Black-and-White (Bi-Level) — Displays an image as black and white. The Black-and-White display type is typically used for applications that manage black-and-white documents exclusively.

Common — Displays an image using the common palette. The Common display type permits the display of multiple images simultaneously while maintaining reasonable image quality. A monitor and controller capable of displaying 256 colors are recommended.

Custom — Displays an image using the best palette possible. The palette used depends on the type of image being displayed, as described in the following list:

Image	Palette Used
Black-and-White	Binary
4-Bit Gray-Scale	Common
8-Bit Gray-Scale	Gray8
24-Bit Color Images	Common

Because the palette used is designed specifically for the type of image being displayed, the custom display type is best suited to application programs that display one image at a time. A monitor and controller capable of displaying 256 colors are recommended.

Gray8 — Displays an image using a linear-scale palette of up to 128 shades of gray.

RGB24 — Displays an image by letting Windows map the image directly to the monitor. With suitable hardware, the RGB24 display type produces the best color at the optimum speed. Further, it allows multiple images to be displayed with no loss in background image quality. A monitor and controller capable of displaying 32,000 colors are recommended.

Summary Properties

Also known as File properties, Summary properties contain information about TIFF image documents.

There are five property types:

- Author
- Comments
- Keywords
- Subject
- Title

The property types provide a way to organize information about TIFF image documents. For example, the Author property could contain the name of the person who scanned the image document (“Dave Monroe”), while the Keyword property could describe the image document (“Purchase Order”).

The Image Admin control has several properties and methods that enable you to add Summary property functions to your image-enabled applications.

If implemented in your program, end users can enter and maintain Summary property information. They will also be able to search for image documents by property content using the Find Image Files dialog box, which can be invoked via the ShowFileDialog method of the Image Admin control.

Note: Reading and writing of Summary properties is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only. Summary properties are read-only with Imaging for Windows 98.

9

Image OCR Control

This chapter describes the properties, methods, and events for the Image OCR control. In addition to the methods and events described in this chapter, the following methods and events, which are described in Chapter 7, apply to the Image OCR control:

- AboutBox method
- GetVersion method
- ReadyStateChange event

In This Chapter

What the Image OCR Control Lets You Do

The Image OCR control lets you — the application developer — add optical character recognition functions to applications that support 32-bit ActiveX controls.

The OCR process analyzes an image and identifies text, pictures, and layout. This information is used to reconstruct the document.

You can design your program to process a document automatically or manually. For manual processing, you would prompt the user for layout details (such as single or multiple columns; text, pictures, or both) of the source document. If TIFF images are used, you can provide an option to define text or picture regions of the document with a selection rectangle. This can be useful for processing a single paragraph in the document.

User-defined dictionaries can be used to complement the standard OCR dictionaries, thereby improving word recognition.

The OCR process can be trained to recognize special characters or words. You can display a dialog box that lists words or symbols that the program is unsure of, and enable the user to provide corrections. The corrections and other information are saved in a training file that can be reused.

Note: OCR processing is available with Imaging for Windows Professional Edition V1.0, V1.1, and V2.0 only.

What the Image OCR Control Lets Your Users Do

Depending on how you design and code your application, the Image OCR control lets your users convert scanned image documents to formatted, editable text.

Users can convert image documents to one of the following formats:

- MS Word for Windows/RTF
- WordPerfect
- HTML
- ASCII text file

CopytoClipBoard Property

Description Specifies that the OCR results are to be placed on the Windows Clipboard.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**CopytoClipBoard**[= {*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Place results onto the Clipboard.
False (default)	Do not place results on the Clipboard.

CopyToClipBoard Example — VB

This example demonstrates how OCR results can be output to the clipboard instead of to an output file of the supported types.

```
Private Sub cmdCopyToClip_Click()
    'If the output type has been set to Text Document, then copy the output
    'to the clipboard. Else save it to a document and launch the associated
    'application.
    If Imgocr1.OutputType = wiAsciiText Then
        Imgocr1.CopyToClipBoard = True
    Else
        Imgocr1.OutputFile = "c:\my documents\instructions"
        Imgocr1.LaunchApplication = True
    End If
        Imgocr1.StartOcr 'Begin OCR process
End Sub
```

Image Property

Description Returns or sets the file name (see page 535) of the current image.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Image**[=*filename*]

Data Type String.

Remarks The source image must be a TIFF file located on a local or redirected drive, or on a 1.x or 3.x server.

Unless you are performing an OcrFromClipBoard function, you must set this value before processing can be performed.

Image Example — VB

This example demonstrates how various images can be specified by the Image property. It also shows how specific pages within the document can be designated for OCRing by the Pages property.

```
Private Sub SetImage_Click()
    'Showing syntax to have page 1 of a 1.x Server document OCRed.
    'Note: prefix Image:// denotes service name for 1.x server.
    Imgocr1.Image = "Image://srvrname\volname:\CAB\DRAWER\FOLDER\YOURDOC"
    Imgocr1.Pages = "1"
    'Showing syntax to have pages 1 & 4 of a 1.x Server filename OCRed.
    Imgocr1.Image = "Image://srvrname/volname:/dirname/temp.tif"
    Imgocr1.Pages = "1,4"
    'Specifying a 3.x Server document to OCR pages 3,4, & 5.
    'Note: prefix Imagex:// denotes service name for 3.x server.
    Imgocr1.Image = "Imagex://Docid1234"
    Imgocr1.Pages = "3-5"
    'Specify an internet image via URL syntax to OCR pages 1,2, & 5.
    Imgocr1.Image = "http://www.eastmansoftware.com/sbu/sampimg.tif"
    Pages = "1-2,5"
    'Specify a UNC filename to OCR pages 1,2,5,6, & 7.
    Imgocr1.Image = "\\srvrname\shared3\images\bw\thisisa.tif"
    Imgocr1.Pages = "1-2,5-7"
End Sub
```

Language Property

Description Sets the language identifier for OCR processing.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Language**[=*value*]

Data Type Long.

Constant	Setting	Description
wiSystemDefault	0 (default)	System default
wiEnglish	1	English
wiSpanish	2	Spanish
wiGerman	3	German
wiItalian	4	Italian
wiFrench	5	French
wiDutch	6	Dutch
wiSwedish	7	Swedish

Remarks Set to the language you want to use. Setting the Language property improves recognition of special characters unique to the specified language.

Language Example — VB

This example shows how OCR Options - including dictionary, language, and OCR training - might be set prior to performing OCR on the current image.

```
Private Sub cmdSetTraining_Click()
    Imgocr1.Language = wiEnglish      'Recognize English characters
    Imgocr1.LoadDictionary ("c:\Program Files\Common Files\Microsoft
    ➥   Shared\Proof\custom.dic")
    Imgocr1.TrainingFile = "d:\windows\system\german.trn"
    Imgocr1.TrainingFileOptions = 1  'Indicate training is to be interactive
    Imgocr1.TrainingThreshold = 400  'Prompt user for correction if
                                     'confidence level < 400
    Imgocr1.StartOcr
End Sub
```

LaunchApplication Property

Description Specifies whether the application that supports the output file is launched upon completion of OCR processing.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**LaunchApplication**[= {*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Launches the application.
False	Does not launch the application.

LaunchApplication Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones
    Imgocr1.RetainPictures = False   'Do not process pictures
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

Layout Property

Description Specifies the layout of the image pages to be processed. For information on layout types, see page 716.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Layout**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiAutoDetectL	0 (default)	Auto detect — the layout of the original is used in the output.
wiSingleNoPictures	1	Single column with no pictures.

Constant	Setting	Description
wiSingleWithPictures	2	Single column with pictures.
wiMultipleNoPictures	3	Multiple columns with no pictures.
wiMultipleWithPictures.	4	Multiple columns with pictures.

Layout Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones
    Imgocr1.RetainPictures = False   'Do not process pictures
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

OcrFromClipBoard Property

Description Specifies that the image data on the Windows Clipboard is the source for OCR processing.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**OcrFromClipBoard**[= {*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Use data from the Clipboard.
False (default)	Do not use data from the Clipboard.

Remarks Clipboard image data must be in vbCF_DIB format.

Setting this property enables the Selection pick on the ShowOCR dialog box (see page 708).

OcrFromClipBoard Example — VB

This example demonstrates how to perform OCR on clipboard data pasted from the currently displayed image.

```
Private Sub cmdOcrClipBoard_Click()
    'Copy an area from the current image to the clipboard.
    ImgEdit1.SelectionRectangle = True
    ImgEdit1.DrawSelectionRect 0, 0, 200, 200
    ImgEdit1.ClipboardCopy
    'OCR the clipboard contents to a Text Document and launch
    'its associated application for viewing.
    Imgocr1.OcrFromClipBoard = True
    Imgocr1.OutputType = wiAsciiText
    Imgocr1.OutputFile = "c:\My Documents\sample.txt"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

OutputFile Property

Description Specifies the name of the output file to which OCR processing results are written.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**OutputFile**[=*filename*]

Data Type String.

Remarks Any valid Windows 95 filename can be specified. If the output file specified already exists, it will be overwritten.

If a filename is not provided, the Save As dialog box is displayed when OCR processing is complete.

For best results, the file extension set by this property must match the OutputType property. For example, use the .txt extension if you set ASCII text as the output type.

OutputFile Example

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
```

```
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones.
    Imgocr1.RetainPictures = False   'Do not process pictures.
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

OutputType Property

Description Specifies the format for the output file. For information on output file types, see page 717.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**OutputType**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
wiWordforWindows	0 (default)	Microsoft Word for Windows (versions 6 and 7) and RTF
wiWordPerfect	1	Corel WordPerfect, version 6.1
wiHTML	2	Hypertext Markup Language (HTML)
wiAsciiText	3	ASCII Text

Remarks For best results, the file extension set by the OutputFile property must match the OutputType value. For example, use the .txt extension if ASCII text is the output type.

OutputType Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
```

```
    Imgocr1.ProcessingMode = 1       'Process only OCR zones.
    Imgocr1.RetainPictures = False   'Do not process pictures.
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

Pages Property

Description Specifies which pages to OCR.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Pages**[=*value*]

Data Type String.

Remarks Single page values (for example, 2 or 5), multiple pages (for example, 1, 3, 9), or page ranges (for example, 2-6 or 1-9) can be used.

The default value is a blank string, which means all pages are processed.

Pages Example — VB

This demonstrates how the Image property can specify various images. It also shows how specific pages within the document can be designated for OCRing by the Pages property.

```
Private Sub SetImage_Click()
    'Showing syntax to have page 1 of a 1.x Server document OCRed.
    'Note: prefix Image:// denotes service name for 1.x server.
    Imgocr1.Image = "Image://srvrname\volname:\CAB\DRAWER\FOLDER\YOURDOC"
    Imgocr1.Pages = "1"
    'Showing syntax to have pages 1 & 4 of a 1.x Server filename OCRed.
    Imgocr1.Image = "Image://srvrname/volname:/dirname/temp.tif"
    Imgocr1.Pages = "1,4"
    'Specifying a 3.x Server document to OCR pages 3,4, & 5.
    'Note: prefix Imagex:// denotes service name for 3.x server.
    Imgocr1.Image = "Imagex://Docid1234"
    Imgocr1.Pages = "3-5"
    'Specify an internet image via URL syntax to OCR pages 1,2, & 5.
    Imgocr1.Image = "http://www.eastmansoftware.com/sbu/sampimg.tif"
    Pages = "1-2,5"
```

```
    'Specify a UNC filename to OCR pages 1,2,5,6, & 7.
    Imgocr1.Image = "\\srvrname\shared3\images\bw\thisisa.tif"
    Imgocr1.Pages = "1-2,5-7"
End Sub
```

ProcessingMode Property

Description Specifies which OCR processing mode to use.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ProcessingMode**[=*value*]

Data Type Long.

Constant	Setting	Description
wiManual	0	Manual processing, using OCR Zones for all pages specified.
wiAutomatic	1 (default)	Automatic processing of all specified pages.

Remarks You cannot perform manual processing on an image that contains only picture zones. At least one text zone must also be included on the page.

See Also AnnotationOcrType property, AnnotationType property, OCR zones.

ProcessingMode Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = wiAutomatic 'Process only OCR zones.
    Imgocr1.RetainPictures = False  'Do not process pictures.
    Imgocr1.RetainPageLayout = True 'Source layout retained in output layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

ProgressDialogCaption Property

Description Provides the text that is displayed in the Progress dialog box (see page 701).

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ProgressDialogCaption**[=*text*]

Data Type String.

Remarks If a string is not specified, the name of the TIFF file being processed is used.

The default value contains an empty, Null terminated string.

See Also ShowProgress property.

ProgressDialogCaption Example — VB

When the OcrProgress event is fired, the Progress dialog is updated at various intervals to indicate the percent the OCR operation has completed for the current page.

```
Private Sub Imgocr1_OcrProgress(ByVal Percentage As Integer)
'NOTE: The ProgressNotification property must be set to TRUE in order
'for progress events to be sent to the application.
    Imgocr1.ShowProgress = True   'Show progress dialog.
    Select Case Percentage
        Case Is > 95
            Imgocr1.ProgressDialogCaption = "almost complete"
        Case Is > 74
            Imgocr1.ProgressDialogCaption = " ~75% complete"
        Case Is > 49
            Imgocr1.ProgressDialogCaption = " ~50% complete"
        Case Is > 24
            Imgocr1.ProgressDialogCaption = " ~25% complete"
        Case Else
            Imgocr1.ProgressDialogCaption = " mydoc.tif"
        End Select
End Sub
```

ProgressNotification Property

Description Specifies whether progress events are sent to the application.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ProgressNotification**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Send progress events.
False (default)	Do not send progress events.

Remarks Progress information pertains to the current page being processed.

See Also OCRProgress event.

ProgressNotification Example — VB

When the OCRProgress event is fired, the Progress dialog is updated at various intervals to indicate the percent the OCR operation has completed for the current page.

```
Private Sub Imgocr1_OcrProgress(ByVal Percentage As Integer)
'NOTE: The ProgressNotification property must be set to TRUE in order
'for progress events to be sent to the application.
    Imgocr1.ProgressNotification = True   'Send progress events.
    Imgocr1.ShowProgress = True           'Show progress dialog.
    Select Case Percentage
        Case Is > 95
            Imgocr1.ProgressDialogCaption = "almost complete"
        Case Is > 74
            Imgocr1.ProgressDialogCaption = " ~75% complete"
        Case Is > 49
            Imgocr1.ProgressDialogCaption = " ~50% complete"
        Case Is > 24
            Imgocr1.ProgressDialogCaption = " ~25% complete"
        Case Else
            Imgocr1.ProgressDialogCaption = " mydoc.tif"
        End Select
End Sub
```

Quality Property

Description Specifies the resolution quality of the pages to be processed. For information on input quality types, see page 718.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**Quality**[=*value*]

Data Type Long.

Constant	Setting	Description
wiAutoDetectQ	0 (default)	Auto detect
wiFax	1	FAX
wiDotMatrix	2	Dot matrix printer
wiNormal	3	Normal

Quality Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones
    Imgocr1.RetainPictures = False   'Do not process pictures
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

ReadyState Property

Description Returns the state of the OCR control's properties.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ReadyState**[=*value*]

Data Type Long.

Setting	Description
0	Control is initializing and retrieving properties.
2	Control is initialized and the download process has started.
4	Control is ready for all requests.

Remarks This property is read-only.

Used for asynchronous downloading of images from a Web site (**Image** property set to a URL).

When downloading images from a Web site, you cannot call the **StartOcr** method until the **ReadyState** property is set to 4.

If the image file specified is not a URL, the **ReadyState** property will always have a default value of 4.

See Also ReadyStateChange event.

ReadyState Example — VB

This example demonstrates how the ReadyState property can be captured and generate a message box indicating the current state of the control.

```
Private Sub Imgocr1_ReadyStateChange(ReadyState As Long)
    sCaption = "Ready State"
    Select Case ReadyState
        Case 0
            MsgBox "Control is initializing and retrieving properties.", ,
            ➥   sCaption
        Case 2
            MsgBox "Control is initialized; download has started.", ,
            ➥   sCaption
        Case 4
            MsgBox "Control is ready.", , sCaption
    End Select
End Sub
```

RetainPageLayout Property

Description Specifies whether the column layout of the image document being processed is retained in the output.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**RetainPageLayout**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Retain column layout.
False	Do not retain column layout.

Remarks This property does not apply to .txt format.

Columns are not retained in HTML, but this property affects the placement of pictures in the HTML file (if applicable).

RetainPageLayout Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones.
    Imgocr1.RetainPictures = False   'Do not process pictures.
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

RetainPictures Property

Description Specifies whether pictures from the image document being processed are retained in the output.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**RetainPictures**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Retain pictures.
False	Do not retain pictures.

Remarks This property does not apply to .txt format.

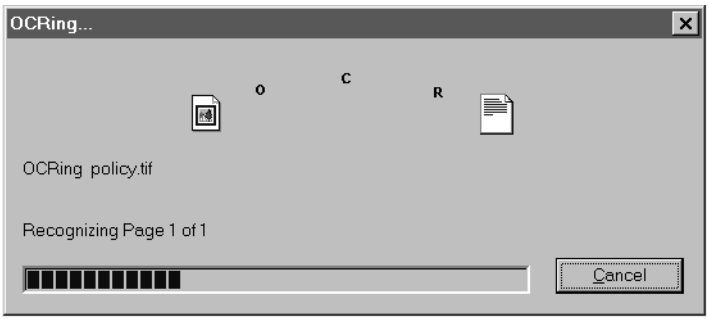
RetainPictures Example — VB

This example shows how various OCR input and output properties might be set prior to performing OCR on an image.

```
Private Sub cmdLayout_Click()
    'Set the OCR input options of the source file.
    Imgocr1.Quality = wiAutoDetectQ
    Imgocr1.Layout = wiMultipleWithPictures
    Imgocr1.ProcessingMode = 1       'Process only OCR zones.
    Imgocr1.RetainPictures = False   'Do not process pictures.
    Imgocr1.RetainPageLayout = True  'Source layout retained in output
                                     'layout
    'Set the OCR output file to HTML type and open it in the associated
    'application.
    Imgocr1.OutputType = wiHTML
    Imgocr1.OutputFile = "d:\My Documents\ArticleText.html"
    Imgocr1.LaunchApplication = True
    Imgocr1.StartOcr
End Sub
```

ShowProgress Property

Description Specifies if an OCR progress dialog box (shown here) is displayed during the OCR process.



Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ShowProgress**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Display dialog box.
False	Do not display dialog box.

See Also ProgressDialogCaption property.

ShowProgress Example — VB

When the OCRProgress event is fired, the Progress dialog is updated at various intervals to indicate the percent the OCR operation has completed for the current page.

```
Private Sub Imgocr1_OcrProgress(ByVal Percentage As Integer)
'NOTE: The ProgressNotification property must be set to TRUE in order
'for progress events to be sent to the application.
    Imgocr1.ShowProgress = True    'Show progress dialog.
    Select Case Percentage
        Case Is > 95
            Imgocr1.ProgressDialogCaption = "almost complete"
        Case Is > 74
            Imgocr1.ProgressDialogCaption = " ~75% complete"
        Case Is > 49
            Imgocr1.ProgressDialogCaption = " ~50% complete"
```

```
        Case Is > 24
            Imgocr1.ProgressDialogCaption = " ~25% complete"
        Case Else
            Imgocr1.ProgressDialogCaption = " mydoc.tif"
        End Select
End Sub
```

StatusCode Property

Description Reports the error code that occurred after a property was set or a method called.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**StatusCode**[=*value*]

Data Type Integer (enumerated).

Value	Description
0 (default)	Success.
1	OCR results cannot be converted to the format specified.
2	Unable to start OCR processing.
3	The OCR server cannot be initialized – it may already be running.
4	An invalid converter name was specified.
5	An invalid filename was specified.
6	An invalid OCR property was specified.
7	The OCR dictionaries cannot be loaded.
8	OCR language support cannot be loaded.
9	OCR processing canceled due to an unrecoverable recognition error.
10	Unable to shut down the OCR server.
11	Image filename not found.
12	Unable to perform processing because the OCR server was not started.
13-17	Reserved.
18	Cannot create the OCR output file.
19	OCR cannot open or read the image file.
20	OCR cannot preprocess the image page.
21	OCR cannot recognize the image.

Value	Description
22	OCR settings, or the OCR dialog box was canceled.
23	Reserved.
24	OCR cannot turn on page layout support.
25	Unable to set the internal timer for OCR processing.
26	Cannot open, read, or save Training File information.
27	An error occurred while accessing the Windows clipboard.
28	OCR processing was canceled by the user.
29	Reserved.
30	An error occurred while converting to a bi-tonal image.
31	An error occurred while converting to a bi-tonal image.
32	A Windows error was received while converting to a bi-tonal image.
33	A TextBridge error was received while converting to a bi-tonal image.
34	The OCR server task is no longer running.
35	The pages specified contain no valid pages to OCR.
36	An Output Document name was not specified in the SaveAs dialog.
37	Unable to delete ICR output document.
38	OCR style sheet could not be created.
39	Text was not found,or not recognized.
40	Unsupported image resolution.

Remarks This is a read-only property.

StatusCode Example — VB

This example demonstrates how the OcrComplete event may be used to inform the user of the outcome of the OCR process.

```
Private Sub Imgocr1_OcrComplete(ByVal Status As Integer)
    If Status = 100 Then
        'Display successful OCR message.
        MsgBox "OCR completed successfully.", , "OCR Status"
    Else
        'Display message indication OCR was unsuccessful and an error code.
        MsgBox "OCR unsuccesful; Status code = " &
        ➥   Str(Imgocr1.StatusCode), , "OCR Status"
    End If
End Sub
```

TrainingFile Property

Description Specifies the name of the training file.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**TrainingFile**[=*filename*]

Data Type String.

Remarks This property must be set before any training functions can be performed. The default value is an empty string, which means training is not performed.

See Also TrainingFileOptions property.

TrainingFile Example — VB

This example shows how OCR Options - including dictionary, language, and OCR training - might be set prior to performing OCR on the current image.

```
Private Sub cmdSetTraining_Click()
    Imgocr1.Language = wiEnglish      'Recognize English characters.
    Imgocr1.LoadDictionary ("c:\Program Files\Common Files\Microsoft
    ➥   Shared\Proof\custom.dic")
    Imgocr1.TrainingFile = "d:\windows\system\german.trn"
    Imgocr1.TrainingFileOptions = 1  'Indicate training is to be
                                     'interactive.
    Imgocr1.TrainingThreshold = 400  'Prompt user for correction if
                                     'confidence level < 400
    Imgocr1.StartOcr
End Sub
```

TrainingFileOptions Property

Description Specifies how the training file is used.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**TrainingFileOptions**[=*value*]

Data Type Long.

Constant	Setting	Description
wiSave	0 (default)	The training file specified in the TrainingFile property is used without interactive training.
wiInteractiveSave	1	The training file specified is used, interactive training occurs, and the data is saved to the file specified by the TrainingFile property.

Remarks If the TrainingFile property is not set, this property is ignored.

See Also TrainingFile property.

TrainingFileOptions Example — VB

This example shows how OCR Options - including dictionary, language, and OCR training - might be set prior to performing OCR on the current image.

```
Private Sub cmdSetTraining_Click()
    Imgocr1.Language = wiEnglish      'Recognize English characters.
    Imgocr1.LoadDictionary ("c:\Program Files\Common Files\Microsoft
    ➥   Shared\Proof\custom.dic")
    Imgocr1.TrainingFile = "d:\windows\system\german.trn"
    'Indicate training is to be interactive.
    Imgocr1.TrainingFileOptions = wiInteractiveSave
    Imgocr1.TrainingThreshold = 400  'Prompt user for correction if
                                     'confidence level < 400
    Imgocr1.StartOcr
End Sub
```

TrainingThreshold Property

Description Specifies a threshold value that is used by the interactive training mode.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**TrainingThreshold**[=*value*]

Data Type Long.

Remarks Any character with a confidence value less than the specified threshold value causes the program to prompt the user for correction. Valid values can be 0 to 999, inclusive. The default value is 200.

TrainingThreshold Example — VB

This example shows how OCR Options - including dictionary, language, and OCR training - might be set prior to performing OCR on the current image.

```
Private Sub cmdSetTraining_Click()
    Imgocr1.Language = wiEnglish    'Recognize English characters.
    Imgocr1.LoadDictionary ("c:\Program Files\Common Files\Microsoft
    ➥   Shared\Proof\custom.dic")
    Imgocr1.TrainingFile = "d:\windows\system\german.trn"
    Imgocr1.TrainingFileOptions = 1  'Indicate training is to be
                                     'interactive.
    Imgocr1.TrainingThreshold = 400  'Prompt user for correction if
                                     'confidence level < 400
    Imgocr1.StartOcr
End Sub
```

LoadDictionary Method

Description Loads a user-defined dictionary for the OCR server to use. For information on user-defined dictionaries, see page 718.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**LoadDictionary** *DictionaryFile*

Arguments The LoadDictionary method has the following parameter:

Parameter	Data Type	Description
DictionaryFile	String	The name of a dictionary

Returns Boolean.

The return value indicates if a dictionary was loaded.

Setting	Description
True	The dictionary specified was loaded.
False	The dictionary was not loaded.

Remarks If you want to use a dictionary, call this method before calling the StartOcr method.

If a dictionary is specified, it is loaded.

LoadDictionary Example — VB

This example shows how OCR Options - including dictionary, language, and OCR training - might be set prior to performing OCR on the current image.

```
Private Sub cmdSetTraining_Click()
    Imgocr1.Language = wiEnglish      'Recognize English characters.
    Imgocr1.LoadDictionary ("c:\Program Files\Common Files\Microsoft
    ➥   Shared\Proof\custom.dic")
    Imgocr1.TrainingFile = "d:\windows\system\german.trn"
    Imgocr1.TrainingFileOptions = 1  'Indicate training is to be
                                     'interactive.
    Imgocr1.TrainingThreshold = 400  'Prompt user for correction if
                                     'confidence level < 400.
    Imgocr1.StartOcr
End Sub
```

SetDefaultValues Method

Description Sets all property values to their defaults.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetDefaultValues** [*UseSystemSettings*]

Arguments The SetDefaultValues method has the following parameter:

Parameter	Data Type	Description
UseSystemSettings	Boolean	Optional. False — (default) restore to original OCR default values True — restore to defaults in the registration database

Returns None.

SetDefaultValues Example — VB

Prior to starting OCR process, this example prompts the user to

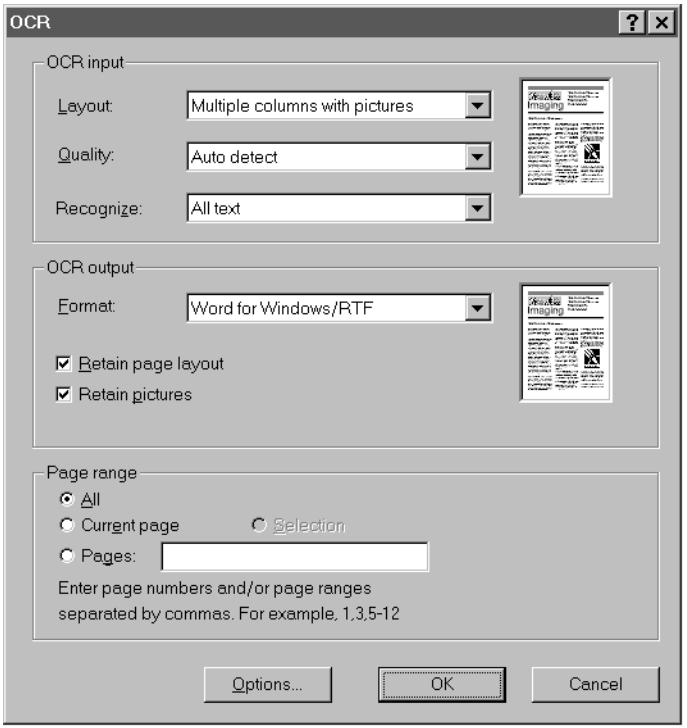
- set layout, output and page properties.
- set OCR options - language, dictionary, and training.

```
Private Sub cmdOcrOptions_Click()
    Imgocr1.SetDefaultValues (True)   'Return OCR properties to system
                                      'defaults.
    ret1 = Imgocr1.ShowOcr            'Display OCR layout and output dialog.
```

```
    If ret1 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.
    ret2 = Imgocr1.ShowOcrOptions          'Display OCR Options dialog.
    If ret2 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.
    Imgocr1.StartOcr
End Sub
```

ShowOcr Method

Description Displays a dialog box (shown here) for OCR settings.



Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ShowOcr**

Arguments None.

Returns Long.

Setting	Description
0	Dialog box was canceled.
1	OK was pressed.

Remarks The dialog box contains the following fields:

OCR Input	OCR Output	Page Range
Quality	Output Format	All
Layout	Page layout	Current Page
Recognize	Retain/discard pictures	Specified pages
		Selection

The following properties are used to set the initial values when the dialog box is opened.

- Layout
- OcrFromClipboard
- OutputType
- Pages
- Processing Mode
- Quality
- RetainPageLayout
- RetainPictures

If OcrFromClipboard is set to true, the selection field is enabled and selected.

Entries made to dialog box fields change the values of the properties listed above.

The Pages property is set to zero if Current Page is selected, and to a blank string if All is selected.

Before you call the StartOcr method, the Pages property must reflect the current page number.

The Options button opens the Options dialog box. Refer to the ShowOcrOptions method (see page 710).

See Also ShowOcrOptions method.

ShowOcr Example — VB

Prior to starting OCR process, this example prompts the user to

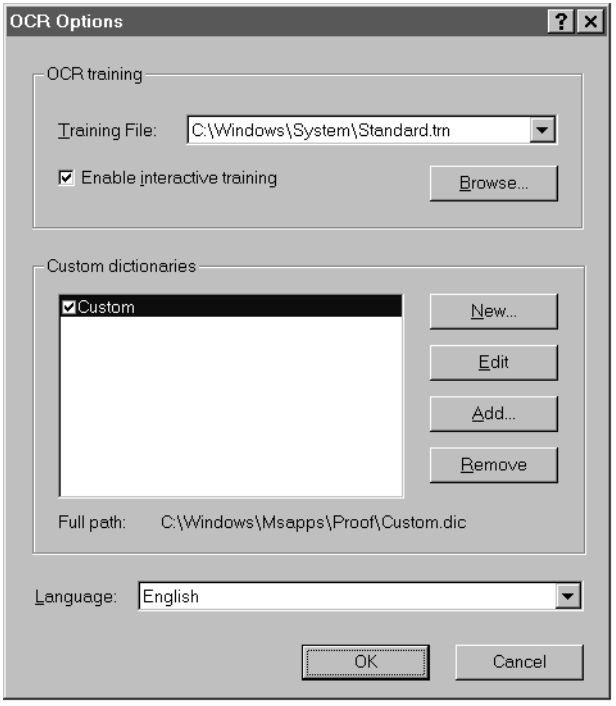
- set layout, output and page properties.
- set OCR options - language, dictionary, and training.

```
Private Sub cmdOCROptions_Click()
    Imgocr1.SetDefaultValues (True)        'Return training file options to
                                           'system defaults.
    ret1 = Imgocr1.ShowOcr                 'Display OCR layout and output
                                           'dialog.
```

```
    If ret1 = CancelDialog Then Exit Sub    'Do not OCR if user pressed
                                            'Cancel.
    ret2 = Imgocr1.ShowOcrOptions           'Display OCR Options dialog.
    If ret2 = CancelDialog Then Exit Sub    'Do not OCR if user pressed
                                            'Cancel.
    Imgocr1.StartOcr
End Sub
```

ShowOcrOptions Method

Description Displays the OCR Options dialog box (shown here) for selecting OCR processing options.



Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**ShowOcrOptions**

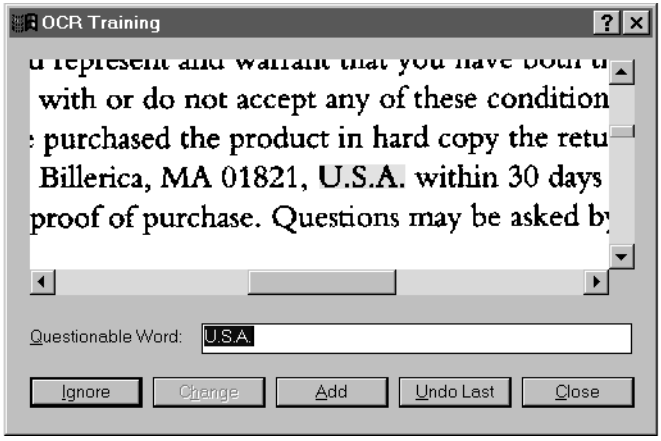
Returns Boolean.

Setting	Description
0	Dialog box was canceled.
1	OK was pressed.

Remarks The dialog box contains the following fields:

OCR Training	Custom Dictionaries	Language
Training file (see page 719) Interactive training (see page 718)	list box of available dictionaries (see page 718)	list box of available languages (see page 719)

If Interactive training (see page 718) is selected, a training dialog box (shown here) is displayed while the OCR process is running. If the training dialog box is closed, OCR processing continues, but the Training process stops.



The following properties are used to set the initial values when the dialog box is opened:

- Language
- TrainingFile
- TrainingFileOptions

Changing the values in the dialog box also changes the values for the properties listed above, and for the LoadDictionary method.

Any dictionaries that are selected are loaded when the user chooses the OK button.

ShowOcrOptions Example — VB

Prior to starting OCR process, this example prompts the user to

- set layout, output and page properties.
- set OCR options - language, dictionary, and training.

```
Private Sub cmdOCROptions_Click()
    Imgocr1.SetDefaultValues (True)        'Return training file options
                                           'to system defaults.
    ret1 = Imgocr1.ShowOcr                 'Display OCR layout and output
                                           'dialog.
    If ret1 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.
    ret2 = Imgocr1.ShowOcrOptions          'Display OCR Options dialog.
    If ret2 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.
    Imgocr1.StartOcr
End Sub
```

StartOcr Method

Description Perform character recognition and recomposition (see page 719).

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**StartOcr**

Arguments None.

Returns Boolean.

Setting	Description
True	OCR recognition completed and recomposition is in progress.
False	Recognition process halted.

Remarks The image is specified by the Image property.

StartOcr Example — VB

Prior to starting OCR process, this example prompts the user to

- set layout, output and page properties.
- set OCR options - language, dictionary, and training.

```
Private Sub cmdOCROptions_Click()
    Imgocr1.SetDefaultValues (True)        'Return training file options to
                                           'system defaults.
    ret1 = Imgocr1.ShowOcr                 'Display OCR layout and output
                                           'dialog.
    If ret1 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.
    ret2 = Imgocr1.ShowOcrOptions          'Display OCR Options dialog.
    If ret2 = CancelDialog Then Exit Sub   'Do not OCR if user pressed
                                           'Cancel.

    Imgocr1.StartOcr
End Sub
```

StopOcr Method

Description Cancels the OCR recomposition process.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**StopOcr**

Arguments None.

Returns Boolean.

Setting	Description
True	Recomposition processing is canceled.
False	Recomposition cannot be canceled.

StopOcr Example — VB

This example demonstrates how the StopOCR method can be invoked via a Cancel button. This might be the code behind the command button on a user-defined progress dialog.

Note: The standard OCR progress dialog is presented when the **ProgressNotification** property is set to TRUE.

```
Private Sub cmdCancelOCR_Click()
    'Stop the OCr process and close the progress dialog.
    Imgocr1.StopOcr
    Unload frmProgress
End Sub
```

OcrComplete Event

Description Signals the completion of OCR processing.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage **Sub** *object*_**OCRComplete**(*status*)

Arguments The OCRComplete event has the following parameter:.

Parameter	Data Type	Values
status	Short	100 — OCR completed successfully. -1 — An error occurred and recognition was terminated.

Remarks This event is triggered when processing is finished.

OcrComplete Example — VB

This example demonstrates how the OcrComplete event can be used to inform the user of the outcome of the OCR process.

```
Private Sub Imgocr1_OcrComplete(ByVal Status As Integer)
    If Status = 100 Then
        'Display successful OCR message.
        MsgBox "OCR completed successfully.", , "OCR Status"
    Else
        'Display message indication OCR was unsuccessful and an error code.
        MsgBox "OCR unsuccessful: Status code = " &
        ➥    Str(Imgocr1.StatusCode), , "OCR Status"
    End If
End Sub
```

OcrProgress Event

Description Indicates the degree of completion of the OCR processing operation.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage **Sub** *object*_**OCRProgress**(*percentage*)

Arguments The OCRProgress event has the following parameter:.

Parameter	Data Type	Values
percentage	Short	Percentage of completion from 0 to 100.

Remarks When this event is fired, the percent of the OCR operation completed for an image file is indicated.

The **ProgressNotification** property must be set to True for this event to occur.

OcrProgress Example — VB

When the OCRProgress event is fired, the Progress dialog is updated at various intervals to indicate the percent the OCR operation has completed for the current page.

```
Private Sub Imgocr1_OcrProgress(ByVal Percentage As Integer)
'NOTE: The ProgressNotification property must be set to TRUE in order
'for progress events to be sent to the application.
    Imgocr1.ShowProgress = True   'Show progress dialog
    Select Case Percentage
        Case Is > 95
            Imgocr1.ProgressDialogCaption = "almost complete"
        Case Is > 74
            Imgocr1.ProgressDialogCaption = " ~75% complete"
        Case Is > 49
            Imgocr1.ProgressDialogCaption = " ~50% complete"
        Case Is > 24
            Imgocr1.ProgressDialogCaption = " ~25% complete"
        Case Else
            Imgocr1.ProgressDialogCaption = " mydoc.tif"
        End Select
End Sub
```

Image OCR Extender Properties

This section lists the Extender properties that are available when the Image OCR control is drawn on a Visual Basic form.

Name	Description
DataBindings	Returns the DataBindings collection object containing the available bindable properties.
DataChanged	Returns or sets a value indicating that data in the bound control has changed.
DataField	Returns or sets a value that binds a control to a data field.
Index	Returns or sets the subscript value of a control in an array of controls.
Name	Returns the name of an object.
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
Tag	Returns or sets ancillary data.

Refer to the Visual Basic on-line help for more information about these properties.

Layout Types

If implemented in your program, the OCR control lets end users indicate the column layout and picture content of the original documents they are going to OCR. Indicating the layout type in advance of performing OCR can improve document recognition.

Auto detect — This setting is appropriate for original documents that contain pictures, tables, or spreadsheets. Imaging automatically detects column layout and picture content. Use of this setting is recommended in most cases.

Single column with no pictures — This setting is appropriate for one-column original documents without pictures, tables, or spreadsheets.

Single column with pictures — This setting is appropriate for one-column original documents that contain straight text and pictures. Imaging does not perform OCR on the pictures.

Multiple columns with no pictures — This setting is appropriate for multiple-column original documents without pictures, tables, or spreadsheets.

Multiple columns with pictures — This setting is appropriate for multiple-column original documents that contain straight text and pictures. Imaging does not perform OCR on the pictures.

Output Types

The Image OCR control enables end users to output image documents in the following types (formats):

Word for Windows/Rich Text Format (RTF) — Generates Rich Text Format (RTF) document files, which can be used by Microsoft Word. Can retain the layout and pictures of the original documents for document recomposition. Document files created have the extension .DOC.

WordPerfect — Generates document files formatted for WordPerfect. Can also retain the layout and pictures of the original documents for document recomposition. Document files created have the extension .WPD.

HTML — Generates HyperText Markup Language (HTML) document files, which are typically used on the World Wide Web. Can also retain the layout and pictures of the original documents for document recomposition, except that support for multiple columns is not provided. Document files created have the extension .HTM.

Text Document — Generates ASCII text files. Does not retain the layout and pictures of the original documents for document recomposition. Document files created have the extension TXT.

OCR Zones

If implemented in your program, the Image Annotation Tool Button control and the Image Edit control let end users draw OCR zones on image documents to control OCR processing.

Depending on the type of zone drawn, OCR processing either occurs or doesn't occur. Using OCR zones increases the efficiency of OCR processing.

There are two types of OCR zones:

Text — OCR Text zones define an area of an image containing text that should be processed by the OCR engine. When the OCR engine encounters a text zone, it converts the underlying image to text.

Picture — OCR Picture zones define an area of an image containing one or more pictures that should not be processed by the OCR engine. When the OCR engine encounters a picture zone, it does not convert the underlying image to text; rather, it retains the image as a graphic.

Actual OCR processing is managed via the Image OCR control.

Input Quality Types

If implemented in your program, the OCR control lets end users indicate the quality of the original documents they are going to OCR. Indicating the quality of the original documents in advance of performing OCR can improve document recognition and processing time.

Auto Detect — This setting is ideal for original documents whose pages vary in quality. Imaging automatically detects the input quality of each page. Note that this option requires additional processing time as Imaging performs its analysis.

Fax — This setting is appropriate for original documents that were faxed or scanned at a resolution of 200 dots per inch or lower.

Dot Matrix — This setting is appropriate for original documents that were printed on a dot-matrix printer. It should not be used to process image documents having a different origin.

Normal — This setting is ideal for original documents of good quality.

User-Defined Dictionaries

Also known as Custom dictionaries, user-defined dictionaries can be used to improve OCR efficiency.

If implemented in your program, end users can create, edit, and select (use) dictionaries that contain unusual terminology, such as:

- Technical or scientific terms
- Proper names
- Any other words that are not likely to be found in a standard dictionary

The OCR engine uses the contents of these user-defined dictionaries to increase the chance that such words or terminology will be recognized.

Users should therefore select a dictionary that is specifically designed for the type of image document to be processed by the OCR engine. Otherwise, recognition accuracy may actually be descreased.

Interactive Training

Interactive Training is a facility that can improve OCR performance. If implemented in your program, end users can conduct an interactive training session as the program performs OCR on an image document. Then it lets users apply the training results to future OCR processing.

When run, the Interactive Training facility highlights each questionable word in its OCR Training dialog box. The dialog box gives the user the opportunity to ignore the word, change the word, and/or add the word to a special training file as well as to the first selected user-defined dictionary.

The training file literally trains the OCR engine to perform OCR with greater efficiency and accuracy when it processes image documents similar to the one processed during the training session.

After the interactive training session is completed, the user deselects interactive training. When recognizing image documents similar to the one processed during the training session, the user selects the training file and user-defined dictionary for use so that their training data can be applied.

Original Document Language

If implemented in your program, end users can indicate the language of the original documents they are going to OCR.

When users specify a language, Imaging loads and uses a built-in system dictionary that contains up to 50,000 words that are common to the language specified. As a result, Imaging conducts OCR processing with greater accuracy.

Training Files

If implemented in your program, end users can select a training file when performing OCR processing. Training files contain accepted or corrected words that the OCR engine highlighted as questionable during one or more interactive training sessions. They improve the process of recognizing image documents similar to the one processed during the training sessions.

You should make sure that your users include a training file that is specifically designed for the type of image document they expect to OCR. Including an inappropriate training file can actually decrease recognition accuracy.

Document Recompositions

Document recomposition is a process that reconstructs OCR output documents so they retain the layout of their originals. As part of the process, the OCR engine analyzes the image and identifies text, pictures, and layout. It uses this information to then recompose the document so it looks like its original.

If implemented in your program, recomposition is available only when the output format is set to Word for Windows/RTF, WordPerfect, or HTML.

The following conditions exist:

- When the output format is set to Word for Windows/RTF or WordPerfect, the OCR engine can be set to provide full document recomposition.
- When the output format is set to HTML, the OCR engine does not provide full document recomposition. The OCR engine can be set so that original documents containing pictures are recomposed, with the pictures appearing in their original positions; however, original multi-column documents are recomposed as single-column documents.

10

Image Scan Control

This chapter describes the properties, methods, and events for the Image Scan control. In addition to the properties and methods described in this chapter, the following properties and methods, which are described in Chapter 7, apply to the Image Scan control:

- StatusCode property
- AboutBox method
- GetVersion method

In This Chapter

What the Image Scan Control Lets You Do

The Image Scan control lets you — the application developer — add scanning functions to applications that support 32-bit ActiveX controls.

Applications that you create with the Image Scan control can only work with TWAIN compliant scanners.

Your scanner requires a TWAIN data source (usually supplied by the manufacturer) and the 32-bit TWAIN DLLs provided with Imaging for Windows.

What the Image Scan Control Lets Your Users Do

The Image Scan control lets your users scan and save image documents as an integral part of your application. Depending on how you design and code your application, users can scan new single-page and multipage image documents, as well as append and insert scanned pages into existing documents.

The resulting image documents can be viewed, manipulated, annotated, printed, or sent within an e-mail package.

Prerequisites

The Scan control is invisible and can work independently of other controls. However, to display images while they are being scanned, you must link the Scan control to the Image Edit control that will display these images.

To link an Image Scan control to an Image Edit control:

1. Set the DestImageControl property of the Image Scan control to the name of the desired Image Edit control.
2. Set the ImageControl property of the Image Edit control to the same Image Edit control name.

These properties are set automatically when you draw an Image Scan control and an Image Edit control on a form.

Obsolete Properties and Methods

The following properties and methods are obsolete, and no longer supported.

- **CompressionInfo** property. Please use the methods GetPageTypeCompressionInfo (see page 749) and SetPageTypeCompressionOpts (see page 764) instead.
- **CompressionType** property. Please use the method GetPageTypeCompressionType instead (see page 751).
- **PageType** property. Please use the following methods instead:
  - GetPageTypeCompressionInfo (see page 749)
  - GetPageTypeCompressionType (see page 751)
  - GetCompressionPreference (see page 745)
  - SetPageTypeCompressionOpts (see page 764)
- **ResetScanner** (see page 762) method**.**
- **ShowScannerSetup** method. Refer to the StartScan method (see page 779), which can be used with the ShowSetupBeforeScan property (see page 742) to provide the same function.

Note: After installing the latest TWAIN DLLs, you may experience compatibility problems with some previously installed applications. This usually means that your data source is no longer current. Contact the manufacturer of your scanner for the latest data sources and compatibility information.

CompressionInfo Property

Description Sets the compression options to use when scanning.

> **Note:** This property is obsolete — the information provided here is for backward compatability. Please refer to the methods GetPageTypeCompressionInfo (see page 749) and SetPageTypeCompressionOpts (see page 764).

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**CompressionInfo**[=*value*]

Data Type Long.

Setting	Description
Note:	Settings 1 to 32 cannot be used with JPEG compression.
1	EOL - include or expect standard end of line bit sequences
2	Packed lines - data is not byte aligned
4	Prefixed EOLs - include or expect prefixed end of line bit sequences
8	Compressed bit order, left to right
16 (default)	Expanded bit order, left to right
32	Negate - reverses black and white on expansion
Note:	The following settings pertain only to JPEG compression.
64	Low resolution, high quality
128	Low resolution, medium quality
256	Low resolution, low quality
512	Medium resolution, high quality
1024	Medium resolution, medium quality
2048	Medium resolution, low quality
4096	High resolution, high quality
8192	High resolution, medium quality
16384	High resolution, low quality

Remarks Using the JPEG compression method, the higher the compression ratio, the more degraded the image quality becomes. The amount of degradation that is acceptable varies with the image and the application. Experimenting with different settings is recommended.

Resolution is independent of the compression ratio. Optimal settings will vary, based on individual requirements.

Be careful when changing these values — the compression type could be affected.

CompressionType Property

Description Sets the compression type to use when scanning.

> **Note:** This property is obsolete — the information provided here is for backward compatability. Please refer to the methods GetPageTypeCompressionType (see page 751) and SetPageTypeCompressionOpts (see page 764) instead.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

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

Data Type Integer (enumerated).

Constant	Setting	Description
NoCompression	1	No compression
CCITTGroup3_1d_Fax	2	Group 3 1D FAX
CCITTGroup3_1d_ModifiedHuffman	3 (default)	Group 3 1D Modified Huffman
PackedBits	4	PackBits
CCITTGroup4_2d_Fax	5	Group 4 2D FAX
JPEG	6	JPEG
LZW	7	LZW

Remarks The value set for this property determines which settings are available in the **CompressionInfo** property.

DestImageControl Property

Description Returns or sets the name of the destination image/edit control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DestImageControl**[=*controlname*]

Data Type String.

Remarks This property links the Image Scan control to a specified Image Edit control, displaying the image that is being scanned.

Use the name of the Image Edit control to which you want to link.

DestImageControl Example — VB

If there are multiple Image Edit controls within an application, set the DestImageControl property to specify which control will receive the scan data.

```
Private Sub cmdDestImage_Click()
    'The DestImageControl property must be set to match the value specified
    'in the ImgEdit's ImageControl property. This would scan to an ImgEdit
    'control referred to as ImgEdit2.
    frmMain.ImgScan1.DestImageControl = "ImgEdit2"
    frmMain.ImgScan1.StartScan
End Sub
```

DestImageControl Example — VC++

If there are multiple Image Edit controls within an application, set the DestImageControl property to specify which control will receive the scan data.

```
Void CNewscanDlg::OnDestimagecontrol()
{
    // The DestImageControl property must be set to match the value
    // specified in the ImgEdit's ImageControl property. This would scan to
    // an ImgEdit control referred to as ImgEdit2.
    ImgScan1.SetDestImageControl("ImgEdit2")
    ImgScan1.StartScan();
}
```

FileType Property

Description Returns or sets the type of image file that will be created.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**FileType**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
TIFF	1 (default)	TIFF
AWD_MicrosoftFax	2	AWD (Windows 95 and 98 only)
BMP_Bitmap	3	BMP

Remarks Set this value to the file type you want to create when scanning.

See Also SetPageTypeCompressionOpts method.

FileType Example — VB

This example shows how to insert a scanned page before page 1 of an existing image file.

```
Private Sub cmdInsert_Click()
    'Use the ImgAdmin control to select a file for display in the ImgEdit
    'control.
    ImgAdmin1.ShowFileDialog OpenDlg  '0
    ImgEdit1.Image = ImgAdmin1.Image
    ImgEdit1.Display
    'For insert or append, set the Scan control's Image property.
    ImgScan1.Image = ImgAdmin1.Image
    'MultiPage must be set to True in order to create files with more than
    'one page.
    ImgScan1.MultiPage = True
    ImgScan1.PageOption = InsertPages '3
    ImgScan1.Page = 1
    ImgScan1.FileType = TIFF '1
    'Scan using the Scan Page dialog box.
    ImgScan1.ShowScanPage
End Sub
```

FileType Example — VC++

This example shows how to insert a scanned page before page 1 of an existing image file.

```
void CNewscanDlg::OnInsertpage( =)
{
    // This example shows how to insert a scanned page before page 1 of an
    // existing image. Use the ImgAdmin control to select a file for display
    // in the ImgEdit control.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg  // 0
    // Check to see if cancel was pressed.
    ImgEdit1.SetImage(ImgAdmin1.GetImage());
    ImgEdit1.Display();
    // For insert or append, set the Scan control's Image property.
    ImgScan1.SetImage(ImgAdmin1.GetImage());
    // Multipage must be set to True in order to create files with more than
    // one page.
    ImgScan1.SetMultiPage(TRUE);
    ImgScan1.SetPageOption(3);  //InsertPages '3
    ImgScan1.SetPage (1);
    ImgScan1.SetFileType(1); // TIFF '1
    // Scan using the Scan Page dialog box.
    VARIANT vModal; V_VT(&vModal) = VT_BOOL;
    V_BOOL(&vModal) = TRUE;
    ImgScan1.ShowScanPage(vModal);
}
```

Image Property

Description Returns or sets the name of the object to which you are scanning.

> **Note:** This property is not the same as, or linked to, the Image properties used by the Image Edit and Thumbnail controls.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Image**[=*filename*]

Data Type String.

Remarks The object you scan to — a file or template — must be set in the **ScanTo** property.

If you point to a file template, image files are generated and incremented as you scan to them. For example, if you use a template named 'img', you must specify a path, such as c:\scan\images\img. The template 'img' will generate files named img00001.xxx, img00002.xxx, and so on.

If a valid path (see page 535) is not specified, an error is returned.

See Also DestImageControl property, ScanTo property.

Image Example — VB

These examples show how to set the Image property when you are scanning to different locations. Single slashes can be forward (/) or backward (\).

```
Private Sub cmdSetImage_Click()
    'Syntax for setting image property to a local file.
    ImgScan1.Image = "C:/scanimg/contract.tif"
    'Syntax for a 1.x Server document. Single slashes can be forward or
    'back. Note that the prefix Image:// represents the service name for a
    '1.x server.
    ImgScan1.Image = "Image://srvrname\volname:
    ➥    \CABNAME\DRAWERNAME\FOLDERNAME\YOURDOC"
    'Syntax for 1.x Server filename with a volume name mapped.
    ImgScan1.Image = "Image://srvrname/volname:/dirname/temp.tif"
End Sub
```

Image Example — VC++

These examples show how to set the Image property when you are scanning to different repositories. Single slashes can be forward (/) or backward (\).

```
void CNewscanDlg::OnImage()
{
    // Example of setting image property to a local file
    ImgScan1.SetImage ("C:\\scanimg\\contract.tif");
    // Syntax for 1.x Server document (Single slashes can be forward or
    // back)
    // Note: prefix Image:// denotes service name for 1.x server.
    ImgScan1.SetImage("Image://srvrname\volname:
    ➥    \\CABNAME\\DRAWERNAME\\FOLDERNAME\\YOUR DOC");
    // Syntax for 1.x Server filename with a volume name mapped.
    ImgScan1.SetImage ("Image://srvrname/volname:/dirname/temp.tif");
}
```

MultiPage Property

Description Determines if one or more image pages will be scanned to an image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**MultiPage**[= {*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Enables multiple image pages to be scanned to an image file.
False (default)	Enables a single image page to be scanned to an image file.

Remarks Use this property when appending or inserting pages into an image file.

This property interacts with the **PageCount** property when you

- scan to a template (see page 782 for more information).
- scan to a file (see page 782 for more information).

See Also PageCount property, PageOption property.

MultiPage Example — VB

This example shows how to insert a scanned page before page 1 of an existing image file.

```
Private Sub cmdInsert_Click()
    'Use the ImgAdmin control to select a file for display in the ImgEdit
    'control.
    ImgAdmin1.ShowFileDialog OpenDlg  '0
    ImgEdit1.Image = ImgAdmin1.Image
    ImgEdit1.Display
    ImgScan1.Image = ImgAdmin1.Image
    'MultiPage must be set to True in order to create files with more than
    'one page.
    ImgScan1.MultiPage = True
    ImgScan1.PageOption = InsertPages '3
    ImgScan1.Page = 1
    ImgScan1.FileType = TIFF '1
    'Scan using the Scan Page dialog box.
    ImgScan1.ShowScanPage
End Sub
```

MultiPage Example — VC++

This example shows how to insert a scanned page before page 1 of an existing image file.

```
void CNewscanDlg::OnInsertpage()
{
     // This example shows how to insert a scanned page before page 1 of an
     // existing image. Use the ImgAdmin control to select a file for
     // display in the ImgEdit control.
     VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
     V_I4(&vhWnd) = (long)m_hWnd;
     ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg  // 0
     // Check to see if cancel was pressed.
     ImgEdit1.SetImage(ImgAdmin1.GetImage());
     ImgEdit1.Display();
     // For insert or append, set the Scan control's Image property.
     ImgScan1.SetImage(ImgAdmin1.GetImage());
     // Multipage must be set to True in order to create files with more
     // than one page.
     ImgScan1.SetMultiPage(TRUE);
     ImgScan1.SetPageOption(3);  //InsertPages '3
     ImgScan1.SetPage (1);
     ImgScan1.SetFileType(1); // TIFF '1
     // Scan using the Scan Page dialog box.
     VARIANT vModal; V_VT(&vModal) = VT_BOOL;
     V_BOOL(&vModal) = TRUE;
     ImgScan1.ShowScanPage(vModal);
}
```

Page Property

Description Returns or sets the starting page for a scanning session.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Page**[=*pagenumber*]

Data Type Long.

Remarks Set this property to the current page number.

To append to a file, set the page number to 0 (zero). Also, the **PageOption** property must be set to 2 (Append Pages).

To insert one or more pages, set the page number to the number of the page you are inserting before. For example, to insert before page 5, enter page number 5.

See Also PageOption property.

Page Example — VB

This example shows how to insert a scanned page before page 1 of an existing image file.

```
Private Sub cmdInsert_Click()
    'Use the ImgAdmin control to select a file for display in the ImgEdit
    'control.
    ImgAdmin1.ShowFileDialog OpenDlg  '0
    ImgEdit1.Image = ImgAdmin1.Image
    ImgEdit1.Display
    'For insert or append, set the Scan control's Image property.
    ImgScan1.Image = ImgAdmin1.Image
    'MultiPage must be set to True in order to create files with more than
    'one page.
    ImgScan1.MultiPage = True
    ImgScan1.PageOption = InsertPages '3
    ImgScan1.Page = 1
    ImgScan1.FileType = TIFF '1
    'Scan using the Scan Page dialog box.
    ImgScan1.ShowScanPage
End Sub
```

Page Example — VC++

This example shows how to insert a scanned page before page 1 of an existing image file.

```
void CNewscanDlg::OnInsertpage()
{
     // This example shows how to insert a scanned page before page 1 of an
     // existing image. Use the ImgAdmin control to select a file for
     // display in the ImgEdit control.
     VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
     V_I4(&vhWnd) = (long)m_hWnd;
     ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg  // 0
     // Check to see if cancel was pressed.
     ImgEdit1.SetImage(ImgAdmin1.GetImage());
     ImgEdit1.Display();
     // For insert or append, set the Scan control's Image property.
     ImgScan1.SetImage(ImgAdmin1.GetImage());
     // Multipage must be set to True in order to create files with more
     // than one page.
     ImgScan1.SetMultiPage(TRUE);
     ImgScan1.SetPageOption(3);  //InsertPages '3
     ImgScan1.SetPage (1);
     ImgScan1.SetFileType(1); // TIFF '1
     // Scan using the Scan Page dialog box.
     VARIANT vModal; V_VT(&vModal) = VT_BOOL;
     V_BOOL(&vModal) = TRUE;
     ImgScan1.ShowScanPage(vModal);
}
```

PageCount Property

Description Returns or sets the number of pages per image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PageCount**[=*number of pages*]

Data Type Long.

Valid values are 0 (zero) to 0x7FFF (32767). The default value is 0x7FFF.

This property interacts with the **MultiPage** property when you

- scan to a template (see page 782 for more information).
- scan to a file (see page 782 for more information).

See Also Image property (Image Scan), MultiPage property.

PageCount Example — VB

This code segment shows how to scan all the pages in the scanner's automatic document feeder (ADF) to a multipage file using a file template. The example creates three image pages and names them using the prefix “img” (for example, img0001.tif, img0002.tif).

```
Private Sub cmdTemplate_Click()
    ImgScan1.ScanTo = UseFileTemplateOnly '4
    'Set the image property to a template name.
    ImgScan1.Image = "D:\image2\img"
    'MultiPage must be true in order to create files with more than one
    'page.
    ImgScan1.MultiPage = True
    'Create 3 page image files.
    ImgScan1.PageCount = 3
    'Do not show the scanner's TWAIN UI.
    ImgScan1.ShowSetupBeforeScan = False
    'Scan without using dialog box.
    ImgScan1.StartScan
End Sub
```

PageCount Example — VC++

This code segment shows how to scan all the pages in the scanner’s automatic document feeder (ADF) to a multipage file using a file template. The example creates three image pages and names them using the prefix “img” (for example, img0001.tif, img0002.tif).

```
void CNewscanDlg::OnTemplate()
{
    ImgScan1.SetScanTo (4);  // UseFileTemplateOnly // 4
    // Set the image property to a template name.
    ImgScan1.SetImage ("D:\\image2\\img");
    // Multipage must be true in order to create files with more than one
    // page.
    ImgScan1.SetMultiPage(TRUE);
    // Create 3 page image files.
    ImgScan1.SetPageCount(3);
    // Do not show the scanner’s TWAIN UI.
    ImgScan1.SetShowSetupBeforeScan(FALSE);
    // Scan without using dialog box.
    ImgScan1.StartScan();
}
```

PageOption Property

Description Indicates or sets if a scanned page will be Appended or Inserted, or if a page will be overwritten with or without a prompt at scan time.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**PageOption**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
CreateNewFile	0	Creates a new image file and adds image pages. An error occurs if the file exists. Pages can be added only if an automatic document feeder (ADF) is present.
PromptToCreateNewFile	1 (default)	Creates a new image file and adds image pages. If an image file exists, a prompt is displayed before overwriting. Pages can be added only if an automatic document feeder (ADF) is present.
AppendPages	2	Appends to an existing image file.
InsertPages	3	Inserts a page in the existing image file, before the current image page, as specified in the Page property. An error will occur if the file does not exist.
OverwritePages	4	Overwrites an image file on an existing image page. If the image page does not exist, an error occurs.
PromptToOverwritePages	5	Displays a prompt prior to overwriting an existing image page. If the image page does not exist, an error will occur.
OverwriteAllPages	6	Overwrites all pages in an image file.

Remarks The **Page** property determines the current image page in a selected file.

If you are appending or inserting pages (Settings 2 and 3), you must set the **MultiPage** property to True. Otherwise, the error message “File already exists” is returned.

If settings 0, 1, 2, or 6 are chosen, a new file is created if the file specified in the **Image** property does not exist.

See Also Image property (Image Scan), MultiPage property, Page property.

PageOption Example — VB

This example shows how to insert a scanned page before page 1 of an existing image file.

```
Private Sub cmdInsert_Click()
    'Use the ImgAdmin control to select a file for display in the ImgEdit
    'control.
    ImgAdmin1.ShowFileDialog OpenDlg  '0
    ImgEdit1.Image = ImgAdmin1.Image
    ImgEdit1.Display
    'For insert or append, set the Scan control's Image property.
    ImgScan1.Image = ImgAdmin1.Image
    'MultiPage must be set to True to create files with more than 1 page
    ImgScan1.MultiPage = True
    ImgScan1.PageOption = InsertPages '3
    ImgScan1.Page = 1
    ImgScan1.FileType = TIFF '1
    'Scan using the Scan Page dialog box.
    ImgScan1.ShowScanPage
End Sub
```

PageOption Example — VC++

This example shows how to insert a scanned page before page 1 of an existing image file.

```
void CNewscanDlg::OnInsertpage()
{
     // This example shows how to insert a scanned page before page 1 of an
     // existing image. Use the ImgAdmin control to select a file for
     // display in the ImgEdit control.
     VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
     V_I4(&vhWnd) = (long)m_hWnd;
     ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg  // 0
     // Check to see if cancel was pressed.
     ImgEdit1.SetImage(ImgAdmin1.GetImage());
     ImgEdit1.Display();
     // For insert or append, set the Scan control's Image property.
     ImgScan1.SetImage(ImgAdmin1.GetImage());
     // Multipage must be set to True to create files with more than 1 page
     ImgScan1.SetMultiPage(TRUE);
     ImgScan1.SetPageOption(3);  //InsertPages '3
     ImgScan1.SetPage (1);
     ImgScan1.SetFileType(1); // TIFF '1
     // Scan using the Scan Page dialog box.
     VARIANT vModal; V_VT(&vModal) = VT_BOOL;
     V_BOOL(&vModal) = TRUE;
     ImgScan1.ShowScanPage(vModal);
}
```

PageType Property

Description Returns or sets the image type for the pages being saved after scanning.

> **Note:** This property is obsolete, and the information provided is for backward compatability. Please use the following methods instead:
>
> - GetPageTypeCompressionInfo (see page 749)
> - GetPageTypeCompressionType (see page 751)
> - GetCompressionPreference (see page 745)
> - SetPageTypeCompressionOptions (see page 764)

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

√ Imaging for Windows 95 and 98

√ Imaging for Windows NT 4.0

Usage *object*.**PageType**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
BlackAndWhite	1 (default)	Black and white
Gray16Shades	2	Gray scale, 4-bit
Gray256Shades	3	Gray scale, 8-bit
Color16Count	4	Palletized, 4-bit (BMP and TIFF)
Color256Count	5	Palletized, 8-bit (BMP and TIFF)
TrueColor24bit	6	RGB, 24-bit
HighColor24bit	7	BGR, 24-bit

Remarks This property also determines the compression type and compression information required for saving the image file.

ScanTo Property

Description Returns or sets the destination of the image being scanned.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ScanTo**[=*value*]

Data Type Integer (enumerated).

Constant	Setting	Description
DisplayOnly	0 (default)	Display the scanned image.
DisplayAndFile	1	Display the scanned image and write it to a file.
FileOnly	2	Write the scanned image to a file.
DisplayAndUseFile Template	3	Write the scanned image to a file using a template, and display the image.
Note: DisplayAndUseFileTemplate pertains only to files on local/redirected drives, and not to 1.x server documents.		
UseFileTemplateOnly	4	Write the scanned image to a file using a template. This option pertains only to files on local/redirected drives, and to 1.x server documents.
FaxOnly	5	Fax the scanned image.

Remarks Ensure that the Image Scan control's **Image** property is set correctly.

Settings 1, 2, 3, and 4 — you cannot write a file directly to an optical disk. You must write to a magnetic drive first, and then copy the file to the optical drive.

Settings 1, 2, and 5 require a file name.

Settings 3 or 4 require that the **Image** property points to a file template. Image files are then generated and incremented as you scan to them. For example, if you use a template named 'img', you must specify a path, such as c:\scan\images\img. The template 'img' will generate files named img00001.xxx, img00002.xxx, and so on.

If a valid path (see page 535) is not provided, an error will be returned.

This property interacts with other properties, depending on the value set.

- scan to a template (see page 782 for more information).
- scan to a file (see page 782 for more information).

See Also Image property (Image Scan), MultiPage property, PageCount property.

ScanTo Example — VB

This code segment scans all pages in the ADF to a multipage file while displaying each page.

```
Private Sub cmdDisplayAndFile_Click()
    ImgScan1.ScanTo = DisplayAndFile
    'Set the image property to a file name.
    ImgScan1.Image = "D:\image2\newpages.tif"
    'Multipage must be true in order to create files with more than one
    'page.
    ImgScan1.MultiPage = True
    'Do not show the scanner's TWAIN UI.
    ImgScan1.ShowSetupBeforeScan = False
    'Scan without using dialog box.
    ImgScan1.StartScan
End Sub
```

The following two examples demonstrate how the **ScanTo** property can be used to specify the destination of the image that is being scanned. In the first example, the image is scanned to the file specified by the **Image** property. In the second example, the image is scanned to the display and to a filename template specified with the **Image** property.

Note: You should set the **Image** property after you set the **ScanTo** property.

```
'Example 1
Private Sub cmdScanToFile_Click()
    On Error GoTo ScanError
    ImgScan1.ScanTo = FileOnly      'Scan to a file.
    ImgScan1.Image = "c:\scan.tif"  'Destination file
    ImgScan1.StartScan
    Exit Sub
ScanError:
    MsgBox Err.Description, , "Scan Error"
    Exit Sub
End Sub

'Example 2
Private Sub cmdScanToDisplay_Click()
    On Error GoTo ScanError
    ImgScan1.ScanTo = DisplayAndUseFileTemplate   'Scan to display and
                                                  'template.
    ImgScan1.Image = "c:\img"                     'Template = imgxxxxx.tif
    ImgScan1.StartScan
    Exit Sub
ScanError:
    MsgBox Err.Description, , "Scan Error"
    Exit Sub
End Sub
```

ScanTo Example — VC++

This code segment scans all pages in the ADF to a multipage file while displaying each page.

```
void CNewscanDlg::OnDisplayAndFile()
{
   ImgScan1.SetScanTo (1);     // DisplayAndFile
    // Set the image property to a file name.
    ImgScan1.SetImage ("D:\\image2\\newpages.tif");
    // Multipage must be true in order to create files with more than one
    // page.
    ImgScan1.SetMultiPage (TRUE);
    // Do not show the scanner's TWAIN UI.
    ImgScan1.SetShowSetupBeforeScan (FALSE);
    // Scan without using dialog box.
    ImgScan1.StartScan();
}
```

Scroll Property

Description Sets or indicates if a displayed image will be scrolled while it is being scanned. Works only if "Memory" was set as the TWAIN transfer mode.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Scroll**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Scanned image is scrolled.
False	Scanned image is not scrolled.

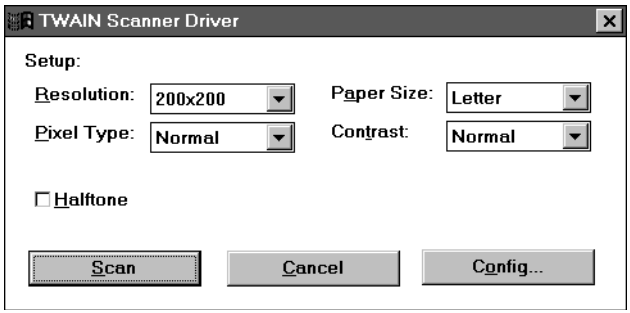
Remarks Used in conjunction with the **ScanTo** property if ScanTo is set to a value of 0, 1, or 3.

The Advanced button on the dialog box created by the **ShowScanPreferences** method presents a choice between two transfer modes — Native and Memory. Users can switch between these modes to see which gives the best performance with their scanner driver. You (the developer) cannot set this mode programmatically.

See Also ScanTo property, ShowScanPreference method.

ShowSetupBeforeScan Property

Description Displays the scanner's user interface (see the sample below) before starting a scan.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowSetupBeforeScan**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Displays the user interface before scanning.
False	Does not display user interface before scanning.

Remarks Used to set up the scanner before the first scan, and then bypass the setup dialog box for subsequent scans.

Some TWAIN scanner data sources require that a user interface dialog box be displayed before each scan, because settings from the previous scan are not saved.

ShowSetupBeforeScan Example — VB

This code segment shows how to scan all the pages in the scanner's automatic document feeder (ADF) to multipage files using a file template. The example creates three-page image files and names them using the prefix "img" (for example, img0001.tif, img0002.tif).

```
Private Sub cmdTemplate_Click()
    ImgScan1.ScanTo = UseFileTemplateOnly '4
    'Set the image property to a template name.
    ImgScan1.Image = "D:\image2\img"
```

```
    'MultiPage must be true in order to create files with more than one
    'page.
    ImgScan1.MultiPage = True
    'Create 3 page image files.
    ImgScan1.PageCount = 3
    'Do not show the scanner's TWAIN UI.
    ImgScan1.ShowSetupBeforeScan = False
    'Scan without using dialog box.
    ImgScan1.StartScan
End Sub
```

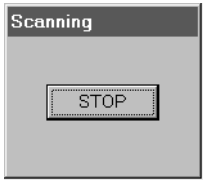
ShowSetupBeforeScan Example — VC++

This code segment shows how to scan all the pages in the scanner's automatic document feeder (ADF) to multipage files using a file template. The example creates three-page image files and names them using the prefix “img” (for example, img0001.tif, img0002.tif).

```
void CNewscanDlg::OnTemplate()
{
    ImgScan1.SetScanTo (4);  // UseFileTemplateOnly // 4
    // Set the image property to a template name.
    ImgScan1.SetImage ("D:\\image2\\img");
    // Multipage must be true in order to create files with more than one
    // page.
    ImgScan1.SetMultiPage(TRUE);
    // Create 3 page image files.
    ImgScan1.SetPageCount(3);
    // Do not show the scanner's TWAIN UI.
    ImgScan1.SetShowSetupBeforeScan(FALSE);
    // Scan without using dialog box.
    ImgScan1.StartScan();
}
```

StopScanBox Property

Sets or indicates if the Stop Scan dialog box (shown here) will be displayed.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**StopScanBox**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Displays the Stop Scan dialog box.
False (default)	Does not display the Stop Scan dialog box.

Remarks The Stop Scan dialog box enables the user to stop the scanning process.

Zoom Property

Description Returns or sets the scale at which scanned pages are displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

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

Data Type Long.

Remarks Used in conjunction with the **ScanTo** property if ScanTo is set to a value of 0, 1, or 3.

At scan time, the zoom value set is rounded to one of the following values: 6.25%, 12.5%, 25%, 50%, 100%, 200%, 400%, or 800%.

If you are scanning to an Image Edit control, set the **Zoom** property in that control to the same value that you set here.

See Also ScanTo property, Zoom property (Image Edit).

CloseScanner Method

Description Closes the scanner by unloading the driver.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**CloseScanner**

Arguments None.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition returns an informational message.

Remarks When a scanner or application error occurs, close the scanner before you access it again to avoid possible problems.

See Also OpenScanner method.

GetCompressionPreference Method

Description Returns the compression preference used by the **ShowScanPreference** method or the **SetPageTypeCompressionOpts** method.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98

Imaging for Windows 95

Imaging for Windows NT 4.0

Usage *object*.**GetCompressionPreference**(*CompPreference*)

Returns CompPreferenceConstants (Integer).

Constant	Setting	Description
BEST_DISPLAY_QUALITY	0 (default)	Best display quality
GOOD_DISPLAY_SMALL_FILE	1	Good display quality, small file size
SMALL_FILE	2	Smallest file size
SCAN_COMP_CUSTOM	3	Custom settings

Remarks The information returned by this method can be used as follows:

- To maintain the compression options already set in the registry. (The information returned is read but not interpreted.)
- To interpret compression information, and then use that information to set explicit compression options during runtime.
- Predefined compression preferences are shown below.

Best Display

Black and White

CCITT Group 3 1D

Group 3 1D Modified Huffman with reversed bit order

16 Shades of Gray

LZW (Professional Edition only)

LZW (no compression information)

256 Shades of Gray

LZW (Professional Edition only)

LZW (no compression information)

256 Colors

LZW (Professional Edition only)

LZW (no compression information)

True Color (RGB 24 bit)

Uncompressed

No compression information

16 Colors

LZW (Professional Edition only)

LZW (no compression information)

Good Display

Black and White

CCITT Group 3 1D

Group 3 1D Modified Huffman with reversed bit order

16 Shades of Gray

LZW (Professional Edition only)

LZW (no compression information)

256 Shades of Gray

JPEG

JPEG medium resolution, medium quality

256 Colors

LZW (Professional Edition only)

LZW (no compression information)

True Color (RGB 24 bit)

JPEG

JPEG medium resolution, medium quality

16 Colors

LZW (Professional Edition only)

LZW (no compression information)

Smallest File

Black and White

CCITT Group 3 1D

Group 3 1D Modified Huffman with reversed bit order

16 Shades of Gray

LZW (Professional Edition only)

LZW (no compression information)

256 Shades of Gray

JPEG

JPEG medium resolution, low quality

256 Colors

LZW (Professional Edition only)

LZW (no compression information)

True Color (RGB 24 bit)

JPEG

JPEG medium resolution, low quality

16 Colors

LZW (Professional Edition only)

LZW (no compression information)

See Also GetPageTypeCompressionInfo method, GetPageTypeCompressionType method, SetPageTypeCompressionOpts method.

GetCompressionPreference Example — VB

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
Private Sub cmdGetCompression_Click()
    'Option Base 1
    Dim lngCompInfo(6) As Long
    Dim intCompType(6) As Integer
    Dim intCompPref, intImgType As Integer
    CompPref = ImgScan1.GetCompressionPreference
    'Using intImgType to loop through 6 times (for each page type) and also
    'as the input parameter to the methods.
    For intImgType = 1 To 6
        intCompType(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionType(intImgType)
        lngCompInfo(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionInfo(intImgType)
    Next intImgType
End Sub
```

GetCompressionPreference Example — VC++

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
void CNewscanDlg::OnGetcompression()
{
    // Option Base 1
    long lCompInfo[6];
    long iCompType[6];
    int iCompPref, iImgType;
    iCompPref = ImgScan1.GetCompressionPreference();
    // Using intImgType to loop through 6 times (for each page type) and
    // also as input parameter to the methods.
    for(iImgType = 0; iImgType < 6 ; iImgType++)
        {
        iCompType[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionType(iImgType+1);
        lCompInfo[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionInfo(iImgType+1);
        }
}
```

GetPageTypeCompressionInfo Method

Description Returns the compression information used by the specified image type. Applications can request and save this information for future use.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**GetPageTypeCompressionInfo**(*ImageType*)

Arguments ImageTypeConstants (Integer).

Constant	Setting	Description
BlackAndWhite1Bit	1	Black and white
Gray4Bit	2	16 shades of gray
Gray8Bit	3	256 shades of gray
ColorPal8Bit	4	256 colors
TrueColor24bitRGB	5	True color (RGB 24 bit)
ColorPal4Bit	6	16 Colors

Returns CompInfoConstants (Integer).

Constant	Setting	Description
NoCompInfo	0	No compression information
G31DModifiedHuffman	4096	Group 3 1D Modified Huffman
G31DModifiedHuffmanRBO	0	Group 3 1D Modified Huffman with reversed bit order
G31DFax	6400	Group 3 1D Fax
G31DFaxRBO	2304	Group 3 1D Fax with reversed bit order
G42DFax	4608	Group 4 2D Fax
G42DFaxRBO	512	Group 4 2D Fax with reversed bit order
TIFFPackbitsInfo	0	TIFF Packbits (no compression information)
LZWInfo	0	LZW (no compression information)
JPEGLowLow	11610	JPEG low resolution, high quality
JPEGLowMed	7740	JPEG low resolution, medium quality

Constant	Setting	Description
JPEGLowHigh	3870	JPEG low resolution, low quality
JPEGMedLow	27994	JPEG medium resolution, high quality
JPEGMedMed	24124	JPEG medium resolution, medium quality
JPEGMedHigh	20254	JPEG medium resolution, low quality
JPEGHighLow	-21158	JPEG high resolution, high quality
JPEGHighMed	-25028	JPEG high resolution, medium quality
JPEGHighHigh	-28898	JPEG high resolution, low quality

Remarks The information returned by this method can be used as follows.

- To maintain the compression options already set in the registry. (The information returned is read but not interpreted.)
- To interpret compression information, and then use that information to set explicit compression options during runtime.

An error is returned if the ImageType is invalid.

See Also GetCompressionPreference method, GetPageTypeCompressionType method, SetPageTypeCompresssionOpts method.

GetPageTypeCompressionInfo Example — VB

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
Private Sub cmdGetCompression_Click()
    'Option Base 1
    Dim lngCompInfo(6) As Long
    Dim intCompType(6) As Integer
    Dim intCompPref, intImgType As Integer
    CompPref = ImgScan1.GetCompressionPreference
    'Using intImgType to loop through 6 times (for each page type) and also
    'as the input parameter to the methods.
    For intImgType = 1 To 6
        intCompType(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionType(intImgType)
        lngCompInfo(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionInfo(intImgType)
    Next intImgType
End Sub
```

GetPageTypeCompressionInfo Example — VC++

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
void CNewscanDlg::OnGetcompression()
{
    // Option Base 1
    long lCompInfo[6];
    long iCompType[6];
    int iCompPref, iImgType;
    iCompPref = ImgScan1.GetCompressionPreference();
    // Using intImgType to loop through 6 times (for each page type)
    // and also as the input parameter to the methods.
    for(iImgType = 0; iImgType < 6 ; iImgType++)
        {
        iCompType[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionType(iImgType+1);
        lCompInfo[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionInfo(iImgType+1);
        }
}
```

GetPageTypeCompressionType Method

Description Returns the compression information used by the specified image type. Applications can then request and save this information for future use.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**GetPageTypeCompressionType**(*ImageType*)

Arguments ImageTypeConstants (Integer).

Constant	Setting	Description
BlackAndWhite1Bit	1	Black and white
Gray4Bit	2	16 shades of gray
Gray8Bit	3	256 shades of gray
ColorPal8Bit	4	256 colors
TrueColor24bitRGB	5	True color (RGB 24 bit)

Constant	Setting	Description
ColorPal4Bit	6	16 Colors

Returns CompTypeConstants (Integer).

Constant	Setting	Description
Uncompressed	0	uncompressed
CCITTGroup31D	1	CCITT Group 3 1D
CCITTGroup42D	2	CCITT Group 4 2D
TIFFPackbits	4	TIFF Packbits
JPEGCompression	8	JPEG
LZWCompression	21	LZW

Remarks The information returned by this method can be used as follows:

- To maintain the compression options already set in the registry. (The information returned is read but not interpreted.)
- To interpret compression information, and then use that information to set explicit compression options during runtime.

An error is returned if the ImageType is invalid.

See Also GetCompressionPreference method, GetPageTypeCompressionInfo method, SetPageTypeCompressionOpts method.

GetPageTypeCompressionType Example — VB

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
Private Sub Sub cmdGetCompression_Click()
    'Option Base 1
    Dim lngCompInfo(6) As Long
    Dim intCompType(6) As Integer
    Dim intCompPref, intImgType As Integer
    CompPref = ImgScan1.GetCompressionPreference
    'Using intImgType to loop through 6 times (for each page type) and also
    'as the input parameter to the methods.
    For intImgType = 1 To 6
        intCompType(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionType(intImgType)
        lngCompInfo(intImgType) =
        ➥    ImgScan1.GetPageTypeCompressionInfo(intImgType)
    Next intImgType
End Sub
```

GetPageTypeCompressionType Example — VC++

This example demonstrates how to read and retain all current compression settings. In the event that another application makes changes to these settings at runtime, the saved values can be restored.

```
void CNewscanDlg::OnGetcompression()
{
    // Option Base 1
    long lCompInfo[6];
    long iCompType[6];
    int iCompPref, iImgType;
    iCompPref = ImgScan1.GetCompressionPreference();
    // Using intImgType to loop through 6 times (for each page type) and
    // also as input parameter to the methods.
    for(iImgType = 0; iImgType < 6 ; iImgType++)
        {
        iCompType[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionType(iImgType+1);
        lCompInfo[iImgType] =
        ➥    ImgScan1.GetPageTypeCompressionInfo(iImgType+1);
        }
}
```

GetScanCapability Method

Description Returns scanner capability information or current option settings.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**GetScanCapability**(*Capability_ID*)

Arguments The GetScanCapability method has the following parameter:

Parameter	Data Type	Description
Capability_ID	Integer	Capability information, current values of options, or Variant for an error or unknown capability.

Returns A variant containing capability values supported by the scanner.

Capability_ID/ Name	Description	Return Value	Data Type
1 Image Types Available	Colors supported	A bit-wise combination of the following values: 1 - black and white 2 - grayscale 16 4 - grayscale 256 8 - color 256 palletized 16 - true color 24-bit RGB 32 - color 16-bit palettized	Integer
2 Maximum Height	Maximum physical page height available on the flatbed.	Inches	Single
3 Maximum Height (with ADF)	Maximum physical page height supported with an Automatic Document Feeder (ADF).	Inches	Single
4 Maximum Width	Maximum physical page width	Inches	Single
5 Minimum Resolution	Minimum resolution available	Dots-per-inch	Long
6 Maximum Resolution	Maximum resolution available	Dots-per-inch	Long
7 Resolution Step	The steps between maximum and minimum resolutions	Resolution increments, or an array of values	Long, if step is available. Otherwise, an array of Long or Integer values.
8 Minimum Brightness	Minimum brightness available	Brightness value	Long
9 Maximum Brightness	Maximum brightness available	Brightness value	Long

Capability_ID/ Name	Description	Return Value	Data Type
10 Brightness step	The steps between maximum and minimum brightness values	Brightness increments, or an array of values	Long, if increments are available. Otherwise, an array of Long or Integers values.
11 Auto Brightness	Determines if an auto brightness feature is available.	True - Auto brightness supported False - Auto brightness *not* supported	Boolean
12 Minimum Contrast	Minimum contrast available	Minimum contrast value	Long
13 Maximum Contrast	Maximum contrast available	Maximum contrast value	Long
14 Contrast Step	The steps between maximum and minimum contrast values	Contrast increments, or an array of values	Long, if increments are available. Otherwise, an array of Long or Integer values.
15 Auto Contrast	Determines if an auto contrast feature is available.	True - Auto contrast supported False - Auto contrast *not* supported	Boolean
16 ADF available	Determines if an automatic document feeder (ADF) is available	True - ADF is available False - ADF is *not* available	Boolean

The following option information shows what values are currently set on the scanner.

Capability_ID/ Name	Description	Return Value	Data Type
100 Image Type	Image Type in use	One of the following values: 1 - black and white 2 - grayscale 16 4 - grayscale 256 8 - color 256 palletized 16 - true color 24-bit RGB 32 - color 16-bit palettized	Integer
101 X Resolution	X Resolution in use	A value between maximum and minimum resolution	Long
102 Brightness Mode	Reports if the Brightness Mode is set to Normal or Automatic	One of the following values: 1 - Normal 2 - Automatic	Integer
103 Brightness	If the Brightness Mode is Normal, the brightness level is returned	A brightness value (unit of measure varies with scanner type)	Long
104 Contrast Mode	Reports if the Contrast Mode is set to Normal or Automatic.	One of the following values: 1 - Normal 2 - Automatic	Integer
105 Contrast	If the Contrast Mode is Normal, the contrast level is returned	Contrast value (unit of measure varies with scanner type)	Long
106 Image Layout	Reports four values (in inches) that define the image layout - Left, Top, Height, and Width.	An array of 4 values, listed in the following order: Left start of scan offset Top start of scan offset Width of scan Height of scan	An array of Single values
107 Scan Mode	Reports if the Scan Mode is set to Flatbed or ADF (Automatic Document Feeder).	One of the following values: 1 - Flatbed 2 - ADF	Short

Capability_ID/ Name	Description	Return Value	Data Type
108 Paper in Feeder	Reports if there is paper in the ADF. The Scanner Mode must be set to 2 (ADF present) for this capability to be valid.	True - Paper is present False - Paper is not present	Boolean
109 Y Resolution	Y Resolution in use	A value between maximum and minimum resolution	Long

Remarks The scanner must be open when this call is made, or an error is returned.

If a capability is not supported, a VT_Error is returned.

See Also SetScanCapability method.

GetScanCapability Example — VB

This example uses SetScanCapability to specify the Image Type for scanning, without using the scanner's TWAIN dialog box.

```
Private Sub CmdSetImgType_Click()
    'User is given the option to select an Image Type for scanning of black
    'and white, grayscale or color. The option selected is stored in the
    'global variable ScanType.
    Const IT_BW = 1
    Const IT_GRAY8 = 4
    Const IT_RGB = 16
    Const CAP_SCAN_IMAGE_TYPE = 100
    Const CAP_SCAN_IMAGE_TYPES_SUPPORTED = 1
    Dim varScanImgType As Variant
    'Global Scantype as integer
    'Scanner must be open for Capability check.
    ImgScan1.OpenScanner
    'See what image types the scanner supports.
    varScanImgType =
    ➥    ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_TYPES_SUPPORTED)
    If VarType(varScanImgType) = vbError Then
        MsgBox "Unable to get Supported Image Types"
    End If
    'Find out if the type the user selected is supported. If so set the
    'capability to that image type.
    Select Case ScanType
    Case 1
        If varScanImgType And IT_BW Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_BW
        Else
           MsgBox "Black & White image type is not supported by your scanner"
        End If
```

```
    Case 2
        If varScanImgType And IT_GRAY8 Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_GRAY8
        Else
           MsgBox "Grayscale image type is not supported by your scanner"
        End If
    Case 3
        If varScanImgType And IT_RGB Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_RGB
        Else
           MsgBox "Color image type is not supported by your scanner"
        End If
    End Select
End Sub
```

GetScanCapability Example — VC++

This example uses SetScanCapability to specify the Image Type for scanning, without using the scanner's TWAIN dialog box.

```
void CNewscanDlg::OnCapability()
{
// User is given the option to select an Image Type for scanning of black
// and white, grayscale or color. The option selected is stored in a global
// variable ScanType.
#define IT_BW  1
#define IT_GRAY8  4
#define IT_RGB 16
#define CAP_SCAN_IMAGE_TYPE 100
#define CAP_SCAN_IMAGE_TYPES_SUPPORTED 1
    VARIANT vScanImgType;
    VARIANT vCap; V_VT(&vCap) = VT_I2;
    // Global Scantype as integer
    // Scanner must be open for Capability check.
    ImgScan1.OpenScanner();
    // See what image types the scanner supports.
    vScanImgType =
    ➥    ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_TYPES_SUPPORTED);
    if(V_VT(&vScanImgType) = VT_ERROR)
        AfxMessageBox ("Unable to get Supported Image Types");
    // Find out if the type the user selected is supported. If so, set the
    // capability to that image type.
    switch (m_iScanType)
        {
    case 1:
        if(V_I4(&vScanImgType) & IT_BW)
                {
                     V_I2(&vCap)= IT_BW;
                     ImgScan1.SetScanCapability (CAP_SCAN_IMAGE_TYPE,vCap);
                }
        else
           AfxMessageBox ("Black & White image type is not supported by
           ➥    your scanner");
                break;
```

```
        case 2:
            if(V_I4(&vScanImgType) & IT_GRAY8)
                    {
                         V_I2(&vCap)= IT_GRAY8;
                         ImgScan1.SetScanCapability (CAP_SCAN_IMAGE_TYPE,vCap);
                     }
            else
               AfxMessageBox ("Grayscale image type is not supported by your
               ➥    scanner");
                    break;
        case 3:
            if(V_I4(&vScanImgType) & IT_RGB)
                    {
                        V_I2(&vCap)= IT_RGB;
                        ImgScan1.SetScanCapability( CAP_SCAN_IMAGE_TYPE,vCap);
                    }
            else
               AfxMessageBox ("Color image type is not supported by your
               ➥    scanner");
            }
}
```

OpenScanner Method

Description Opens the scanner by loading the driver.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**OpenScanner**

Arguments None.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition returns an informational message.

Remarks When the scan operation is complete, you should call the **CloseScanner** method. This frees up the scanner so that .dlls and executable files can be unloaded.

See Also CloseScanner method.

OpenScanner Example — VB

This example uses SetScanCapability to specify the Image Type for scanning, without using the scanner's TWAIN dialog box.

```
Private Sub cmdSetImgType_Click()
    'User is given the option to select an Image Type for scanning of black
    'and white, grayscale or color. The option selected is stored in the
    'global variable ScanType.
    Const IT_BW = 1
    Const IT_GRAY8 = 4
    Const IT_RGB = 16
    Const CAP_SCAN_IMAGE_TYPE = 100
    Const CAP_SCAN_IMAGE_TYPES_SUPPORTED = 1
    Dim varScanImgType As Variant
    'Global Scantype as integer
    'Scanner must be open for Capability check.
    ImgScan1.OpenScanner
    'See what image types the scanner supports.
    varScanImgType =
    ➥    ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_TYPES_SUPPORTED)
    If VarType(varScanImgType) = vbError Then
        MsgBox "Unable to get Supported Image Types"
    End If
    'Find out if the type the user selected is supported; if so set the
    'capability to that image type.
    Select Case ScanType
    Case 1
        If varScanImgType And IT_BW Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_BW
        Else
           MsgBox "Black & White image type is not supported by your scanner"
        End If
    Case 2
        If varScanImgType And IT_GRAY8 Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_GRAY8
        Else
           MsgBox "Grayscale image type is not supported by your scanner"
        End If
    Case 3
        If varScanImgType And IT_RGB Then
           ImgScan1.SetScanCapability CAP_SCAN_IMAGE_TYPE, IT_RGB
        Else
           MsgBox "Color image type is not supported by your scanner"
        End If
    End Select
End Sub
```

OpenScanner Example — VC++

This example uses SetScanCapability to specify the Image Type for scanning, without using the scanner's TWAIN dialog box.

```
void CNewscanDlg::OnCapability()
{
// User is given the option to select an Image Type for scanning of black
// and white, grayscale or color. The option selected is stored in a global
// variable ScanType.
#define IT_BW  1
#define IT_GRAY8  4
#define IT_RGB 16
#define CAP_SCAN_IMAGE_TYPE 100
#define CAP_SCAN_IMAGE_TYPES_SUPPORTED 1
    VARIANT vScanImgType;
    VARIANT vCap; V_VT(&vCap) = VT_I2;
    // Global Scantype as integer
    // Scanner must be open for Capability check.
    ImgScan1.OpenScanner();
    // See what image types the scanner supports.
    vScanImgType =
    ➥    ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_TYPES_SUPPORTED);
    if(V_VT(&vScanImgType) = VT_ERROR) AfxMessageBox ("Unable to get
    ➥    Supported Image Types");
    // Find out if the type the user selected is supported. If so, set the
    // capability to that image type.
    switch (m_iScanType)
        {
    case 1:
        if(V_I4(&vScanImgType) & IT_BW)
                {
                     V_I2(&vCap)= IT_BW;
                     ImgScan1.SetScanCapability (CAP_SCAN_IMAGE_TYPE,vCap);
                }
        else
            AfxMessageBox ("Black & White image type is not supported by your
            ➥    scanner");
                break;
    case 2:
        if(V_I4(&vScanImgType) & IT_GRAY8)
                {
                     V_I2(&vCap)= IT_GRAY8;
                     ImgScan1.SetScanCapability (CAP_SCAN_IMAGE_TYPE,vCap);
                 }
        else
           AfxMessageBox ("Grayscale image type is not supported by your
           ➥    scanner");
                break;
    case 3:
        if(V_I4(&vScanImgType) & IT_RGB)
                {
```

```
                    V_I2(&vCap)= IT_RGB;
                    ImgScan1.SetScanCapability( CAP_SCAN_IMAGE_TYPE,vCap);
                }
        else
           AfxMessageBox ("Color image type is not supported by your
           ➥    scanner");
        }
}
```

ResetScanner Method

Description Resets the scanner hardware and software.

> **Note:** This method is obsolete, as it is not supported by the TWAIN interface. The following information is provided for older applications that used it.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ResetScanner**

Arguments None.

Returns Long.

Remarks Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition returns an informational message.

ScannerAvailable Method

Description Checks to see if any TWAIN-compatible software is available.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ScannerAvailable**

Returns Boolean.

Setting	Description
True	Support for a TWAIN-compliant scanner is available.
False	Support for a TWAIN-compliant scanner is not available.

Remarks This is done by checking the software (drivers and .dlls), not the hardware.

ScannerAvailable Example — VB

This example shows how to determine if TWAIN Scanning is available before you enable or disable scanning options within the application.

```
Private Sub Form_Load()
     If ImgScan1.ScannerAvailable Then
        mnuScanNew.Enabled = True
        mnuScanAppendPage.Enabled = True
        mnuScanInsertPage.Enabled = True
     'If ScannerAvailable returns false, disable scanner-related menu
     'options.
     Else
        mnuScanNew.Enabled = False
        mnuScanAppendPage.Enabled = False
        mnuScanInsertPage.Enabled = False
    End If
End Sub
```

ScannerAvailable Example — VC++

This example shows how to determine if TWAIN Scanning is available before you enable or disable scanning options within the application.

```
void CNewscanDlg::OnEnablescanning()
{
    if(ImgScan1.ScannerAvailable())
        {
        m_mnuScanNewEnabled = TRUE;
        m_mnuScanAppendPageEnabled = TRUE;
        m_mnuScanInsertPageEnabled = TRUE;
        }
    // If ScannerAvailable returns False, disable scanner-related menu
    // options.
    else
        {
        m_mnuScanNewEnabled = FALSE;
        m_mnuScanAppendPageEnabled = FALSE;
        m_mnuScanInsertPageEnabled = FALSE;
        }
}
```

SetPageTypeCompressionOpts Method

Description Sets all compression options, based on the input Image type.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Note: The ImageType, CompType and CompInfo parameters are ignored unless the CompPreference parameter 3 (Custom) is chosen. Each parameter, however, must have a value set.

Usage *object*.**SetPageTypeCompressionOpts** *CompPreference,ImageType, CompType,CompInfo*

Arguments CompPreferenceConstants (Integer).

Constant	Setting	Description
BestDisplay	0 (default)	Best display quality
GoodDisplay	1	Good display quality, small file size
SmallestFile	2	Smallest file size
CustomSettings	3	Custom settings

ImageTypeConstants (Integer).

Constant	Setting	Description
BlackAndWhite1Bit	1	Black and white
Gray4Bit	2	16 shades of gray
Gray8Bit	3	256 shades of gray
ColorPal8Bit	4	256 colors
TrueColor24bitRGB	5	True color (RGB 24 bit)
ColorPal4Bit	6	16 Colors

CompTypeConstants (Integer).

Constant	Setting	Description
Uncompressed	0	Uncompressed
CCITTGroup31D	1	CCITT Group 3 1D
CCITTGroup42D	2	CCITT Group 4 2D
TIFFPackbits	4	TIFF Packbits
JPEGCompression	8	JPEG
LZWCompression	21	LZW

CompInfoConstants (Integer).

Constant	Setting	Description
NoCompInfo	0	No compression information
G31DModifiedHuffman	4096	Group 3 1D Modified Huffman
G31DModifiedHuffmanRBO	0	Group 3 1D Modified Huffman with reversed bit order
G31DFax	6400	Group 3 1D Fax
G31DFaxRBO	2304	Group 3 1D Fax with reversed bit order
G42DFax	4608	Group 4 2D Fax
G42DFaxRBO	512	Group 4 2D Fax with reversed bit order
TIFFPackbitsInfo	0	TIFF Packbits (no compression information)
LZWInfo	0	LZW (no compression information)
JPEGLowLow	11610	JPEG low resolution, high quality
JPEGLowMed	7740	JPEG low resolution, medium quality
JPEGLowHigh	3870	JPEG low resolution, low quality
JPEGMedLow	27994	JPEG medium resolution, high quality
JPEGMedMed	24124	JPEG medium resolution, medium quality
JPEGMedHigh	20254	JPEG medium resolution, low quality
JPEGHighLow	-21158	JPEG high resolution, high quality
JPEGHighMed	-25028	JPEG high resolution, medium quality
JPEGHighHigh	-28898	JPEG high resolution, low quality

Remarks An error occurs when

- the Compression Preference is invalid.
- the Image Type is invalid.
- the Compression Type does not match the Image Type.
- the Compression Info does not match the Compression Type.

The relationship between Image Types, Compression Types, and Compression Info is shown below.

Black and White

Uncompressed

No compression information

CCITT Group 3 1D

Group 3 1D Modified Huffman
Group 3 1D Modified Huffman with reversed bit order
Group 3 1D Fax
Group 3 1D Fax with reversed bit order

TIFF Packbits

TIFF Packbits (no compression information)

16 Shades of Gray

Uncompressed

No compression information

LZW (Professional Edition only)

LZW (no compression information)

256 Shades of Gray

Uncompressed

No compression information

LZW (Professional Edition only)

LZW (no compression information)

JPEG

JPEG Low resolution, High quality
JPEG Low resolution, Medium quality
JPEG Low resolution, Low quality
JPEG Medium resolution, High quality
JPEG Medium resolution, Medium quality
JPEG Medium resolution, Low quality
JPEG High resolution, High quality
JPEG High resolution, Medium quality
JPEG High resolution, Low quality

256 Colors

Uncompressed

No compression information

LZW (Professional Edition only)

LZW (no compression information)

True Color (RGB 24-bit)

Uncompressed

No compression information

LZW (Professional Edition only)

LZW (no compression information)

JPEG

JPEG Low resolution, High quality
JPEG Low resolution, Medium quality
JPEG Low resolution, Low quality
JPEG Medium resolution, High quality
JPEG Medium resolution, Medium quality
JPEG Medium resolution, Low quality
JPEG High resolution, High quality
JPEG High resolution, Medium quality
JPEG High resolution, Low quality

16 Colors

Uncompressed

No compression information

LZW (Professional Edition only)

LZW (no compression information)

Returns None.

See Also FileType property, GetCompressionPreference method, GetPageTypeCompressionInfo method, GetPageTypeCompressionType method.

SetPageTypeCompressionOpts Example — VB

This example demonstrates how to set up several predefined compressions for black and white, grayscale and color image types.

```
Private Sub CmdSetCompression_Click()
    ImgScan1.SetPageTypeCompressionOpts CustomSettings, BlackAndWhite1Bit
    ➥    CCITTGroup42D, G42DFaxRBO
    ImgScan1.SetPageTypeCompressionOpts CustomSettings, Gray8Bit,
    ➥    JPEGCompression, JPEGHighHigh
    ImgScan1.SetPageTypeCompressionOpts CustomSettings, TrueColor24bitRGB,
    ➥    LZWCompression, LZWInfo
End Sub
```

SetPageTypeCompressionOpts Example — VC++

This example demonstrates how to set up several predefined compressions for black and white, grayscale and color image types.

```
void CNewscanDlg::OnSetcompression()
{
    ImgScan1.SetPageTypeCompressionOpts (3,1,2,512); //CustomSettings,
    ➥    BlackAndWhite1Bit, CCITTGroup42D, G42DFaxRBO;
    ImgScan1.SetPageTypeCompressionOpts (3,3,8,28898); //CustomSettings,
    ➥    Gray8Bit, JPEGCompression, JPEGHighHigh
    ImgScan1.SetPageTypeCompressionOpts (3,5,21,0); //CustomSettings,
    ➥    TrueColor24bitRGB, LZWCompression, LZWInfo
}
```

SetScanCapability Method

Data Type Sets various scanner options, based on values retrieved by the GetScanCapability method.

Available With

√ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetScanCapability** *Capability_ID,Capability_Value*

Arguments The SetScanCapability method has the following arguments:

Parameter	Data Type	Description
Capability_ID	Integer	One of the scanner capabilities listed in the following table.
Capability_Value	Variant	Sets the following capability values.

Capability_ID Name	Description	Capability Value	Data Type
100 Image Type	Sets the Image Type	One of the following values: 1 - black and white 2 - grayscale 16 4 - grayscale 256 8 - color 256 palletized 16 - true color 24-bit RGB 32 - color 16-bit palettized	Integer
101 X Resolution	Sets the X Resolution. Use the Minimum Resolution, Maximum Resolution, and Resolution Step Capability Ids of the GetScanCapability method to get valid values.	A value between maximum and minimum resolution.	Long
	Note: This capability also sets the Y resolution, symmetric with the X resolution. If you wish to set X and Y resolutions independently, you must first set the X resolution, and then the Y resolution.		
102 Brightness Mode	Sets Brightness Mode to Normal or Automatic. If you set this value to 1 (Normal), you must also set a Brightness value.	One of the following values: 1 - Normal 2 - Automatic	Integer
103 Brightness	Sets the brightness when Brightness mode is 1 (Normal). Use the Minimum Brightness, Maximum Brightness, and Brightness Step Capability Ids of the GetScanCapability method to obtain valid values.	A value between maximum and minimum brightness	Long

Capability_ID Name	Description	Capability Value	Data Type
104 Contrast Mode	Sets Contrast Mode to Normal or Automatic. If you set this value to 1 (Normal), you must also set a Contrast value.	One of the following values: 1 - Normal 2 - Automatic	Integer
105 Contrast	Sets the contrast when Contrast mode is 1 (Normal). Use the Minimum Contrast, Maximum Contrast, and Contrast Step Capability Ids of the GetScanCapability method to obtain valid values.	A value between maximum and minimum contrast.	Long
106 Image Layout	Sets four values (in inches) that define the image layout - Left, Top, Height, and Width. Use the Maximum Height and Maximum width capability Ids of the GetScanCapability method to obtain valid values.	An array of 4 values (inches), set in the following order: Left start of scan offset Top start of scan offset Width of scan Height of scan	An array of Single values
107 Scan Mode	Sets the Scan Mode to Flatbed or ADF (Automatic Document Feeder).	One of the following values: 1 - Flatbed 2 - ADF	Short
109 Y Resolution	Sets the Y Resolution. Use the Minimum Resolution, Maximum Resolution, and Resolution Step Capability Ids of the GetScanCapability method to get valid values.	A value between maximum and minimum resolution.	Long

Note: This capability overrides the Y value set by the X Resolution Capability ID. If you wish to set X and Y resolutions independently, you must first set the X resolution, and then the Y resolution.

Remarks Before you set a capability, you should call the **GetScanCapability** method.

The scanner must be open when this call is made, or an error is returned.

If a capability is not supported, a VT_Error is returned.

Returns None.

See Also GetScanCapability method.

SetScanCapability Example — VB

This example accepts four values from the user to specify the left start, top start, width and height of the area to be scanned in inches. Values are passed via SetScanCapability, which would scan the area specified without displaying the TWAIN UI.

```
Private Sub CmdSetLayout_Click()
    Const CAP_SCAN_IMAGE_LAYOUT = 106
    Const CAP_SCAN_IMAGE_HEIGHT = 3
    Const CAP_SCAN_IMAGE_WIDTH = 4
    Dim varScanWidth, varScanHeight As Variant
    'Open scanner for capability check.
    ImgScan1.OpenScanner
    'Determine what width and height the scanner can support.
    varScanWidth = ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_WIDTH)
    If VarType(varScanWidth) = vbError Then
        MsgBox "Unable to get maximum width"
    End If
    varScanHeight = ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_HEIGHT)
    If VarType(varScanHeight) = vbError Then
        MsgBox "Unable to get maximum height"
    End If
    'The user enters width and height values in a dialog box.
    Dim sngLayout(4) As Single
        sngLayout(0) = CSng(Text1.Text)  'left start
        sngLayout(1) = CSng(Text2.Text)  'top start
        sngLayout(2) = CSng(Text3.Text)  'width
        sngLayout(3) = CSng(Text4.Text)  'height
    'Validate values the user entered before trying to set them.
    If sngLayout(0) + sngLayout(2) > varScanWidth Then
        MsgBox "Too Wide"
        Exit Sub
    End If
    If sngLayout(1) + sngLayout(3) > varScanHeight Then
        MsgBox "Too Long"
        Exit Sub
    End If
    ImgScan1.  CAP_SCAN_IMAGE_LAYOUT, sngLayout
End Sub
```

SetScanCapability Example — VC++

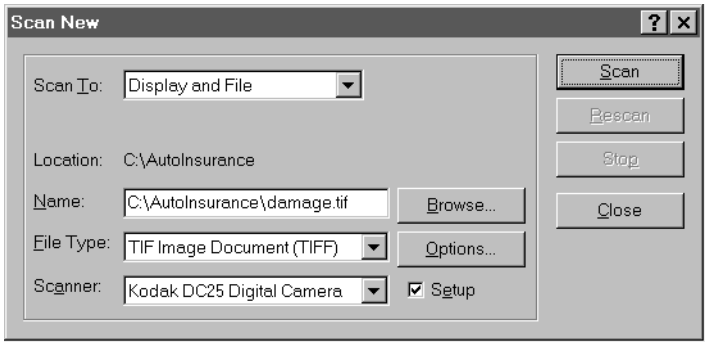
This example accepts four values from the user to specify the left start, top start, width and height of the area to be scanned in inches. Values are passed via SetScanCapability, which would scan the area specified without displaying the TWAIN UI.

```
void CNewscanDlg::OnLayout()
{
    #define CAP_SCAN_IMAGE_LAYOUT  106
    #define CAP_SCAN_IMAGE_HEIGHT  3
    #define CAP_SCAN_IMAGE_WIDTH  4
    VARIANT vScanWidth, vScanHeight;
```

```
    // Open scanner for capability check.
    ImgScan1.OpenScanner();
    // Determine what width and height the scanner can support.
    vScanWidth = ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_WIDTH);
    if(V_VT(&vScanWidth) == VT_ERROR)
        {
        AfxMessageBox ("Unable to get maximum width");
        }
    vScanHeight = ImgScan1.GetScanCapability(CAP_SCAN_IMAGE_HEIGHT);
    if(V_VT(&vScanHeight) == VT_ERROR)
        {
        AfxMessageBox("Unable to get maximum height");
        }
    // The user enters width and height values in a dialog box.
    VARIANT sngLayout[4];
    V_VT(&sngLayout[0]) = VT_R4;  // left start
    V_VT(&sngLayout[1]) = VT_R4;  // top start
    V_VT(&sngLayout[2]) = VT_R4; // width
    V_VT(&sngLayout[3]) = VT_R4; // height
    V_R4(&sngLayout[0]) = 10;  // left start
    V_R4(&sngLayout[1]) = 20;  // top start
    V_R4(&sngLayout[2]) = 400; // width
    V_R4(&sngLayout[3]) = 410; // height
    COleSafeArray  sa;
    sa.CreateOneDim(VT_R4,4,&sngLayout);
    // Validate values the user entered before trying to set them.
    if (V_R4(&sngLayout[0]) + V_R4(&sngLayout[2]) > V_I4(&vScanWidth))
        AfxMessageBox ("Too Wide");
    if (V_R4(&sngLayout[1]) + V_R4(&sngLayout[3]) > V_I4(&vScanHeight))
        AfxMessageBox("Too Long");
    ImgScan1.SetScanCapability (CAP_SCAN_IMAGE_LAYOUT, sa);
}
```

ShowScanNew Method

Displays the Scan New dialog box (shown here) from which the user can change options before scanning.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowScanNew**[={*True*|*False*}]

Arguments Boolean (optional).

Variant.

Setting	Description
True (default)	Modal dialog.
False	Modeless dialog.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error... in Visual Basic.

A busy condition or cancel returns an informational message.

Remarks Provides an interactive method to change scanning options before starting a session.

If you specify a modeless dialog box (Modal = False), the user will not be able to navigate using the keyboard.

The scanner will be opened (if it is not already) and closed again at the end of the scanning session.

See Also ShowScanPage method.

ShowScanNew Example — VB

This example illustrates one technique for scanning a new file.

```
Private Sub cmdScanNew_Click()
    'Use the Admin Control's Save As dialog box to allow the user to specify
    'a filename for the new file.
    ImgAdmin1.ShowFileDialog SaveDlg
    'Set the Scan control's Image property to the new filename.
    ImgScan1.Image = ImgAdmin1.Image
    'Scan a new file using the Scan New Dialog box.
    'Note: You do not need to set the Image property before you
    'call the dialog box. It may be set in the dialog box if desired.
    ImgScan1.ShowScanNew
End Sub
```

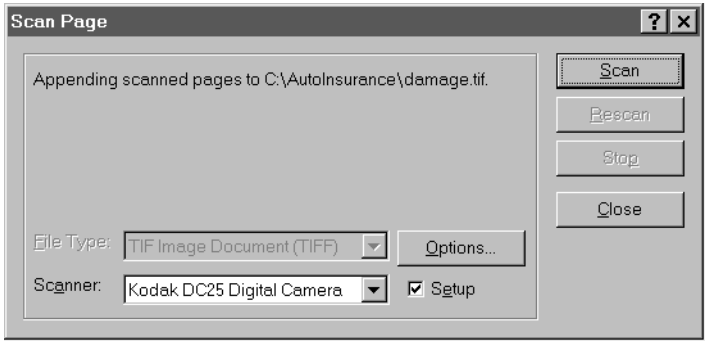
ShowScanNew Example — VC++

This example illustrates one technique for scanning a new file.

```
void CNewscanDlg::OnScannew()
{
    // Using the Admin Control Save As dialog box to allow user to specify a
    // filename for the new file.
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
    V_I4(&vhWnd) = (long)m_hWnd;
    ImgAdmin1.ShowFileDialog (1,vhWnd);      //SaveDlg
    // Set the Scan control's image property to the new filename.
    ImgScan1.SetImage (ImgAdmin1.GetImage());
    // Scan a new file using the Scan New Dialog box.
    // Note: You do not need to set the Image property before you
    // call the dialog box. It may be set in the dialog box if desired.
    VARIANT vModal; V_VT(&vModal) = VT_BOOL;
    V_BOOL(&vModal) = TRUE;
    ImgScan1.ShowScanNew(vModal);
}
void CNewscanDlg::OnFileNameRequestScanctrl1(long PageNumber)
{
    if(PageNumber > m_iPagesPerFile)
        // intPageCount will be used to sequentially increment file names
        m_iPageCount++;
    // Filename is comprised of 4 digit month/day and 4 digit sequential
    // number.
    CString  szName;
    szName.Format("D:\\image2\\%n",m_iPageCount);
    szName += ".tif";
    ImgScan1.SetImage (szName);
}
```

ShowScanPage Method

Displays the Scan Page dialog box (shown here) from which you can change options before scanning.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowScanPage**[={*True*|*False*}]

Arguments Boolean (optional).

Variant.

Setting	Description
True (default)	Modal dialog.
False	Modeless dialog.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error... in Visual Basic.

A busy condition or cancel returns an informational message.

Remarks This method is used to Insert, Append, or Overwrite pages in an existing image file while scanning.

If you specify a modeless dialog box (Modal = False), the user will not be able to navigate using the keyboard.

The scanner will be opened (if it is not already) and closed again at the end of the scanning session.

ShowScanPage Example — VB

This example shows how to insert a scanned page before page 1 of an existing image.

```
Private Sub cmdInsert_Click()
     'Use ImgAdmin to select a file for display in the ImgEdit control.
     ImgAdmin1.ShowFileDialog OpenDlg  '0
     ImgEdit1.Image = ImgAdmin1.Image
     ImgEdit1.Display
     'For insert or append, set the Scan control's Image property.
     ImgScan1.Image = ImgAdmin1.Image
     'MultiPage property must be set to True in order to create files with
     'more than one page.
     ImgScan1.MultiPage = True
     ImgScan1.PageOption = InsertPages '3
     ImgScan1.Page = 1
     ImgScan1.FileType = TIFF '1
     'Scan using the Scan Page dialog box.
     ImgScan1.ShowScanPage
End Sub
```

ShowScanPage Example — VC++

This example shows how to insert a scanned page before page 1 of an existing image file.

```
void CNewscanDlg::OnInsertpage()
{
     // This example shows how to insert a scanned page before page 1 of an
     // existing image. Use the ImgAdmin control to select a file for
     // display in the ImgEdit control.
     VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
     V_I4(&vhWnd) = (long)m_hWnd;
     ImgAdmin1.ShowFileDialog(0,vhWnd); // OpenDlg  // 0
     // Check to see if cancel was pressed.
     ImgEdit1.SetImage(ImgAdmin1.GetImage());
     ImgEdit1.Display();
     // For insert or append, set the Scan control's Image property.
     ImgScan1.SetImage(ImgAdmin1.GetImage());
     // Multipage must be set to True in order to create files with more
     // than one page.
     ImgScan1.SetMultiPage(TRUE);
     ImgScan1.SetPageOption(3);  // InsertPages '3
     ImgScan1.SetPage (1);
     ImgScan1.SetFileType(1); // TIFF '1
     // Scan using the Scan Page dialog box.
     VARIANT vModal; V_VT(&vModal) = VT_BOOL;
     V_BOOL(&vModal) = TRUE;
     ImgScan1.ShowScanPage(vModal);
}
```

ShowScanPreferences Method

Description Sets compression preferences for scanning.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowScanPreferences**

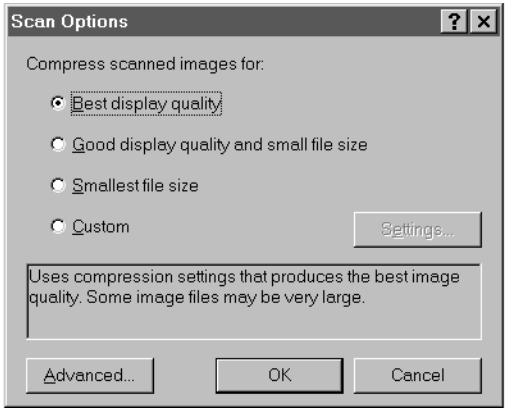
Arguments None.

Returns Long.

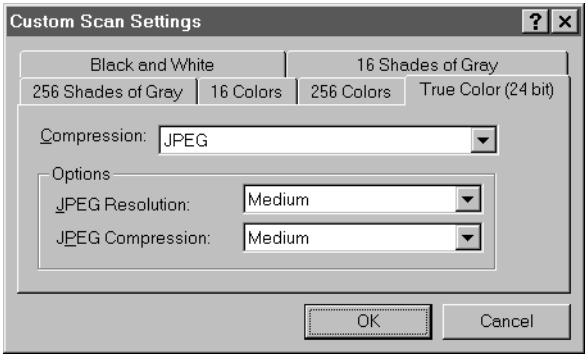
Most errors need to be intercepted by Catch () in C++ or by On Error... in Visual Basic.

A busy condition returns an informational message.

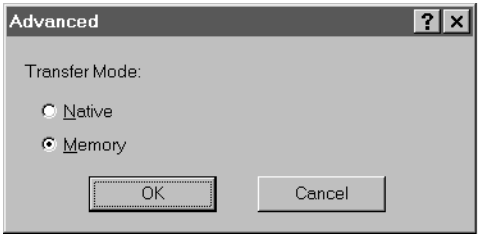
Remarks Displays a dialog box (shown here) with a list of available compression options.



You can select a custom option that displays a dialog box (shown here) with a detailed list of compression preferences.



The Advanced button on the dialog box created by the **ShowScanPreferences** method presents a choice between two transfer modes — Native and Memory. Users can switch between these modes to see which gives the best performance with their scanner driver. You (the developer) cannot set this mode programmatically.



See Also GetCompressionPreference method, GetPageTypeCompressionInfo method, GetPageTypeCompressionType method, SetPageTypeCompressionOpts method.

ShowScannerSetup Method

Description Displays the setup dialog box provided by the TWAIN scanner driver.

> **Note:** This method is no longer supported, and calling it will return an error. The recommended way to set up a scanner is:
> Use the StartScan method (see page 779) with the ShowSetupBeforeScan property (see page 742).
> Set the ShowSetupBeforeScan property to TRUE.

Available With

- Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 95
- √ Imaging for Windows NT 4.0

Usage *object*.**ShowScannerSetup**

Arguments None.

Returns Long.

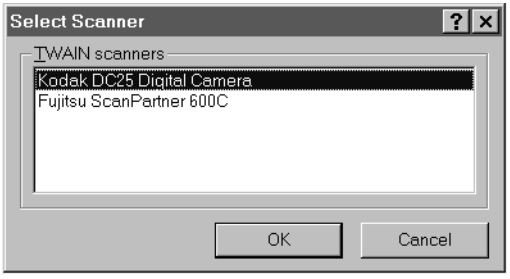
Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition returns an informational message.

Remarks The dialog box and selectable options are provided by the library twain.dll.

ShowSelectScanner Method

Description Displays a dialog box (shown here) that shows the installed TWAIN scanners. The user selects a scanner from this list.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage **Sub** *object*.**ShowSelectScanner**

Arguments None.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition or cancel returns an informational message.

See Also ScannerAvailable method.

StartScan Method

Description Starts the scanning process.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**StartScan**

Arguments None.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error… in Visual Basic.

A busy condition or cancel returns an informational message.

Remarks If the scanner is closed, it will be opened and closed again at the end of the scanning process.

See Also Image property, PageOption property, ScanTo property.

StartScan Example — VB

This code segment shows how to scan all the pages in the scanner's automatic document feeder (ADF) to a multipage file using a file template. The example creates three image pages and names them using the prefix “img” (for example, img0001.tif, img0002.tif).

```
Private Sub CmdTemplate_Click()
```

```
    ImgScan1.ScanTo = UseFileTemplateOnly '4
    'Set the image property to a template name.
    ImgScan1.Image = "C:\image2\img"
    'MultiPage property must be set to true in order to create files with
    'more than one page.
    ImgScan1.MultiPage = True
    'Create 3 page image files.
    ImgScan1.PageCount = 3
    'Do not show the scanner's TWAIN UI.
    ImgScan1.ShowSetupBeforeScan = False
    'Scan without using dialog box.
    ImgScan1.StartScan
End Sub
```

StartScan Example — VC++

This code segment shows how to scan all the pages in the scanner's automatic document feeder (ADF) to a multipage file using a file template. The example creates three image pages and names them using the prefix “img” (for example, img0001.tif, img0002.tif).

```
void CNewscanDlg::OnTemplate()
{
    ImgScan1.SetScanTo (4);  // UseFileTemplateOnly // 4
    // Set the image property to a template name.
    ImgScan1.SetImage ("D:\\image2\\img");
    // Multipage must be true in order to create files with more than one
    // page.
    ImgScan1.SetMultiPage(TRUE);
    // Create 3 page image files.
    ImgScan1.SetPageCount(3);
    // Do not show the scanner's TWAIN UI.
    ImgScan1.SetShowSetupBeforeScan(FALSE);
    // Scan without using dialog box.
    ImgScan1.StartScan();
}
```

StopScan Method

Description Stops a scanning operation in progress.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**StopScan**

Arguments None.

Returns Long.

Most errors need to be intercepted by Catch () in C++ or by On Error... in Visual Basic.

A busy condition returns an informational message.

Remarks The scan operation will stop on the page that is currently being scanned. Cleanup, including clearing paper and buffers, will be performed.

PageDone Event

Description Signals the completion of each scanned page.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage **Sub** *object*_**PageDone**

Remarks This event is triggered after a page is scanned.

Returns Long.

Passes the Number of the page just scanned.

ScanDone Event

Description Signals the end of a scanning operation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage **Sub** *object*_**ScanDone**

Remarks This event is triggered immediately after a scanning operation is completed.

ScanStarted Event

Description Signals the start of a scanning operation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage **Sub** *object*_**ScanStarted**

Remarks This event is triggered just before the first image page is transfered to the control.

ScanUIDone Event

Description Signals that the TWAIN user interface is about to come down.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage **Sub** *object*_**ScanUIDone**

Remarks Enables an application to re-activate its main window after the scan has been performed. Otherwise, if the main window is activated immediately after a call to the StartScan method, it may lose its focus and another open window will be displayed on top of it.

When you scan to a template (**ScanTo** property set to 3 or 4), the **PageCount** and **MultiPage** properties interact. The following table shows what happens when you set these properties to different values. All scenarios begin with 20 pages in the scanner's automatic document feeder.

Settings	Results
PageCount = 5 MultiPage = True	All 20 pages are scanned — 4 image files are created, and each file contains 5 pages.
PageCount = 5 MultiPage = False	5 pages are scanned — 5 image files are created, and each file contains 1 page.
PageCount = 0 MultiPage = True	All 20 pages are scanned — 1 image file is created, which contains 20 pages.

Settings	Results
PageCount = 0 MultiPage = False	All 20 pages are scanned — 20 image files are created, and each file contains 1 page.

Image Scan Extender Properties

This topic lists the Extender properties (see page 536) that are available when the Scan control is drawn on a Visual Basic form.

Name	Description
Index	Returns or sets the subscript value of a control in an array of controls.
Name	Returns the name of an object.
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
Tag	Returns or sets ancillary data.

Refer to the Visual Basic on-line help for more information about these properties.

11

Image Thumbnail Control

This chapter describes the properties, methods, and events for the Thumbnail control. In addition to the properties, methods, and events described in this chapter, the following properties, methods, and events, which are described in Chapter 7, apply to the Image Thumbnail control:

- Enabled property
- hWnd property
- StatusCode property
- AboutBox method
- GetVersion method
- ReadyStateChange event

In This Chapter

What the Image Thumbnail Control Lets You Do

The Image Thumbnail control lets you — the application developer — add thumbnail display and management functions to applications that support 32-bit ActiveX controls.

The Image Thumbnail control displays thumbnail views of the images contained in an image file in uniformly sized and spaced frames called thumbnail boxes. Thumbnails are useful for displaying each page of a multipage image file in a miniature format.

When the end user specifies an image file, thumbnail representations of images are generated and buffered for use. You can use the buffered thumbnails or choose to regenerate thumbnails to reflect subsequent changes made to the image file, such as inserted or deleted pages, or alterations made to pages (annotations).

The Thumbnail control can be configured to scroll either vertically or horizontally. When scrolled vertically, thumbnails are displayed sequentially from the top left, across the rows and down to the bottom right. When scrolled horizontally, thumbnails are displayed sequentially from the top left, down the columns and across to the bottom right. The scroll bar is displayed only when there are more thumbnails available than can fit in the control's window. A scrollbar is never displayed in the non-scrolling direction.

You can change the size of the thumbnail boxes collectively, making them smaller, to fit more into the control's window, or making them larger to see more detail. The views displayed in the thumbnail boxes maintain the same aspect ratio (height to width relationship) as the image pages they represent.

Although the Thumbnail control can only display views of the image, it can be used in conjunction with the Image Admin control to add or delete pages in the image file.

You can also choose to display captions beneath the thumbnails, and assign different fonts, colors, and styles to the captions.

What the Image Thumbnail Control Lets Your Users Do

Depending on how you design and code your application, the Image Thumbnail control lets your users view thumbnail images, format the appearance of the control, and perform other functions, such as displaying or deleting image pages represented by the thumbnail images.

Prerequisites

You must include at least one Image Admin control and Image Edit control in your application if you want to permit end users to display and manipulate image documents using the Image Thumbnail control.

If your project is going to use 1.x documents, the Image Admin control must be included, and the Admin **FileStgLoc1x** property set appropriately. You can also use the Admin control **ForceFileLinking1x** property with 1.x documents, if appropriate.

AutoSelect Property

Description Returns or sets whether the thumbnail control or the application selects thumbnails.

Available With

- √ Imaging for Windows Professional Edition (V1.0, V1.1, and V2.0)
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**AutoSelect**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True	Thumbnail control selects thumbnails.
False (default)	Calling application selects thumbnails.

AutoSelect Example — VB

This example shows how to set the highlight color for selected thumbnail boxes using the Common Color dialog.

```
Private Sub mnuHighLightColor_Click()
    On Error GoTo ErrColor
    'Thumbnails must be selected programmatically
    ImgThumbnail1.AutoSelect = False
    'Properties for the Common color dialog
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCCRGBInit Or cdlCCPreventFullOpen
    CommonDialog1.ShowColor
    ImgThumbnail1.HighlightColor = CommonDialog1.Color
    'Show first thumb highlighted
    If ImgThumbnail1.ThumbCount > 0 Then ImgThumbnail1.ThumbSelected(1) =
    ➥    True
    ImgThumbnail1.HighlightSelectedThumbs = True
    Exit Sub
ErrColor:
    MsgBox Err.Description, , "Set Colors"
    Exit Sub
End Sub
```

BackColor Property

Description Returns or sets the background color of the entire control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**BackColor**[=*color*]

Data Type Long.

Remarks Use the RGB format (see page 540). The default value is light gray — RGB (192, 192, 192).

The color specified should be different from the color set by the **HighlightColor** property so that the highlight remains visible against the background.

Do not confuse this property with the **ThumbBackColor** property, which sets the color of the background of the thumbnail boxes.

BackColor Example — VB

This example shows how to set the control color using the Common Color dialog.

```
Private Sub mnuControlColor_Click()
    On Error GoTo ErrColor
    'Properties for the Common color dialog
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCCRGBInit Or cdlCCPreventFullOpen
    CommonDialog1.ShowColor
    ImgThumbnail1.BackColor = CommonDialog1.Color
    Exit Sub
ErrColor:
    MsgBox Err.Description, , "Set Colors"
    Exit Sub
End Sub
```

BorderStyle Property

Description Returns or sets if the control is displayed with or without a border.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**BorderStyle**[=*value*]

Data Type Integer (enumerated).

Setting	Description
0	No border is displayed.
1 (default)	A single line, fixed size border is displayed.

Remarks This property can only be set at design time.

DropFailed Property

Description Sets a value to indicate the thumbnail drop operation failed.

Available With

- √ Imaging for Windows Professional Edition V2.0
- √ Imaging for Windows Professional Edition V1.1
- Imaging for Windows Professional Edition V1.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**DropFailed**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Thumbnail drop operation succeeded.
False	Thumbnail drop operation failed.

DropFailed Example — VB

This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight.

The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
        srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
        'Insert page into image file first, then Thumbnail control
        ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
        'Image file saved after insert
        ImgThumbnail1.InsertThumbs InsertBefore, 1
        'Provide information regarding dropped page.
        DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
        ➥    inserted before page " & Str(InsertBefore)
        MsgBox DropMsg, , "Drop Page"
    Next InsertCount
    Exit Sub
DropError:
    ImgThumbnail1.DropFailed = True
    MsgBox Err.Description, , "Drop Error"
End Sub
```

EnableDragDrop Property

Description Indicates the drop type allowed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Window 95
- Imaging for Windows NT 4.0

Usage *object*.**EnableDragDrop**

Data Type Long.

Constant	Setting	Description
NoDragDrop	0 (default)	Drag and drop is not allowed.
DragLeft	1	Drag from the control using the left mouse button is allowed.
DragRight	2	Drag from the control using the right mouse button is allowed.
DragLeftRight	3	Drag from the control using the left and right mouse buttons is allowed.
Drop	4	Drop into the control is allowed.
DropDragLeft	5	Drop into the control is allowed/Drag from the control using the left mouse button is allowed.
DropDragRight	6	Drop into the control is allowed/Drag from the control using the right mouse button is allowed.
DropDragLeftRight	7	Drop into the control is allowed/Drag from the control using the left and right mouse buttons is allowed.
DropFiles	8	Drop image files into the control is allowed.
DropFilesDragLeft	9	Drop image files into the control is allowed/Drag from the control using the left mouse button is allowed.

Constant	Setting	Description
DropFilesDragRight	10	Drop image files into the control is allowed/Drag from the control using the right mouse button is allowed.
DropFilesDragLeftRight	11	Drop image files into the control is allowed/Drag from the control using the left and right mouse buttons is allowed.
DropFilesDrop	12	Drop into the control is allowed/Drop image files into the control is allowed.
DropFilesDropDragLeft	13	Drop into the control is allowed/Drop image files into the control is allowed/ Drag from the control using the left mouse button is allowed.
DropFilesDropDragRight	14	Drop into the control is allowed/Drop image files into the control is allowed/ Drag from the control using the right mouse button is allowed.
DropFilesDropDragLeftRight	15	Drop into the control is allowed/Drop image files into the control is allowed/ Drag from the control using the left and right mouse buttons is allowed.

Remarks Settings can be combined in bit-wise combinations.

Drops are not accepted until the Image property has been set.

EnableDragDrop Example — VB

Description This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight. The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
```

```
            srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
            'Insert page into image file first, then Thumbnail control
            ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
            'Image file saved after insert
            ImgThumbnail1.InsertThumbs InsertBefore, 1
            'Provide information regarding dropped page
            DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
            ➥    inserted before page " & Str(InsertBefore)
            MsgBox DropMsg, , "Drop Page"
        Next InsertCount
        Exit Sub
DropError:
        ImgThumbnail1.DropFailed = True
        MsgBox Err.Description, , "Drop Error"
End Sub
```

FirstSelectedThumb Property

Description Returns the page number of the first selected thumbnail.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**FirstSelectedThumb**

Data Type Long.

Remarks Available at run-time as read-only.

The value is in the range 0 (zero) to ThumbCount, inclusive.

A value of 0 (zero) indicates there are no thumbnails selected.

FirstSelectedThumb Example — VB

This example demonstrates how to delete all the currently selected thumbnails. It deletes them one at a time, deleting the page from the source first.

```
Private Sub cmdDelete_Click()
    Dim iPagesDeleted As Integer
    Dim ThumbsToDelete, ThumbCount As Integer
    iPagesDeleted = 0
    ThumbsToDelete = ImgThumbnail1.SelectedThumbCount  'Get # of selected
                                                       'thumbnails
    TotalThumbs = ImgThumbnail1.ThumbCount
    On Error GoTo ErrDelete
        'Loop until a selected thumbnail is encountered - delete it. Must
        'reset the loop because the thumbcount has changed as well as the
```

```
        'index of the next selected thumbnail.
        For i = ImgThumbnail1.LastSelectedThumb To
        ➥    ImgThumbnail1.FirstSelectedThumb Step -1
            If ImgThumbnail1.ThumbSelected(i) = True Then
                ImgAdmin1.DeletePages i, 1  'Must delete pages from image
                                            'file first
                ImgThumbnail1.DeleteThumbs i, 1
                iPagesDeleted = iPagesDeleted + 1
            End If
        Next i
        'Display a message regarding the status of the delete attempt
        If iPagesDeleted <> ThumbsToDelete Then
            MsgBox "Error deleting pages - " &
            ➥    Str(ImgThumbnail1.StatusCode), , "Thumbnail"
        Else
            MsgBox (Str(ThumbsToDelete) & " of " & Str(TotalThumbs) & "
            ➥    were deleted.")
        End If
    Exit Sub
ErrDelete:
    MsgBox Err.Description, , "Thumbnail"
    Exit Sub
End Sub
```

HighlightColor Property

Description Returns or sets the color of the highlight that is applied to selected thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**HighlightColor**[=*color*]

Data Type Long.

Remarks The highlight appears only if the **HighlightSelectedThumbs** property is set to True.

Use the RGB format (see page 540). The default value is black RGB (0, 0, 0).

The color specified should be different from the color set by the **BackColor** property to ensure the highlight remains visible against the background.

HighlightColor Example — VB

This example shows how to set the highlight color for selected thumbnail boxes using the Common Color dialog.

```
Private Sub mnuHighLightColor_Click()
```

```
    On Error GoTo ErrColor
    'Thumbnails must be selected programmatically
    ImgThumbnail1.AutoSelect = False
    'Properties for the Common color dialog
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCCRGBInit Or cdlCCPreventFullOpen
    CommonDialog1.ShowColor
    ImgThumbnail1.HighlightColor = CommonDialog1.Color
    'Show first thumb highlighted
    If ImgThumbnail1.ThumbCount > 0 Then ImgThumbnail1.ThumbSelected(1) =
    ➥    True
    ImgThumbnail1.HighlightSelectedThumbs = True
    Exit Sub
ErrColor:
    MsgBox Err.Description, , "Set Colors"
End Sub
```

HighlightSelectedThumbs Property

Description Returns or sets whether a highlight will be displayed around selected thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**HighlightSelectedThumbs**[={*True*|*False*}]

Data Type Boolean.

Setting	Description
True (default)	Selected thumbnails highlight is displayed.
False	Selected thumbnails highlight is not displayed.

HighlightSelectedThumbs Example — VB

This example shows how to set the highlight color for selected thumbnail boxes using the Common Color dialog.

```
Private Sub mnuHighLightColor_Click()
    On Error GoTo ErrColor
    'Thumbnails must be selected programmatically
    ImgThumbnail1.AutoSelect = False
    'Properties for the Common color dialog
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCCRGBInit Or cdlCCPreventFullOpen
    CommonDialog1.ShowColor
    ImgThumbnail1.HighlightColor = CommonDialog1.Color
```

```
    'Show first thumb highlighted
    If ImgThumbnail1.ThumbCount > 0 Then ImgThumbnail1.ThumbSelected(1) =
    ➥    True
    ImgThumbnail1.HighlightSelectedThumbs = True
    Exit Sub
ErrColor:
    MsgBox Err.Description, , "Set Colors"
    Exit Sub
End Sub
```

Image Property

Description Returns or sets the filename (see page 535) of the image file/document for which thumbnails are displayed.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Image**[=*filename*]

Data Type String.

Remarks An empty string is a valid value that resets the control so that no thumbnails are displayed.

Specifying a new image or respecifying the current image forces the following changes:

- The thumbnails of the image are displayed with the thumbnail of the first page located in the upper-left corner of the control.
- **SelectedThumbCount** property value is reset to 0 (zero).
- **FirstSelectedThumb** property value is reset to 0 (zero).
- **LastSelectedThumb** property value is reset to 0 (zero).
- **ThumbCount** property value is reset to the number of pages in the specified file.
- Each element in the **ThumbSelected(PageNumber)** array is reset to False.

Image Example — VB

This example demonstrates how various images can be specifed by the **Image** property. It also shows how specific pages within the document can be designated for OCRing by the **Pages** property.

```
Private Sub SetImage_Click()
    'Showing syntax to display thumbnails of a 1.x Server document
    'Note: Prefix Image:// denotes service name for 1.x server
    ImgThumbnail1.Image = "Image://srvrname\volname:\CAB\DRAWER\FOLDER\
    ➥    YOURDOC"
```

```
    'Showing syntax to display thumbnails of a 1.x Server filename
    ImgThumbnail1.Image = "Image://srvrname/volname:/dirname/temp.tif"
    'Showing syntax to display thumbnails of a 3.x Server document
    'Note: Prefix Imagex:// denotes service name for 3.x server
    ImgThumbnail1.Image = "Imagex://Docid1234"
    'Showing syntax to display thumbnails of a an internet image via URL
    ImgThumbnail1.Image = "http://www.eastmansoftware.com/sbu/sampimg.tif"
    'Showing syntax to display thumbnails of a UNC filename
    ImgThumbnail1.Image = "\\srvrname\shared3\images\bw\thisisa.tif"
End Sub
```

LastSelectedThumb Property

Description Returns the page number of the last selected thumbnail.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**LastSelectedThumb**

Data Type Long.

Remarks Available at run-time as read-only.

The value is in the range 0 (zero) to ThumbCount, inclusive.

A value of 0 indicates there are no thumbnails selected.

LastSelectedThumb Example — VB

This example demonstrates how to delete all the currently selected thumbnails. It deletes them one at a time, deleting the page from the source first.

```
Private Sub cmdDelete_Click()
    Dim iPagesDeleted As Integer
    Dim ThumbsToDelete, ThumbCount As Integer
    iPagesDeleted = 0
    ThumbsToDelete = ImgThumbnail1.SelectedThumbCount  'Get # of selected
                                                       'thumbnails.
    TotalThumbs = ImgThumbnail1.ThumbCount
    On Error GoTo ErrDelete
        'Loop until a selected thumbnail is encountered - delete it. Must
        'reset the loop because the thumbcount has changed as well as the
        'index of the next selected thumbnail.
        For i = ImgThumbnail1.LastSelectedThumb To
        ➥    ImgThumbnail1.FirstSelectedThumb Step -1
            If ImgThumbnail1.ThumbSelected(i) = True Then
```

```
                ImgAdmin1.DeletePages i, 1  'Must delete pages from image
                                            'file first.
                ImgThumbnail1.DeleteThumbs i, 1
                iPagesDeleted = iPagesDeleted + 1
            End If
        Next i
        'Display a message regarding the status of the delete attempt.
        If iPagesDeleted <> ThumbsToDelete Then
            MsgBox "Error deleting pages - " &
            ➥    Str(ImgThumbnail1.StatusCode), , "Thumbnail"
        Else
            MsgBox (Str(ThumbsToDelete) & " of " & Str(TotalThumbs) & "
            ➥    were deleted.")
        End If
    Exit Sub
ErrDelete:
    MsgBox Err.Description, , "Thumbnail"
    Exit Sub
End Sub
```

MouseIcon Property

Description Returns or sets the cursor displayed when a custom MousePointer is used.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**MouseIcon**[=*Picture*]

Data Type Picture.

Remarks Only ICO and CUR file types are supported.

MouseIcon Example — VB

This example shows how to set a custom mouse pointer for the control.

```
Private Sub cmdChangeMouse_Click()
   Dim strMIcon As String
    strMIcon = "C:\Program Files\DevStudio\VB\samples\PGuide\Optimize
    ➥    \Litening.ico"
    ImgThumbnail1.MouseIcon = LoadPicture(strMIcon)
    ImgThumbnail1.MousePointer = wiMPCustom  'Custom pointer = 99
End Sub
```

MousePointer Property

Description Returns or sets the type of mouse cursor displayed when the cursor is positioned over the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**MousePointer**[=*integer*]

Data Type Integer (enumerated).

Constant	Setting	Description
Default	0 (Default)	For this control, the default is Arrow
Arrow	1	Arrow
Cross	2	Cross (cross-hair pointer)
I-Beam	3	I-Beam
Icon	4	Arrow
wSize	5	Size (four-pointed arrow pointing North, South, East, and West)
SizeNESW	6	Size NE SW (double arrow pointing NorthEast and SouthWest)
SizeNS	7	Size N S (double arrow pointing North and South)
SizeNWSE	8	Size NW SE (double arrow pointing NorthWest and SouthEast)
SizeWE	9	Size W E (double arrow pointing West and East)
UpArrow	10	Up Arrow
Hourglass	11	Hourglass (wait)
NoDrop	12	No Drop
ArrowandHourglass	13	Arrow and Hourglass
Arrowand QuestionMark	14	Arrow and Question mark
SizeAll	15	Size All
Custom	99	Custom pointer defined by the **MouseIcon** property

MousePointer Example — VB

This example shows how to set a custom mouse pointer for the control.

```
Private Sub cmdChangeMouse_Click()
   Dim strMIcon As String
    strMIcon = "C:\Program Files\DevStudio\VB\samples\PGuide\Optimize
    ➥    \Litening.ico"
    ImgThumbnail1.MouseIcon = LoadPicture(strMIcon)
    ImgThumbnail1.MousePointer = wiMPCustom  'Custom pointer = 99
End Sub
```

ReadyState Property

Description Returns the state of the control's properties before downloading images from a web site.

Available With

- √ Imaging for Windows Professional Edition V2.0
- Imaging for Windows Professional Edition V1.0 and V1.1
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**ReadyState**

Data Type Long.

Setting	Description
0	Control is initializing and retrieving properties.
2	Control is initialized and the download process has started.
4	Control is ready for all requests.

Remarks This property is read-only.

You cannot call any methods until a value of 4 is returned.

ReadyState Example — VB

This example demonstrates how the **ReadyState** property can be captured and generate a message box indicating the current state of the control.

When the Thumbnail control completes downloading a file, the file will also be displayed in the Imgedit control.

```
Private Sub ImgThumbnail1_ReadyStateChange(ReadyState As Long)
    sCaption = "Ready State"
    Select Case ImgThumbnail1.ReadyState
```

```
            Case 0
                MsgBox "Control is initializing and retrieving properties.", ,
                ➥     sCaption
            Case 2
                MsgBox "Control is initialized; download has started.", ,
                ➥    sCaption
                ImgThumbnail1.ClearThumbs   'Clear control during download.
            Case 4
                MsgBox "Control is ready.", , sCaption
                ImgEdit1.Image = ImgThumbnail1.Image
        End Select
End Sub
```

ScrollDirection Property

Description Returns or sets the scrolling direction.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ScrollDirection**[= {0|1}]

Data Type Integer (enumerated).

Constant	Setting	Description
Horizontal	0	Displays a horizontal scrollbar.
Vertical	1 (default)	Displays a vertical scrollbar.

Remarks The scrollbar will be displayed in the direction selected, when required, to accommodate more thumbnails.

> **Note:** Either horizontal or vertical scroll can be specified, but not both.

ScrollDirection Example — VB

This example demonstrates how to change the default size of the thumbnail width, height, and scroll direction. It will also scroll right the distance of almost a full control screen.

```
Private Sub cmdSetProps_Click()
    sRet = InputBox("Enter the desired thumbnail height", "Thumbnail
    ➥    Height", ImgThumbnail1.ThumbHeight)
    If sRet <> "" Then ImgThumbnail1.ThumbHeight = Val(sRet)
    sRet = InputBox("Enter the desired thumbnail width", "Thumbnail Width",
    ➥    ImgThumbnail1.ThumbWidth)
    If sRet <> "" Then ImgThumbnail1.ThumbWidth = Val(sRet)
```

```
    sRet = InputBox("Enter a 0 for horizontal scrolling or" & vbCrLf &
    ➥    "enter a 1 for vertical scrolling.", "Scrolling Direction")
    If sRet <> "" Then
        If Val(sRet) = 0 Then
            ImgThumbnail1.ScrollDirection = Horizontal
        Else
            ImgThumbnail1.ScrollDirection = Vertical
        End If
    End If
    ImgThumbnail1.ScrollThumbs 0, 0
End Sub
```

SelectedThumbCount Property

Returns the number of thumbnails currently selected.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**SelectedThumbCount**

Data Type Long.

Remarks Available at run-time as read-only.

The value will be in the range 0 (zero) to **ThumbCount**, inclusive.

A value of 0 (zero) means that no thumbnails are currently selected. This is the default value set each time a new image is specified by the **Image** property.

SelectedThumbCount Example — VB

This example demonstrates how to delete all the currently selected thumbnails. It deletes them one at a time, deleting the page from the source first.

```
Private Sub cmdDelete_Click()
    Dim iPagesDeleted As Integer
    Dim ThumbsToDelete, ThumbCount As Integer
    iPagesDeleted = 0
    ThumbsToDelete = ImgThumbnail1.SelectedThumbCount  'Get # of selected
                                                       'thumbnails
    TotalThumbs = ImgThumbnail1.ThumbCount
    On Error GoTo ErrDelete
        'Loop until a selected thumbnail is encountered - delete it. Must
        'reset the loop because the thumbcount has changed as well as the
        'index of the next selected thumbnail.
        For i = ImgThumbnail1.LastSelectedThumb To
        ➥    ImgThumbnail1.FirstSelectedThumb Step -1
```

```
            If ImgThumbnail1.ThumbSelected(i) = True Then
                ImgAdmin1.DeletePages i, 1  'Must delete pages from image
                                            'file first.
                ImgThumbnail1.DeleteThumbs i, 1
                iPagesDeleted = iPagesDeleted + 1
            End If
        Next i
        'Display a message regarding the status of the delete attempt.
        If iPagesDeleted <> ThumbsToDelete Then
            MsgBox "Error deleting pages - " &
            ➥    Str(ImgThumbnail1.StatusCode), , "Thumbnail"
        Else
            MsgBox (Str(ThumbsToDelete) & " of " & Str(TotalThumbs) & "
            ➥    were deleted.")
        End If
    Exit Sub
ErrDelete:
    MsgBox Err.Description, , "Thumbnail"
    Exit Sub
End Sub
```

ThumbBackColor Property

Description Returns or sets the background color of the thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbBackColor**[=*color*]

Data Type Long.

Remarks Use the RGB format (see page 540). The default value is dark gray RGB (128, 128, 128).

Do not confuse this property with the **BackColor** property, which sets the color for the control's entire background.

ThumbBackColor Example — VB

This example shows how to set the thumbnail background color using the Common Color dialog.

```
Private Sub mnuBackColor_Click()
    On Error GoTo ErrColor
    'Properties for the Common color dialog
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCCRGBInit Or cdlCCPreventFullOpen
    CommonDialog1.ShowColor
```

```
    ImgThumbnail1.ThumbBackColor = CommonDialog1.Color
    Exit Sub
ErrColor:
    MsgBox Err.Description, , "Set Colors"
    Exit Sub
End Sub
```

ThumbCaption Property

Description Returns or sets the optional custom caption used with the ThumbCaptionStyle property.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbCaption**[=*caption*]

Data Type String.

Remarks Enables you to customize the caption beneath each thumbnail.

Used when the **ThumbCaptionStyle** property specifies that captions are to be displayed.

You can use the **ThumbCaptionFont** property and **ThumbCaptionColor** property to alter the appearance of captions.

You can use a caption string constructed with the following symbols:

current thumbnail number

* number of thumbnails

$ name of current image file

? total pages in the current image file (manual mode)

For example, you can use the reserved characters # (number sign) to represent the thumbnail's page number, and * (asterisk) to represent the total number of thumbnails. For example, the captions 'Page 1 of 20', 'Page 2 of 20', and so on, can be displayed by specifying the string 'Page # of *'. (The ThumbCaptionStyle property must be set to 3 or 4 in this case.)

ThumbCaption Example — VB

This example demonstrates how a caption might be set. In this case, a custom caption along with a graphic, indicating that a page contains annotations, is applied to the thumbnails. The following characters are mapped to the properties: # indicates current thumbnail, ? indicates the total number of thumbnails, and $ indicates the file name.

```
Private Sub cmdCaption_Click()
```

```
    On Error GoTo Err_Handle
    ImgThumbnail1.ThumbCaptionStyle = CaptionWithAnno 'Display a custom
                                                      'caption
    ImgThumbnail1.ThumbCaption = "Pg #/? in $"        'Ex. Pg 1/4 in test.tif
    ImgThumbnail1.Refresh
    Exit Sub
Err_Handle:
        MsgBox Err.Description, , "Caption error"
        Exit Sub
End Sub
```

ThumbCaptionColor Property

Description Returns or sets the color of caption labels displayed beneath thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbCaptionColor**[=*color*]

Data Type Long.

Remarks Use the RGB format (see page 540). The default value is black RGB (0, 0, 0).

You can use the **ThumbCaptionFont** property and **ThumbCaptionStyle** property to alter the appearance of captions.

ThumbCaptionColor Example — VB

This example shows how the Common Font dialog can be used to set the font characteristics for the thumbnail caption.

```
Private Sub cmdSetCaptionFont_Click()
    On Error GoTo Err_Handle
    'Display the Font dialog box.
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects 'Enable all fonts and
                                                    'effects.
    CommonDialog1.ShowFont
    Set cptFont = ImgThumbnail1.ThumbCaptionFont
    'Font characteristics returned from Font dialog.
    With cptFont
        .Name = CommonDialog1.FontName
        .Size = CommonDialog1.FontSize
        .Bold = CommonDialog1.FontBold
        .Italic = CommonDialog1.FontItalic
        .Underline = CommonDialog1.FontUnderline
        .Strikethrough = CommonDialog1.FontStrikethru
```

```
    End With
    ImgThumbnail1.ThumbCaptionFont = cptFont
    'Caption color is also set in Font dialog.
    ImgThumbnail1.ThumbCaptionColor = CommonDialog1.Color
    Exit Sub
Err_Handle:
    MsgBox Err.Description, , "Error"
    Exit Sub
End Sub
```

ThumbCaptionFont Property

Description Returns or sets the font used to display captions.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbCaptionFont**[=*font*]

Data Type FONT.

Remarks The default font is MS Sans Serif, regular weight, 12 point; no italic, underline, or strikethrough.

You can use the **ThumbCaptionColor** property and **ThumbCaptionStyle** property to alter the appearance of captions.

ThumbCaptionFont Example — VB

This example shows how the Common Font dialog can be used to set the font characteristics for the thumbnail caption.

```
Private Sub cmdSetCaptionFont_Click()
    On Error GoTo Err_Handle
    'Display the Font dialog box.
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects 'Enable all fonts and
                                                    'effects
    CommonDialog1.ShowFont
    Set cptFont = ImgThumbnail1.ThumbCaptionFont
    'Font characteristics returned from Font dialog.
    With cptFont
        .Name = CommonDialog1.FontName
        .Size = CommonDialog1.FontSize
        .Bold = CommonDialog1.FontBold
        .Italic = CommonDialog1.FontItalic
        .Underline = CommonDialog1.FontUnderline
        .Strikethrough = CommonDialog1.FontStrikethru
```

```
    End With
    ImgThumbnail1.ThumbCaptionFont = cptFont
    'Caption color is also set in Font dialog.
    ImgThumbnail1.ThumbCaptionColor = CommonDialog1.Color
    Exit Sub
Err_Handle:
    MsgBox Err.Description, , "Error"
    Exit Sub
End Sub
```

ThumbCaptionStyle Property

Description Returns or sets the type of caption that appears beneath each thumbnail. You can also choose not to display captions.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbCaptionStyle**[=*integer*]

Data Type Integer (enumerated).

Constant	Setting	Description
NoCaption	0	No captions beneath the thumbnails.
SimpleNoAnno	1 (default)	Page number, left justified, beneath each thumbnail.
SimpleWithAnno	2	Same as Setting 1, but also includes a graphic before the caption for those thumbnail pages whose image file pages have annotations present.
CaptionNoAnno	3	The string set by the property **ThumbCaption** is displayed, left justified, beneath the thumbnail.
CaptionWithAnno	4	Same as Setting 3, but also includes a graphic before the caption for those thumbnail pages whose image file pages have annotations present.

Remarks You can use the **ThumbCaptionColor** and **ThumbCaptionFont** properties to alter the appearance of captions.

Note: Any captions displayed are clipped to the width of the thumbnail box.

ThumbCaptionStyle Example — VB

This example demonstrates how a caption might be set. In this case, a custom caption along with a graphic, indicating that a page contains annotations, is applied to the thumbnails. The following characters are mapped to the properties: # indicates current thumbnail, ? indicates the total number of thumbnails, and $ indicates the file name.

```
Private Sub cmdCaption_Click()
    On Error GoTo Err_Handle
    ImgThumbnail1.ThumbCaptionStyle = CaptionWithAnno 'Display a custom
                                                      'caption
    ImgThumbnail1.ThumbCaption = "Pg #/? in $"        'Ex. Pg 1/4 in test.tif
    ImgThumbnail1.Refresh
    Exit Sub
Err_Handle:
        MsgBox Err.Description, , "Caption error"
        Exit Sub
End Sub
```

ThumbCount Property

Description Returns the total number of pages (thumbnails) present in the current image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbCount**

Data Type Long.

Remarks Available at run-time as read-only.

ThumbCount is recalculated each time a valid image is specified, or when either of the methods **InsertThumbs** or **DeleteThumbs** is executed.

ThumbCount Example — VB

This example demonstrates how thumbnails might be displayed. If the thumbnail count is less than 5, the first thumbnail is displayed at the top of the control.

If the count is between 5 and 20, the 5th thumbnail is placed in the center row of the control. If the count is greater than 20, the last thumbnail is displayed in the last row of the control.

```
Private Sub cmdDisplayThumbs_Click()
    ImgThumbnail1.ScrollDirection = Vertical  'Set vertical scrolling.
    Select Case ImgThumbnail1.ThumbCount
        Case 1 To 4
            ImgThumbnail1.DisplayThumbs 'Default parameters are
                                        'ThumbNumber=1 and Option = 0.
        Case 5 To 20
            ImgThumbnail1.DisplayThumbs 5, 1
        Case Is > 20
            ImgThumbnail1.DisplayThumbs ImgThumbnail1.ThumbCount, 2
    End Select
    ImgThumbnail1.Refresh
End Sub
```

ThumbDropNames Property

Description Returns a list of filenames dropped on the Thumbnail control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**ThumbDropNames**[=*item*]

Data Type String array.

Remarks Drag and drop with the Thumbnail control is limited to thumbnail to thumbnail operations, and file drops from the Explorer. Drops onto the Desktop or imaging window in the Imaging for Windows 95 application are not supported.

This property is read-only at run-time, and available only in conjunction with the **ThumbDrop** event.

ThumbDropNames Example — VB

This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight. The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0.
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
        srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
        'Insert page into image file first, then Thumbnail control.
        ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
        'Image file saved after insert.
        ImgThumbnail1.InsertThumbs InsertBefore, 1
        'Provide information regarding dropped page.
        DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
        ➥    inserted before page " & Str(InsertBefore)
        MsgBox DropMsg, , "Drop Page"
    Next InsertCount
    Exit Sub
DropError:
    ImgThumbnail1.DropFailed = True
    MsgBox Err.Description, , "Drop Error"
End Sub
```

ThumbDropPages Property

Description Returns a list of pages for the files dropped on the Thumbnail control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage *object*.**ThumbPagesNames**[=*item*]

Data Type String array.

Remarks Valid indices for the array are in the range 0 (zero) to DropCount, where DropCount is the value passed in the ThumbDrop event.

This property is read-only at run-time, and available only in conjunction with the **ThumbDrop** event.

ThumbDropPages Example — VB

This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight. The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0.
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
        srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
        'Insert page into image file first, then Thumbnail control.
        ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
        'Image file saved after insert.
        ImgThumbnail1.InsertThumbs InsertBefore, 1
        'Provide information regarding dropped page.
        DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
        ➥    inserted before page " & Str(InsertBefore)
        MsgBox DropMsg, , "Drop Page"
    Next InsertCount
    Exit Sub
DropError:
    ImgThumbnail1.DropFailed = True
    MsgBox Err.Description, , "Drop Error"
End Sub
```

ThumbHeight Property

Description Returns or sets the height of thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbHeight**[=*value*]

Data Type Long.

Remarks Used with the **ThumbWidth** property to specify the size of the displayed thumbnail boxes.

Specify a height between 50 and 500 (pixels).

The default value of 110, used with the ThumbWidth property default value of 85, produces a close representation of a standard letter-sized sheet.

If the thumbnail box size is altered, new thumbnail representations must be generated for all pages of the specified image file. This is equivalent to calling the **ClearThumbs** method or respecifying the **Image** property.

ThumbHeight Example — VB

This example demonstrates how to change the default size of the thumbnail width, height, and scroll direction. It will also scroll right the distance of almost a full control screen.

```
Private Sub cmdSetProps_Click()
    sRet = InputBox("Enter the desired thumbnail height",  "Thumbnail
    ➥    Height", ImgThumbnail1.ThumbHeight)
    If sRet <> "" Then ImgThumbnail1.ThumbHeight = Val(sRet)
    sRet = InputBox("Enter the desired thumbnail width", "Thumbnail Width",
    ➥    ImgThumbnail1.ThumbWidth)
    If sRet <> "" Then ImgThumbnail1.ThumbWidth = Val(sRet)
    sRet = InputBox("Enter a 0 for horizontal scrolling or" & vbCrLf &
    ➥    "enter a 1 for vertical scrolling.", "Scrolling Direction")
    If sRet <> "" Then
        If Val(sRet) = 0 Then
            ImgThumbnail1.ScrollDirection = Horizontal
        Else
            ImgThumbnail1.ScrollDirection = Vertical
        End If
    End If
    ImgThumbnail1.ScrollThumbs 0, 0
End Sub
```

ThumbSelected Property

Description Returns or sets the selection status of the specified thumbnail.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbSelected**(*PageNumber*)[={*True*|*False*}]

Data Type PageNumber - Long.

ThumbSelected - Boolean.

Setting	Description
True	The thumbnail is selected.
False (default for each item in the array)	The thumbnail is not selected.

Remarks Valid PageNumber values must be between 1 and the current value of the **ThumbCount** property (inclusive).

If the **HighlightSelectedThumbs** property is set to True, selected thumbnails will display a border of the color set in the **HighlightColor** property.

This property is run-time only.

ThumbSelected Example — VB

This example demonstrates how to delete all the currently selected thumbnails. It deletes them one at a time, deleting the page from the source first.

```
Private Sub cmdDelete_Click()
    Dim iPagesDeleted As Integer
    Dim ThumbsToDelete, ThumbCount As Integer
    iPagesDeleted = 0
    ThumbsToDelete = ImgThumbnail1.SelectedThumbCount  'Get # of selected
                                                       'thumbnails.
    TotalThumbs = ImgThumbnail1.ThumbCount
    On Error GoTo ErrDelete
        'Loop until a selected thumbnail is encountered - delete it. Must
        'reset the loop because the thumbcount has changed as well as the
        'index of the next selected thumbnail.
        For i = ImgThumbnail1.LastSelectedThumb To
        ➥    ImgThumbnail1.FirstSelectedThumb Step -1
            If ImgThumbnail1.ThumbSelected(i) = True Then
                ImgAdmin1.DeletePages i, 1  'Must delete pages from image
                                            'file first
                ImgThumbnail1.DeleteThumbs i, 1
```

```
                iPagesDeleted = iPagesDeleted + 1
            End If
        Next i
        'Display a message regarding the status of the delete attempt.
        If iPagesDeleted <> ThumbsToDelete Then
            MsgBox "Error deleting pages - " &
            ➥    Str(ImgThumbnail1.StatusCode), , "Thumbnail"
        Else
            MsgBox (Str(ThumbsToDelete) & " of " & Str(TotalThumbs) & "
            ➥    were deleted.")
        End If
    Exit Sub
ErrDelete:
    MsgBox Err.Description, , "Thumbnail"
    Exit Sub
End Sub
```

ThumbWidth Property

Description Returns or sets the width of the thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ThumbWidth**[=*value*]

Data Type Long.

Used with the ThumbHeight property to specify the size of the displayed thumbnail boxes.

Specify a width between 50 and 500 (pixels).

The default value of 85, used with the **ThumbHeight** default value of 110, produces a close representation of a standard letter sized sheet.

If the thumbnail box size is altered, new thumbnail representations must be generated for all pages of the specified image file. This is equivalent to calling the **ClearThumbs** method or respecifying the **Image** property.

ThumbWidth Example — VB

This example demonstrates how to change the default size of the thumbnail width, height, and scroll direction. It will also scroll right the distance of almost a full control screen.

```
Private Sub cmdSetProps_Click()
    sRet = InputBox("Enter the desired thumbnail height",
    ➥    "Thumbnail Height", ImgThumbnail1.ThumbHeight)
    If sRet <> "" Then ImgThumbnail1.ThumbHeight = Val(sRet)
```

```
    sRet = InputBox("Enter the desired thumbnail width", "Thumbnail Width",
    ➥     ImgThumbnail1.ThumbWidth)
    If sRet <> "" Then ImgThumbnail1.ThumbWidth = Val(sRet)
    sRet = InputBox("Enter a 0 for horizontal scrolling or" & vbCrLf &
    ➥     "enter a 1 for vertical scrolling.", "Scrolling Direction")
    If sRet <> "" Then
        If Val(sRet) = 0 Then
            ImgThumbnail1.ScrollDirection = Horizontal
        Else
            ImgThumbnail1.ScrollDirection = Vertical
        End If
    End If
    ImgThumbnail1.ScrollThumbs 0, 0
End Sub
```

ClearThumbs Method

Description Clears the buffered thumbnail representations of one or all pages.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ClearThumbs** *PageNumber*

Arguments The ClearThumbs method has the following parameter:

Parameter	Data Type	Description
PageNumber	Long	Optional. If a value is not specified, a default value of 0 (zero) is assumed. The value 0 clears all pages. The page number that corresponds to the page whose thumbnail will be cleared.

Remarks Thumbnails that have been cleared will be regenerated either when needed for display, or by calling the **GenerateThumb** method.

DeleteThumbs Method

Description Deletes the thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DeleteThumbs** *DeleteAt,DeleteCount*

Arguments The DeleteThumbs method has the following parameters:

Parameter	Data Type	Description
DeleteAt	Long	The number of the first page (thumbnail) to delete. Valid values are in the range 1 to ThumbCount.
DeleteCount	Long	Optional. The number of pages to be deleted. If a value is not specified, a default value of 1 is used.

The pages must be removed from the image file itself prior to calling this method. This can be done by first calling the Admin control's **DeletePages** method.

DeleteThumbs preserves the integrity of the control's internal mapping of displayed thumbnails and the pages of the specified image file. If pages are deleted from the displayed image, and this method is not called, the thumbnails displayed will not reflect the change.

This method refreshes the control's display.

DeleteThumbs Example — VB

This example demonstrates how to delete all the currently selected thumbnails. It deletes them one at a time, deleting the page from the source first.

```
Private Sub cmdDelete_Click()
    Dim iPagesDeleted As Integer
    Dim ThumbsToDelete, ThumbCount As Integer
    iPagesDeleted = 0
    ThumbsToDelete = ImgThumbnail1.SelectedThumbCount  'Get # of selected
                                                       'thumbnails.
    TotalThumbs = ImgThumbnail1.ThumbCount
    On Error GoTo ErrDelete
        'Loop until a selected thumbnail is encountered - delete it. Must
        'reset the loop because the thumbcount has changed as well as the
        'index of the next selected thumbnail.
        For i = ImgThumbnail1.LastSelectedThumb To
        ➥    ImgThumbnail1.FirstSelectedThumb Step -1
            If ImgThumbnail1.ThumbSelected(i) = True Then
                ImgAdmin1.DeletePages i, 1  'Must delete pages from image
                                            'file first
```

```
                ImgThumbnail1.DeleteThumbs i, 1
                iPagesDeleted = iPagesDeleted + 1
            End If
        Next i
        'Display a message regarding the status of the delete attempt.
        If iPagesDeleted <> ThumbsToDelete Then
           MsgBox "Error deleting pages - " & Str(ImgThumbnail1.StatusCode), , "Thumbnail"
        Else
            MsgBox (Str(ThumbsToDelete) & " of " & Str(TotalThumbs) & "
            ➥    were deleted.")
        End If
    Exit Sub
ErrDelete:
    MsgBox Err.Description, , "Thumbnail"
    Exit Sub
End Sub
```

DeselectAllThumbs Method

Description Deselects all thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DeselectAllThumbs**

Remarks This method is equivalent to setting each item in the **ThumbSelected()** property array to False.

DeselectAllThumbs Example — VB

This example demonstrates how a menu item can be used to toggle the **SelectAllThumbs** and **DeselectAllThumbs** methods. When the menu item gets checked, all thumbnails are selected. They are deselected when the menu item becomes unchecked.

```
Private Sub mnuSelect_Click()
    If mnuSelect.Checked = True Then
        ImgThumbnail1.DeselectAllThumbs
        mnuSelect.Checked = False
    Else
        ImgThumbnail1.SelectAllThumbs
        mnuSelect.Checked = True
    End If
End Sub
```

DisplayThumbs Method

Description Displays thumbnails using specified options.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**DisplayThumbs** *ThumbNumber,Option*

Arguments The DisplayThumbs method has the following parameters:

Parameter	Data Type	Description
ThumbNumber	Long	Optional. The number of the Thumbnail whose position will be set. If a value is not specified, a default value (1) is used, and the Option parameter cannot be specified.
Option	Integer	Optional. Sets the position of the specified Thumbnail within the control. If no value is specified, a default value of 0 (zero) is used. Allowable values are: 0 Thumbnail box is displayed in the top row (for vertical scrolling), in the leftmost column (for horizontal scrolling) 1 Thumbnail box is displayed in the middle row 2 Thumbnail box is displayed in the bottom row (for vertical scrolling) in the rightmost column (for horizontal scrolling)

Remarks This method refreshes the control's display.

It is not necessary to call this method after setting the **Image** property to initiate a display of the specified image file's thumbnails.

DisplayThumbs Example — VB

This example demonstrates how thumbnails might be displayed. If the thumbnail count is less than 5, the first thumbnail is displayed at the top of the control.

If the count is between 5 and 20, the 5th thumbnail is placed in the center row of the control. If the count is greater than 20, the last thumbnail is displayed in the last row of the control.

```
Private Sub cmdDisplayThumbs_Click()
    ImgThumbnail1.ScrollDirection = Vertical  'Set vertical scrolling
    Select Case ImgThumbnail1.ThumbCount
```

```
            Case 1 To 4
                ImgThumbnail1.DisplayThumbs 'Default parameters are
                                            'ThumbNumber=1 and Option = 0.
            Case 5 To 20
                ImgThumbnail1.DisplayThumbs 5, 1
            Case Is > 20
                ImgThumbnail1.DisplayThumbs ImgThumbnail1.ThumbCount, 2
        End Select
        ImgThumbnail1.Refresh
End Sub
```

GenerateThumb Method

Description Forces the generation or regeneration of the specified image's thumbnail representation.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GenerateThumb** *Option* [,*PageNumber*]

Arguments The GenerateThumb method has the following parameters:

Parameter	Data Type	Description
Option	Integer	Indicates when the Thumbnail of a specified page should be generated. Allowable values are: 0 Generates a thumbnail if one has not already been obtained. Not specifying a PageNumber, or specifying a PageNumber of 0 causes all pages that have not been obtained to be generated. 1 Causes the thumbnail to be generated immediately. If the thumbnail was previously generated, a new one will be obtained. Not specifying a PageNumber, or specifying a PageNumber of 0 causes all pages that have not been obtained to be generated.

Parameter	Data Type	Description
		2 Suppresses the regeneration of thumbnails while the Image property is being set. This is useful for creating a temporary file for the current image and setting the Image property to the new file name without regenerating all of the thumbnails.
		Note: Once the Image property has been updated, reset this value to the previous setting so that thumbnails are updated correctly in future operations.
PageNumber	Long	Optional. The number of the page for which a Thumbnail is generated. If a value is not specified, a default value of 0 (indicates ALL pages are to be generated) is used.

Remarks This method should be called to reflect any changes made to the image file after a previous thumbnail was generated.

It is not always necessary to obtain new thumbnails (Option =1). Thumbnail images that have been generated during the current session remain buffered, even though they may not be displayed.

Use caution when using a PageNumber of 0. Resulting operations can be lengthy.

GenerateThumb Example — VB

This example demonstrates how to regenerate a thumbnail after the source page has been modified. In this case, the page is rotated left.

```
Private Sub cmdGenerateThumb_Click()
    Dim iPage As Integer
    On Error GoTo Gen_Error
    iPage = 1
    ImgEdit1.Image = ImgThumbnail1.Image
    ImgEdit1.Display
    'Rotate page 1 left 90 degrees.
    ImgEdit1.Page = iPage
    ImgEdit1.RotateLeft
    ImgThumbnail1.GenerateThumb 1, iPage 'Regenerate corresponding thumb.
    Exit Sub
Gen_Error:
    MsgBox (Err.Description)
End Sub
```

GetManualThumbFilename Method

Description Returns the image filename from the thumbnail definition.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**GetManualThumbFilename** *ThumbNumber*

Arguments The GetManualThumbFilename method has the following parameter:

Parameter	Data Type	Description
ThumbNumber	Long	Used to specify the index in an array of thumbnails. Each thumbnail defined is associated with a filename. The value must be less than or equal to the ThumbCount parameter passed in a call to the **SetManualMode** method.

Remarks Calling **SetManualMode** initiates an interactive mode, enabling the programmer to identify each thumbnail in the array by using the ThumbNumber parameter.

Call this method to get the name of the image file for each thumbnail used in manual mode.

Setting the **Image** property returns to the standard mode of operation.

GetManualThumbFilename Example — VB

This example demonstrates how to display the source page of a selected thumbnail added to the Thumbnail control while manual mode is active.

```
Private Sub ImgThumbnail1_Click(ByVal ThumbNumber As Long)
    On Error GoTo Click_Error
    'Get the source and page # of the selected thumb (0-based index).
    iSrcPage = ImgThumbnail1.GetManualThumbPage(ThumbNumber - 1)
    strSrcFile = ImgThumbnail1.GetManualThumbFilename(ThumbNumber - 1)
    'Display the source of the specified manual thumbnail in ImgEdit
    'control.
    ImgEdit1.Image = strSrcFile
    ImgEdit1.Page = iSrcPage
    ImgEdit1.Display
    Exit Sub
Click_Error:
    Exit Sub
End Sub
```

GetManualThumbPage Method

Description Returns the page within the image file from the thumbnail definitions.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**GetManualThumbPage**(*ThumbNumber*)

Arguments The GetManualThumbPage method has the following parameter:

Parameter	Data Type	Description
ThumbNumber	Long	Used to specify the index in an array of thumbnails. The value must be less than or equal to the ThumbCount parameter passed in a call to the **SetManualMode** method.

Remarks Calling **SetManualMode** initiates an interactive mode, enabling the programmer to identify each thumbnail in the array by using the ThumbNumber parameter.

Call this method to get the name of the page within an image file for each thumbnail used in manual mode.

Setting the **Image** property returns to the standard mode of operation.

GetManualThumbPage Example — VB

This example demonstrates how to display the source page of a selected thumbnail added to the Thumbnail control while manual mode is active.

```
Private Sub ImgThumbnail1_Click(ByVal ThumbNumber As Long)
    On Error GoTo Click_Error
    'Get the source and page # of the selected thumb (0-based index).
    iSrcPage = ImgThumbnail1.GetManualThumbPage(ThumbNumber - 1)
    strSrcFile = ImgThumbnail1.GetManualThumbFilename(ThumbNumber - 1)
    'Display the source of the specified manual thumbnail in ImgEdit
    'control.
    ImgEdit1.Image = strSrcFile
    ImgEdit1.Page = iSrcPage
    ImgEdit1.Display
    Exit Sub
Click_Error:
    Exit Sub
End Sub
```

GetMaximumSize Method

Description Determines the largest non-scrolling control dimension that can be used to display a specified number of thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetMaximumSize**(*ThumbCount,ScrollBar*)

Arguments The GetMaximumSize method has the following parameters:

Parameter	Data Type	Description
ThumbCount	Long	The number of thumbnail boxes desired in the non-scrolling direction.
ScrollBar	Boolean	Indicates whether or not the size value returned allows space for a scroll bar. True — Allows space for a scroll bar. False — Does not allow space for a scroll bar.

Returns Long.

The maximum control size, in pixels, needed to accommodate the number of thumbnail boxes specified by the ThumbCount parameter.

Remarks **GetMaximumSize** uses the current values of the **ScrollDirection**, **ThumbHeight**, and **ThumbWidth** properties to calculate this value.

GetMaximumSize and **GetMinimumSize** should be used together to determine a range of control sizes that allow for a display that fits the required number of thumbnails in the non-scrolling dimension.

A control size larger than this dimension in the non-scrolling direction will contain space for more than the specified number of thumbnails.

Once a control size is determined (based on the results of GetMaximumSize and GetMinimumSize), a call to the **GetScrollDirectionSize** method can be used to calculate the size required in the scrolling direction to display a specified number of thumbnails.

GetMaximumSize Example — VB

This example demonstrates how a call to the **GetScrollDirectionSize** method can be used to calculate the size needed to display 4 thumbnails in the scrolling direction while displaying 2 thumbnails in the non-scrolling direction.

First, the user is allowed to set the thumbnail width and height. The non-scrolling dimension is then derived from the average of the **GetMimimumSize** and **GetMaximumSize** methods. In this particular example, the scrolling direction is horizontal.

```
Private Sub cmdControlSize_Click()
    Dim lMinSize, lMaxSize, lVSize, lHSize As Long
    Dim bScroll As Boolean
    Dim iNonScrollThumbCount As Long
    bScroll = True
    iNonScrollThumbCount = 2        'Number of thumbnails desired in
                                    'non-scrolling direction
    ImgThumbnail1.ScrollDirection = Horizontal
    ImgThumbnail1.UISetThumbSize    'User sets thumb size interactively.
    'Determine min and max control size taking a scrollbar into account.
    lMinSize = ImgThumbnail1.GetMinimumSize(iNonScrollThumbCount, bScroll)
    lMaxSize = ImgThumbnail1.GetMaximumSize(iNonScrollThumbCount, bScroll)
    lVSize = (lMinSize + lMaxSize) / 2  'Average of the min & max
                                        'nonscrolling size
    lHSize = ImgThumbnail1.GetScrollDirectionSize(4, iNonScrollThumbCount,
    ➥    lVSize, True)
    'Prior to resizing the Thumbnail control, it is necessary to ensure  the
    'unit of measure for the form and Thumbnail control are the same. The
    'thumbnail methods return units in pixels - so the scale mode for the
    'form must be set to pixels.
    Form1.ScaleMode = vbPixel
    ImgThumbnail1.Width = lHSize
    ImgThumbnail1.Height = lVSize
End Sub
```

GetMinimumSize Method

Description Determines the smallest non-scrolling control dimension that can be used to display a specified number of thumbnail boxes.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**GetMinimumSize**(*ThumbCount,ScrollBar*)

Arguments The GetMinimumSize method has the following parameters:

Parameter	Data Type	Description
ThumbCount	Long	The number of thumbnail boxes desired in the non-scrolling direction.
ScrollBar	Boolean	Indicates whether or not the size value returned allows space for a scroll bar. True — Allows space for a scroll bar. False — Does not allow space for a scroll bar.

Returns Long.

The minimum control size needed to accommodate the number of thumbnail boxes specified by the ThumbCount parameter.

Remarks **GetMinimumSize** uses the current values of the **ScrollDirection**, **ThumbHeight**, and **ThumbWidth** properties to calculate this value.

GetMaximumSize and GetMinimumSize should be used together to determine a range of control sizes that allow for a display that fits the required number of thumbnails in the non-scrolling dimension.

A control size smaller than this dimension in the non-scrolling direction will contain space for fewer than the specified number of thumbnails.

Once a control size is determined (based on results of GetMaximumSize and GetMinimumSize), a call to the **GetScrollDirectionSize** method can be used to calculate the size required in the scrolling direction to display a specified number of thumbnails.

GetMinimumSize Example — VB

This example demonstrates how a call to the **GetScrollDirectionSize** method can be used to calculate the size needed to display 4 thumbnails in the scrolling direction while displaying 2 thumbnails in the non-scrolling direction.

First, the user is allowed to set the thumbnail width and height. The non-scrolling dimension is then derived from the average of the **GetMimimumSize** and **GetMaximumSize** methods. In this particular example, the scrolling direction is horizontal.

```
Private Sub cmdControlSize_Click()
    Dim lMinSize, lMaxSize, lVSize, lHSize As Long
    Dim bScroll As Boolean
    Dim iNonScrollThumbCount As Long
    bScroll = True
    iNonScrollThumbCount = 2        'Number of thumbnails desired in
                                    'non-scrolling direction.
    ImgThumbnail1.ScrollDirection = Horizontal
    ImgThumbnail1.UISetThumbSize    'User sets thumb size interactively.
```

```
    'Determine min and max control size taking a scrollbar into account
    lMinSize = ImgThumbnail1.GetMinimumSize(iNonScrollThumbCount, bScroll)
    lMaxSize = ImgThumbnail1.GetMaximumSize(iNonScrollThumbCount, bScroll)
    lVSize = (lMinSize + lMaxSize) / 2  'Average of the min & max
    ➥    nonscrolling size
    lHSize = ImgThumbnail1.GetScrollDirectionSize(4, iNonScrollThumbCount,
    ➥    lVSize, True)
    'Prior to resizing the Thumbnail control, it is necessary to ensure the
    'unit of measure for the form and Thumbnail control are the same. The
    'thumbnail methods return units in pixels - so the scale mode for the
    'form must be set to pixels.
    Form1.ScaleMode = vbPixel
    ImgThumbnail1.Width = lHSize
    ImgThumbnail1.Height = lVSize
End Sub
```

GetScrollDirectionSize Method

Description Determines what the control size has to be, in the scrolling dimension, to display a specified number of thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage

```
object.GetScrollDirectionSize(ScrollDirectionThumbCount,
                              NonScrollDirectionThumbCount,
                              NonScrollDirectionSize,ScrollBar)
```

Arguments The GetScrollDirectionSize method has the following parameters:

Parameter	Data Type	Description
ScrollDirectionThumbCount	Long	The number of thumbnail boxes desired in the scrolling direction.
NonScrollDirectionThumbCount	Long	The number of thumbnail boxes desired in the non-scrolling direction.
NonScrollDirectionSize	Long	The size of the control's area, in pixels, in the non-scrolling dimension for which a corresponding scroll direction size is desired.

Parameter	Data Type	Description
ScrollBar	Boolean	Indicates whether or not the size specified for NonScrollDirectionSize allows space for a scroll bar. True — Allows space for a scroll bar. False — Does not allow space for a scroll bar.

Returns Long.

Returns the control size, in pixels, needed to accommodate the number of thumbnail boxes specified by the parameter ScrollDirectionThumbCount in the scrolling direction.

Remarks If NonScrollDirectionSize is determined by first calling the **GetMaximumSize** and **GetMinimumSize** methods, **GetScrollDirectionSize** can calculate the size and scrolling direction required to display a specified number of thumbnails.

The value of the argument NonScrollDirectionThumbCount must be the same value passed as the argument ThumbCount to the GetMaximumSize and GetMinimumSize methods.

The value of the argument ScrollBar must be the same in the GetMaximumSize, GetMinimumSize, and GetScrollDirectionSize methods.

GetScrollDirectionSize Example — VB

This example demonstrates how a call to the **GetScrollDirectionSize** method can be used to calculate the size needed to display 4 thumbnails in the scrolling direction while displaying 2 thumbnails in the non-scrolling direction.

First, the user is allowed to set the thumbnail width and height. The non-scrolling dimension is then derived from the average of the **GetMimimumSize** and **GetMaximumSize** methods. In this particular example, the scrolling direction is horizontal.

```
Private Sub cmdControlSize_Click()
    Dim lMinSize, lMaxSize, lVSize, lHSize As Long
    Dim bScroll As Boolean
    Dim iNonScrollThumbCount As Long
    bScroll = True
    iNonScrollThumbCount = 2        'Number of thumbnails desired in
                                    'non-scrolling direction.
    ImgThumbnail1.ScrollDirection = Horizontal
    ImgThumbnail1.UISetThumbSize    'User sets thumb size interactively
    'Determine min and max control size taking a scrollbar into account.
    lMinSize = ImgThumbnail1.GetMinimumSize(iNonScrollThumbCount, bScroll)
    lMaxSize = ImgThumbnail1.GetMaximumSize(iNonScrollThumbCount, bScroll)
    lVSize = (lMinSize + lMaxSize) / 2  'Average of the min & max
                                        'nonscrolling size
```

```
    lHSize = ImgThumbnail1.GetScrollDirectionSize(4, iNonScrollThumbCount,
    ➥    lVSize, True)
    'Prior to resizing the Thumbnail control, it is necessary to ensure the
    'unit of measure for the form and Thumbnail control are the same. The
    'thumbnail methods return units in pixels - so the scale mode for the
    'form must be set to pixels.
    Form1.ScaleMode = vbPixel
    ImgThumbnail1.Width = lHSize
    ImgThumbnail1.Height = lVSize
End Sub
```

InsertThumbs Method

Description Informs the Thumbnail control that pages have been inserted into the currently displayed image file.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**InsertThumbs** *InsertBeforeThumb,InsertCount*

Arguments The InsertThumbs method has the following parameters:

Parameter	Data Type	Description
InsertBeforeThumb	Long	Optional. The page number of the thumbnail before which the new thumbnails will be inserted. A value of 1 means the thumbnails will be inserted before the first page. A value of **ThumbCount**+1 means the thumbnails will be inserted after the last page. If InsertBeforeThumb is not specified, the default value of **ThumbCount**+1 is used, and the InsertCount parameter cannot be set.
InsertCount	Long	Optional. The number of pages inserted. If a value is not specified, the default value of 1 is used.

Remarks The pages must be inserted into the image file itself prior to calling this method. This can be done by first calling the Admin control's **Insert** method.

InsertThumbs preserves the integrity of the control's internal mapping of displayed thumbnails and the pages of the specified image file. If pages are inserted into the displayed image file and this method is not called, the thumbnails displayed will not reflect the change.

This method refreshes the control's display.

InsertThumbs Example — VB

This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight. The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0.
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
        srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
        'Insert page into image file first, then Thumbnail control.
        ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
        'Image file saved after insert.
        ImgThumbnail1.InsertThumbs InsertBefore, 1
        'Provide information regarding dropped page.
        DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
        ➥    inserted before page " & Str(InsertBefore)
        MsgBox DropMsg, , "Drop Page"
    Next InsertCount
    Exit Sub
DropError:
    ImgThumbnail1.DropFailed = True
    MsgBox Err.Description, , "Drop Error"
End Sub
```

Refresh Method

Description Repaints the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**Refresh**

Arguments None.

Remarks If the control has an active window, the refresh occurs synchronously. Otherwise, the control is refreshed asynchronously.

This method refreshes the control's display.

Refresh Example — VB

This example demonstrates how a caption might be set. In this case, a custom caption along with a graphic, indicating that a page contains annotations, is applied to the thumbnails. The following characters are mapped to the properties: # indicates current thumbnail, ? indicates the total number of thumbnails, and $ indicates the file name.

```
Private Sub cmdCaption_Click()
    On Error GoTo Err_Handle
    ImgThumbnail1.ThumbCaptionStyle = CaptionWithAnno 'Display a custom
                                                      'caption.
    ImgThumbnail1.ThumbCaption = "Pg #/? in $"        'Ex. Pg 1/4 in test.tif
    ImgThumbnail1.Refresh
    Exit Sub
Err_Handle:
        MsgBox Err.Description, , "Caption error"
        Exit Sub
End Sub
```

SaveAs Method

Description Enables the programmer to control when thumbnail representations of the pages in the specified object (local/redirected drive, 1.x, or 3.x) are saved to a specified file format.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SaveAs**(*Filename,SaveOptions,FirstThumb,LastThumb*)

Arguments The SaveAs method has the following parameters:

Parameter	Data Type	Description
Filename	String	Specifies the name of the file that will receive the saved thumbnails.

Parameter	Data Type	Description
SaveOptions	Integer	Optional. Specifies the format of the file that will receive the saved thumbnails. If no format is specified, TIFF will be used. Valid values are: 1 – Contact Sheet image in multipage TIFF format. Thumbnails will be organized by using the current size, on multiple pages based on the range of thumbnails. 2 – Contact Sheet image in BMP format with a single page of thumbnails. 3 – TIFF file with 1 thumbnail per page. 4 – BMP file with 1 thumbnail. Additive options are: +256 – Compressed TIFF contact sheets using LZW compression. +512 – "Burn in" thumbnail image stamps, backgrounds, and captions into the TIFF contact sheets. +1024 – Generate thumbnail images with annotations displayed. +2048 – Create hyperlinks between the thumbnail image stamp annotations on the contact sheet and the source file. This option cannot be ORed with the value of +512 ("burn in" annotations). +4096 – Save the currently selected thumbnails. The current image file must be the source image file for the call to the **SaveAs** method. These additive options can be ORed with the main options 1, 2, 3, or 4, as follows: 0x0100 – LZW compression (options 1 and 3 only). 0x0200 – Burn in annotations. 0x0400 – Generate thumbnails including annotations. 0x0800 – Hyperlink image stamps to original image (options 1 and 3 only). 0x1000 – Save only sthe selected thumbs.
FirstThumb	Long	Optional. Indicates the one (1) relative thumbnail that corresponds to the page of the thumbnail to be generated and saved. (1 relative means the first page that is specified by passing FirstThumb=1). The default value for this parameter is 1.

Parameter	Data Type	Description
LastThumb	Long	Optional. Indicates the one (1) relative thumbnail that corresponds to the page of the thumbnail to be generated and saved. If this parameter is not specified, the default value of **ThumbCount** property is used for TIFF output, and FirstThumb for BMP output.

Remarks Use the **SetSaveAsDimensions** method to generate thumbnails of a different size.

In all cases, the aspect ratio of the thumbnail is kept, and the generated thumbnails are centered within the specified dimensions.

If you are saving to a 1.x document, you must include the Admin control in the project and set the **FileStgLoc1x**. The Admin **ForceFileLinking1x** property can be used if required, for 1.x document operations.

SaveAs Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

ScrollThumbs Method

Description Specifies a direction and amount for scrolling.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**ScrollThumbs**(*Direction,Amount*)

Arguments The ScrollThumbs method has the following parameters:

Parameter	Data Type	Description
Direction	Integer	Specifies the direction of scroll. **0** (zero) — scrolls the control towards the last thumbnail. **1** — scrolls the control towards the first thumbnail.
Amount	Integer	Specifies the amount of scroll desired. **0** (zero) — scrolls in the requested direction by almost a full control screenful. A small amount of original screen area is left to serve as a reference. The amount scrolled is equivalent to the amount scrolled when you click in the scroll bar (between the up/down arrows and the scroll box). **1** — scrolls the control in the requested direction by a small amount. The amount scrolled is equivalent to the amount scrolled when you click the scroll bar's up or down arrow box once.

Returns Boolean.

The return value indicates if the control was scrolled.

Setting	Description
True	The scroll was performed.
False	The scroll was not performed (for example, if the scroll was at either end of its scrolling limit and could not scroll any further).

ScrollThumbs Example — VB

This example demonstrates how to change the default size of the thumbnail width, height, and scroll direction. It will also scroll right the distance of almost a full control screen.

```
Private Sub cmdSetProps_Click()
    sRet = InputBox("Enter the desired thumbnail height", "Thumbnail
    ➥    Height", ImgThumbnail1.ThumbHeight)
    If sRet <> "" Then ImgThumbnail1.ThumbHeight = Val(sRet)
    sRet = InputBox("Enter the desired thumbnail width", "Thumbnail Width",
    ➥    ImgThumbnail1.ThumbWidth)
    If sRet <> "" Then ImgThumbnail1.ThumbWidth = Val(sRet)
    sRet = InputBox("Enter a 0 for horizontal scrolling or" & vbCrLf &
    ➥    "enter a 1 for vertical scrolling.", "Scrolling Direction")
    If sRet <> "" Then
        If Val(sRet) = 0 Then
            ImgThumbnail1.ScrollDirection = Horizontal
        Else
            ImgThumbnail1.ScrollDirection = Vertical
        End If
    End If
    ImgThumbnail1.ScrollThumbs 0, 0
End Sub
```

SelectAllThumbs Method

Description Selects all thumbnails.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage *object*.**SelectAllThumbs**

Remarks This method is equivalent to setting each item in the **ThumbSelected()** property array to True.

SelectAllThumbs Example — VB

This example demonstrates how a menu item can be used to toggle the **SelectAllThumbs** and **DeselectAllThumbs** methods. When the menu item gets checked, all thumbnails are selected. They are deselected when the menu item becomes unchecked.

```
Private Sub mnuSelect_Click()
    If mnuSelect.Checked = True Then
        ImgThumbnail1.DeselectAllThumbs
        mnuSelect.Checked = False
```

```
    Else
        ImgThumbnail1.SelectAllThumbs
        mnuSelect.Checked = True
    End If
End Sub
```

SetManualMode Method

Description Sets an interactive mode in which the programmer specifies what is displayed in each thumbnail.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetManualMode** *ThumbCount*

Arguments The SetManualMode method has the following parameters:

Parameter	Data Type	Description
ThumbCount	Long	Specifies the number of thumbnails to be manually added to the control.

Remarks Call this method before you call the **SetManualThumb** method to initialize the control for the number of thumbnails to be entered.

Set the **Image** property to return to the standard mode of operation.

SetManualMode Example — VB

This example shows how thumbnail pages can be programmatically added to the Thumbnail control. It also demonstrates how the source file and page can be retrieved.

```
Private Sub cmdManual_Click()
    Dim strThumbFile, strThumbMsg As String
    Dim iSrcPage As Integer
    'Add 2 thumbnails to the control via interactive mode.
    ImgThumbnail1.SetManualMode 2
    ImgThumbnail1.SetManualThumb 1, "C:\windows\waves.bmp", 1
    ImgThumbnail1.SetManualThumb 2, "c:\windows\tiles.bmp", 1
End Sub
```

SetManualThumb Method

Description Sets an interactive mode in which the programmer adds thumbnails to an array using the ThumbNumber parameter.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetManualThumb** *ThumbNumber,FileName,Page*

Arguments The SetManualThumb method has the following parameter:

Parameter	Data Type	Description
ThumbNumber	Long	Used to specify the index in an array of thumbnails. Each thumbnail defined is associated with a filename. The value must be less than or equal to the ThumbCount parameter passed in a call to the **SetManualMode** method.
FileName	String	Specifies the name of an image file from which the specified thumbnail will be generated.
Page	Long	Specifies which page in the image file (specified in FileName) will be used to generate the thumbnail.

Remarks Call this method to initialize each thumbnail to be used in the manual mode of the control.

This method does not verify the existence of a file, or of the pages within a file. The method is validated once the thumbnail is actually displayed or saved.

Set the **Image** property to return to the standard mode of operation.

SetManualThumb Example — VB

This example shows how thumbnail pages can be programmatically added to the Thumbnail control. It also demonstrates how the source file and page can be retrieved.

```
Private Sub cmdManual_Click()
    Dim strThumbFile, strThumbMsg As String
    Dim iSrcPage As Integer
    'Add 2 thumbnails to the control via interactive mode.
    ImgThumbnail1.SetManualMode 2
    ImgThumbnail1.SetManualThumb 1, "C:\windows\waves.bmp", 1
    ImgThumbnail1.SetManualThumb 2, "c:\windows\tiles.bmp", 1
End Sub
```

SetSaveAsBackColor Method

Description Sets the background color of thumbnails saved on a contact sheet.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsBackColor** *Color*

Remarks The default value is light gray RGB (192, 192, 192). Use the RGB format (see page 540).

This method must be called before the **SaveAs** method.

SetSaveAsBackColor Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

SetSaveAsCaption Method

Description Sets the caption font of thumbnails saved on a contact sheet.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsCaption** *Style,Caption*

Arguments The SetSaveAsCaption method has the following parameters:

Parameter	Data Type	Description
Style	Integer	Specifies the type of caption to use when saving thumbnails to a contact sheet. 0 – No caption. 1 – Simple caption using thumbnail number. A caption string is not required. 2 – Reserved. 3 – Use a caption string constructed with the following symbols: # – current thumbnail number * – number of thumbnails $ – name of current image file ? – total pages in the current image file (manual mode) 4 – Reserved.
Caption	String	Optional. Specifies the text to appear in the caption.

Remarks This method must be called before the **SaveAs** method.

SetSaveAsCaption Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
```

```
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

SetSaveAsCaptionFont Method

Description Sets the caption font of thumbnails saved on a contact sheet.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsCaptionFont** *FontName,PointSize,Color,Options*

Arguments The SetSaveAsCaptionFont method has the following parameters:

Parameter	Data Type	Description
FontName	Long	Specifies the name of the font.
PointSize	Integer	Specifies the point size of the font.
Color	VT_COLOR (OLE_COLOR)	Specifies text color.
Options	Integer	Specifies font attributes: 0 – no options 1 – bold 2 – italic 4 – strikethrough 8 – underline

Remarks If you do not specify a value for a parameter, the following defaults are applied:

FontName = MS Sans Serif
PointSize = 8
Color = black
Options = 0

This method must be called before the **SaveAs** method.

SetSaveAsCaptionFont Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

SetSaveAsDimensions Method

Description Sets the height and width of thumbnails to be saved.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsDimensions** *ThumbWidth,ThumbHeight*

Arguments The SetSaveAsDimensions method has the following parameters:

Parameter	Data Type	Description
ThumbWidth	Long	Specifies the width of the thumbnail, in pixels. Values must be in the range of 50 - 500, inclusive. A value of 0 (zero) indicates the current thumbnail width will be used.
ThumbHeight	Long	Specifies the height of the thumbnail, in pixels. Values must be in the range of 50 - 500, inclusive. A value of 0 (zero) indicates the current thumbnail height will be used.

Remarks This method must be called before the **SaveAs** method.

SetSaveAsDimensions Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

SetSaveAsOrientation Method

Description Sets the orientation (horizontal or vertical) of thumbnails to be saved on a contact sheet.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsOrientation** *Direction*

Arguments The SetSaveAsOrientation method has the following parameter:

Parameter	Data Type	Description
Direction	Integer	Specifies thumbnail orientation on the contact sheet: 0 – Horizontal orientation. Thumbnails are positioned and numbered in rows, from left to right. Rows are arranged from top to bottom. 1 – Vertical orientation. Thumbnails are positioned and numbered in columns, from top to bottom. Columns are arranged from left to right.

Remarks This method must be called before the **SaveAs** method.

SetSaveAsOrientation Example — VB

This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF
    ImgThumbnail1.SetSaveAsThumbType 6.
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
```

```
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

SetSaveAsThumbType Method

Description Sets the image type of thumbnails to generated and saved by the **SaveAs** method.

Available With

√ Imaging for Windows Professional Edition V2.0

Imaging for Windows Professional Edition V1.0 and V1.1

Imaging for Windows 95 and 98

Imaging for Windows NT 4.0

Usage *object*.**SetSaveAsThumbType** *ImageType*

Arguments The SetSaveAsThumbType method has the following parameter:

Parameter	Data Type	Description
ImageType	Integer	Specifies image type of saved thumbnail: 0 – none (BMP and TIFF) (default) 1 – black & white (BMP and TIFF) 2 – 4-bit gray (TIFF) 3 – 8-bit gray (TIFF) 4 – 4-bit palette (Not available) 5 – 8-bit palette (BMP and TIFF) 6 – 24-bit RGB (BMP and TIFF) 7 – 24-bit RGB (Not available)

Remarks If this method is not called, the default value of 0 (zero) is used.

If ImageType=0, *and* annotations are being rendered, *and* there are annotations on the image page, then the thumbnail image type will be set to 6 (24-bit RGB).

SetSaveAsThumbType Example — VB

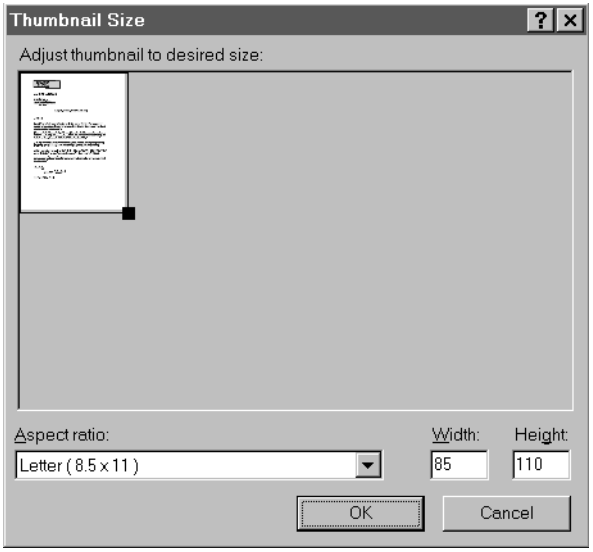
This example demonstrates how a contact sheet can be created using various methods to set the image type, caption attributes, backcolor, orientation, dimension, and file type.

```
Private Sub cmdContactSheet_Click()
    Dim iRet, ContactOption As Long
    On Error GoTo Set_Error
```

```
    'Set orientation, backcolor and dimensions.
    ImgThumbnail1.SetSaveAsBackColor vbRed
    ImgThumbnail1.SetSaveAsOrientation Vertical
    ImgThumbnail1.SetSaveAsDimensions 200, 200
    'Set caption and its properties.
    ImgThumbnail1.SetSaveAsCaption CaptionWithAnno, "Page # of *"
    ImgThumbnail1.SetSaveAsCaptionFont "Courier", 14, vbGreen, 0
    'Save the contact sheet as an RGB24 TIF.
    ImgThumbnail1.SetSaveAsThumbType 6
    ContactOption = esiTIFFContactSheet     'Set contact sheet to multipage
                                            'TIFF format.
    ContactFile = "c:\contact.tif"
    iRet = ImgThumbnail1.SaveAs(ContactFile, ContactOption, 1)
    'Display the resulting contact sheet.
    ImgEdit1.Image = ContactFile
    ImgEdit1.Display
    Exit Sub
Set_Error:
     MsgBox Err.Description, , "Set Error"
     Exit Sub
End Sub
```

UISetThumbSize Method

Displays a dialog box (shown here) from which a user can set the width and height of the thumbnail box.



Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 95 and 98
- √ Imaging for Windows NT 4.0

Usage `object.UISetThumbSize(Image, PageNumber)`

Arguments The UISetThumbSize method has the following parameters:

Parameter	Data Type	Description
Image	String	Optional. Used with the PageNumber parameter to specify the name and page of the image file to be displayed in the sample thumbnail. If not specified, the default value of an empty string will be used, and an image is not displayed.
PageNumber	Long	Optional. Used with the Image parameter to specify the name and page of the image file to be displayed in the sample thumbnail. If a value is not specified, a default value of 1 is used.

Returns Boolean.

Setting	Description
True	The dialog box was dismissed with the OK button, generally indicating new values have been set for the **ThumbHeight** and **ThumbWidth** properties.
False	The dialog box was dismissed with the CANCEL button, indicating the values for the ThumbHeight and ThumbWidth properties are unchanged.

Remarks If the thumbnail box size is altered, new thumbnail representations must be generated for all pages of the specified image file. This is equivalent to calling the **ClearThumbs** method or respecifying the **Image** property.

UISetThumbSize Example — VB

This example demonstrates how a call to the **GetScrollDirectionSize** method can be used to calculate the size needed to display 4 thumbnails in the scrolling direction while displaying 2 thumbnails in the non-scrolling direction.

First, the user is allowed to set the thumbnail width and height. The non-scrolling dimension is then derived from the average of the **GetMimimumSize** and **GetMaximumSize** methods. In this particular example, the scrolling direction is horizontal.

```
Private Sub cmdControlSize_Click()
    Dim lMinSize, lMaxSize, lVSize, lHSize As Long
    Dim bScroll As Boolean
    Dim iNonScrollThumbCount As Long
    bScroll = True
    iNonScrollThumbCount = 2        'Number of thumbnails desired in
                                    'non-scrolling direction.
    ImgThumbnail1.ScrollDirection = Horizontal
    ImgThumbnail1.UISetThumbSize    'User sets thumb size interactively.
    'Determine min and max control size taking a scrollbar into account.
    lMinSize = ImgThumbnail1.GetMinimumSize(iNonScrollThumbCount, bScroll)
    lMaxSize = ImgThumbnail1.GetMaximumSize(iNonScrollThumbCount, bScroll)
    lVSize = (lMinSize + lMaxSize) / 2  'Average of the min & max
                                        'nonscrolling size
    lHSize = ImgThumbnail1.GetScrollDirectionSize(4, iNonScrollThumbCount,
    ➥    lVSize, True)
    'Prior to resizing the Thumbnail control, it is necessary to ensure the
    'unit of measure for the form and Thumbnail control are the same. The
    'thumbnail methods return units in pixels - so the scale mode for the
    'form must be set to pixels.
    Form1.ScaleMode = vbPixel
    ImgThumbnail1.Width = lHSize
    ImgThumbnail1.Height = lVSize
End Sub
```

Standard Events (Thumbnail Control)

Description The following standard Events are used by the Thumbnail control:

- *Click
- *MouseDown
- *DblClick
- Error
- *MouseMove
- KeyDown
- *MouseUp
- KeyUp

Events with an asterisk (*) use the additional argument, ThumbNumber.

Parameter	Data Type	Description
ThumbNumber	Long	The number of the thumbnail on which the event occurred. A 0 (zero) value means an event occurred on the control, but not on a thumbnail.

ThumbDrag Event

Description Drags a specified number of thumbnails from the control.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage **Sub** *object_***ThumbDrag***(DragCount, Shift)*

Arguments The ThumbDrag event has the following parameters:

Parameter	Data Type	Description
DragCount	Long	Number of thumbnails dragged from the Thumbnail control.
Shift	Short	Bit mask that details the state of the Ctrl, Shift, and Alt keys, in addition to mouse key information.

ThumbDrop Event

Description Drops a specified number of thumbnails before a designated thumbnail.

Available With

- √ Imaging for Windows Professional Edition V1.0, V1.1, and V2.0
- √ Imaging for Windows 98
- Imaging for Windows 95
- Imaging for Windows NT 4.0

Usage **Sub** *object_***ThumbDrop***(InsertBefore, DropCount, Shift)*

Arguments The ThumbDrop event has the following parameters:

Parameter	Data Type	Description
InsertBefore	Long	The page number of the thumbnail image before which the dropped item is inserted. A value that is 1 greater than the value of the ThumbCount property indicates an Append operation.
DropCount	Long	The number of thumbnails or image files dropped on the Thumbnail control.

Parameter	Data Type	Description
Shift	Short	Bit mask that details the state of the Ctrl, Shift, and Alt keys, in addition to mouse key information. Drag operations initiated from the Thumbnail control have a bit value of 0x8000.

ThumbDrop Example — VB

This example demonstrates how the Thumbnail control can be updated when images are dropped onto it. A message box is displayed detailing the source file and pages as well as the insertion thumbnail for each thumbnail droppped onto the control. The source can be a drop from Explorer, Imaging for Windows, or selected thumbnails from another Thumbnail control.

For this example, the **EnableDragDrop** property has been set to DropFileDropDragLeftRight. The property must have been set prior to performing any drag or drop operation.

```
Private Sub ImgThumbnail1_ThumbDrop(ByVal InsertBefore As Long, ByVal
    ➥    DropCount As Long, ByVal Shift As Integer)
    Dim DropMsg As String
    On Error GoTo DropError
    'Loop to process each thumb being dropped; index starts at 0.
    For InsertCount = 0 To DropCount - 1
        srcFile = ImgThumbnail1.ThumbDropNames(InsertCount)
        srcPage = ImgThumbnail1.ThumbDropPages(InsertCount)
        'Insert page into image file first, then Thumbnail control.
        ImgAdmin1.Insert srcFile, srcPage, InsertBefore, 1
        'Image file saved after insert.
        ImgThumbnail1.InsertThumbs InsertBefore, 1
        'Provide information regarding dropped page.
        DropMsg = "Page " & Str(srcPage) & " from " & srcFile & " was
        ➥    inserted before page " & Str(InsertBefore)
        MsgBox DropMsg, , "Drop Page"
    Next InsertCount
    Exit Sub
DropError:
    ImgThumbnail1.DropFailed = True
    MsgBox Err.Description, , "Drop Error"
End Sub
```

Image Thumbnail Extender Properties

This section lists the Extender properties that are available when the Image Thumbnail control is drawn on a Visual Basic form.

Name	Description
Container	Returns the container of an object.

Name	Description
DataBindings	Returns the DataBindings collection object containing the available bindable properties.
DataChanged	Returns or sets a value indicating that data in the bound control has changed.
DateField	Returns or sets a value that binds the Thumbnail control to a field.
DragIcon	Returns or sets the mouse pointer icon in a drag-and-drop operation.
DragMode	Returns or sets the drag mode (manual or automatic).
Height	Returns or sets the height of an object.
HelpContextID	Returns or sets the Help context ID for an object.
Index	Returns or sets the subscript value of a control in an array of controls.
Left	Returns or sets the distance between the left edge of an object and the left edge of its container.
Name	Returns the name of an object.
Object	Returns a reference to a property or method of a control that has the same name as a property or method extended to the control.
Parent	Returns the form, object, or collection that contains either a control, or another object or collection.
TabIndex	Returns or sets the tab order of an object within its parent.
TabStop	Returns or sets whether the Tab key can be used to move focus to an object.
Tag	Returns or sets ancillary data.
ToolTipText	Returns or sets a ToolTip.
Top	Returns or sets the distance between the top edge of an object and the top edge of its container.
Visible	Returns or sets whether an object is visible or hidden.
WhatsThisHelpID	Returns or sets the context-sensitive help ID for an object.
Width	Returns or sets the width of an object.

Refer to the Visual Basic on-line help for more information about these properties.

Image Thumbnail Extender Methods

This section lists the Extender methods that are available when the Image Thumbnail control is drawn on a Visual Basic form.

Name	Description
Drag	Begins, ends, or cancels a drag operation.
Move	Moves an object.
SetFocus	Gives focus to the object specified.
ShowWhatsThis	Displays the help topic corresponding to the value of the WhatsThisHelpID property.
Zorder	Places an object at the front or back of the z-order within its graphical level.

Refer to the Visual Basic on-line help for more information about these methods.

Image Thumbnail Extender Events

This section lists the Extender events that are available when the Image Thumbnail control is drawn on a Visual Basic form.

Name	Description
DragDrop	Fires when a drag-and-drop operation is completed.
DragOver	Fires when a drag-and-drop operation is in progress.
GotFocus	Fires when an object receives focus.
LostFocus	Fires when an object loses focus.

Refer to the Visual Basic on-line help for more information about these events.

A

Imaging ActiveX Sample Applications

This appendix describes the Imaging ActiveX sample applications that are available on the *Eastman Software Imaging Professional Developer's Guide* compact disk (CD).

In This Appendix

Overview

This section introduces you to the Imaging ActiveX sample applications.

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. Eastman Software, Inc. 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 Eastman Sample application, to compile and run the Imaging ActiveX sample applications, you must use:

- Microsoft Visual Basic 5.0 with Service Pack 3.
- Imaging for Windows Professional Edition Version 2.0 or greater.

To compile and run the Eastman Sample application, you must use:

- Microsoft Visual Basic 5.0 with Service Pack 3.
- Imaging for Windows 95 (OSR2 release).

Note: Once compiled and made into an executable file, the Eastman Sample application will run on any version of Imaging for Windows — including Imaging for Windows 98, Imaging for Windows NT, and Imaging for Windows Professional Edition. If you see an error message stating that the Image Admin control could not be loaded, you are attempting to load a sample application that is designed for a later version of Imaging for Windows.

Sample Applications

This section describes the Imaging ActiveX 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.

Eastman Sample

The file name for the Eastman Sample project is `EastSamp.vbp`.

Eastman Sample emulates the look and feel of Imaging for Windows 95. It is a baseline image editor that — due to its simplicity — is the best one from which to study and learn.

Note: The term *baseline* means that all of the functions within Eastman Sample were developed using the OSR2 version of Imaging for Windows 95. Once compiled and made into an executable file, it will work with any version of Imaging for Windows.

Later, this document refers to the other, non-baseline sample applications as *advanced* sample applications.

Eastman Sample's 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 `kdkImgEdit1_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 `kdkImgEdit1_DragDrop` and `kdkImgThumbnail1_DragDrop` event procedures.

Image Editor

The file name for the Image Editor project is `ImgEditr.vbp`.

Image Editor is a more sophisticated application that emulates the look and feel of Imaging for Windows Professional Edition Version 2.0.

After you master Eastman Sample, investigate Image Editor — provided you are using Imaging Professional 2.0. The Image Editor 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.

Image Properties

The file name for the Image Properties project is `ImgProp.vbp`.

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.

Image Thumbnails

The file name for the Image Thumbnails project is `ImgThumb.vbp`.

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 on-line 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 Imaging Flow’s on-line help system to learn more about flow variables.

B

Imaging ActiveX Tips and Tricks

This appendix describes some tips and tricks you might find useful when working with the Imaging ActiveX controls.

In This Appendix

Tips and Tricks

This section provides some tips and tricks for using the Imaging ActiveX controls.

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

How to use functions of the *Version 2.0* ActiveX sample applications

With the exception of Eastman Sample, the ActiveX sample applications on the *Eastman Software Imaging Professional Developer's Guide* CD are designed for programmers and users running Imaging for Windows Professional Edition Version 2.0 and greater.

If you and your users are not running this version of Imaging, you may still be able to use some of the functions of the Version 2.0 sample applications.

To do so, perform the following steps:

1. Use a text editor to view the `.frm` files within each Version 2.0 sample application on the CD.
2. When you see a function you want to use, refer to Chapters 7 through 11 of this guide to determine whether the version of Imaging that you and your users are running supports the properties, methods, events, or parameters the function employs.

Note: Eastman Software, Inc. has made every attempt to document the parameters that are available in each version of Imaging for Windows. When setting a property, invoking a method, or responding to an event, if you encounter results you consider unusual, consider changing the parameter(s) you are using. Refer to Appendix A in this guide for more information about the ActiveX sample applications.

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.

Note: Only Imaging for Windows Professional Edition Version 2.0 supports the **RotateAll** method.
Only Imaging for Windows 98 and Imaging for Windows Professional Edition support specifying a rotation amount in the **RotateRight** and **RotateLeft** methods.

Use the DisplayScaleAlgorithm property to scale black-and-white image pages to gray

Eastman Software, Inc. 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 Eastman 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 kdkImgAdmin1.StatusCode <> 0 Then
    MsgBox Err.Description & " (ImgAdmin error " & _
        Hex(kdkImgAdmin1.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.

How to retain generated file names when template scanning

As you know from Chapters 4 and 8, 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.

Refer to Chapters 4, 8, and 10 for more information about scanning.

How to 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 table like it 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
AWD	Black & White	Microsoft proprietary	81 KB	120 KB	149 KB
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

[1] By 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. Refer to Chapter 4 and Chapters 7 though 11 for more information.

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.

Note: If you make the LZW compression type available to your users, you need to negotiate a license with Unisys Corporation.

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 Eastman 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 = kdkImgThumbnail1.ThumbCount To 1 Step -1
   If kdkImgThumbnail1.ThumbSelected(intCounter) = True Then
        lngDeletedPageNo = intCounter
        kdkImgEdit1.ClearDisplay
        kdkImgAdmin1.DeletePages lngDeletedPageNo, 1
        kdkImgThumbnail1.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

When 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.

When 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 tell 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.

C

Imaging ActiveX Controls Summary

This appendix lists by product version the properties, methods and events of the Imaging ActiveX controls

In This Appendix

Overview

This appendix contains a series of tables that list by product version the properties, methods and events of the following Imaging ActiveX controls:

- Image Admin
- Image Annotation Tool Button
- Image Edit
- Image OCR
- Image Scan
- Image Thumbnail

Imaging ActiveX Table Definitions

This section describes the product version headings and cell entries that appear in the Imaging ActiveX tables.

Product Version Headings

The following table lists the product version headings that appear in the Imaging ActiveX tables.

Imaging for Windows Product Version Headings

Heading	Product
Win 95	Microsoft Imaging for Windows 95 (OSR2 release)
NT 4.0	Microsoft Imaging for Windows NT (4.0 release)
Win 98	Microsoft Imaging for Windows 98
NT 5.0	Microsoft Imaging for Windows NT (5.0 release)
Pro 1.0	*Eastman Software* Imaging for Windows Professional Edition Version 1.0
Pro 1.1	*Eastman Software* Imaging for Windows Professional Edition Version 1.1
Pro 2.0	*Eastman Software* Imaging for Windows Professional Edition Version 2.0

Cell Entries

The following table describes the cell entries that appear in the Imaging ActiveX tables.

Cell Entries

Cell Entry	Description
Blank	The property, method, or event is not available.
A	The property, method, or event is available.
+	The property, method, or event is available and contains enhancements not included in a previous version.
++	The property, method, or event is available and contains additional enhancements not included in previous versions.
DT	Setting the property is available at design-time only; reading the property is available at run-time.
HD	The property, method, or event is hidden. For the most part, hidden properties, methods, and events do not appear in the user interface of the development environment; however, the **Object Browser** of Visual Basic V5.0+ can optionally display them. The Imaging ActiveX help system does not describe the hidden properties, methods, and events. If you enter the hidden properties, methods, and events in your code manually, the Imaging software accepts them.

Cell Entries (continued)

Cell Entry	Description
OB	The property, method, or event is no longer supported. For backward compatibility, it is available to the programmer, however it is hidden from the user interface of the development environment. The Imaging ActiveX controls help system discourages the use of obsolete properties, methods, and events and recommends that you use their replacements.
RO	Reading the property is permitted; setting the property is not permitted.
RT	The property, method, or event is available at run-time only. Visual Basic V5.0 displays such properties in the **Properties** window at design time. The value associated with the property is meaningless and cannot be modified. If you attempt to modify the value, VB informs you that the property is only available at run-time. Earlier versions of VB do not list run-time properties in the **Properties** window.
US	The property, method, or event is no longer supported. The return code for an unsupported property or method is replaced with an error code.
WO	Setting the property is permitted; reading the property is not permitted. If you attempt to read the value of a write-only property, the Imaging software returns an error. Write-only properties do not appear in the **Properties** window of VB.

Image Admin Control

This table lists the properties, methods, and events of the Image Admin control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES							
Author			RO	RO	A	A	A
Browse1xReturnedPath							RO
Browse1xReturnedType							RO
CancelError	A	A	A	A	A	A	A
Comments			RO	RO	A	A	A
CompressionInfo	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
CompressionType	HD RO	HD RO +	RO +	RO +	HD RO +	HD RO +	RO +
DefaultExt	A	A	A	A	A	A	A
DialogTitle	A	A	A	A	A	A	A
Domain							A
FileStgLoc1x							A
FileType	HD RO	HD RO +	RO ++	RO ++	HD RO ++	HD RO ++	RO ++
Filter	A	A	A	A	A	A	A
FilterIndex	A	A	A	A	A	A	A
Flags	A	A	A	A	A	A	A

Image Admin Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES							
ForceFileDeletion1x							A
ForceFileLinking1x							A
ForceLowerCase1x							A
ForceTiffSingleStrip							A
HelpCommand	A	A	A	A	A	A	A
HelpContextId	A	A	A	A	A	A	A
HelpFile	A	A	A	A	A	A	A
HelpKey	A	A	A	A	A	A	A
Image	A	A	A	A	A	A	A
ImageHeight	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
ImageResolutionX	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
ImageResolutionY	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
ImageWidth	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
Init1xFindDir							A
InitDir	A	A	A	A	A	A	A
Keywords			RO	RO	A	A	A
NameServer							A

Image Admin Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
PageCount	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
PageNumber	HD RT	HD RT	RT	RT	HD RT	HD RT	RT
PageType	HD RO	HD RO	RO	RO	HD RO	HD RO	RO
PrintAnnotations	A	A	A	A	A	A	A
PrintCollate							A
PrintEndPage	HD RT	HD RT	RT	RT	HD RT	HD RT	RT
PrintNumCopies	A	A	A	A	A	A	A
PrintOrientation							A
PrintOutputFormat	A	A	A	A	A	A	A +
PrintRangeOption	A	A	A	A	A	A	A +
PrintStartPage	HD RT	HD RT	RT	RT	HD RT	HD RT	RT
PrintToFile	A	A	A	A	A	A	A
SaveAsName				A			A
StatusCode	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
Subject			RO	RO	A	A	A

Image Admin Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Title			RO	RO	A	A	A
METHODS							
AboutBox()	A	A	A	A	A	A	A
Append()	A	A	A	A	A	A	A
Browse1x()							A
ConvertDate()							A
CreateDirectory()	A	A	A	A	A	A	A
Delete()	A	A	A	A	A	A	A
DeletePages()	A	A	A	A	A	A	A
GetSysCompression Info()	A	A	A	HD OB	A	A	HD OB
GetSysCompression Type()	A	A +	A +	HD OB +	A +	A +	HD OB +
GetSysFileType()	A	A	A +	HD OB +	A +	A +	HD OB +
GetUniqueName()	A	A	A	A	A	A	A
GetVersion()	HD	HD	HD	A	HD	HD	A
GetVolumeType()							A
ImgQuery()							A

Image Admin Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
ImgQueryEnd()							A
Insert()	A	A	A	A	A	A	A
LoginToServer()							A
LogOffServer()							A
Rename()							A
Replace()	A	A	A	A	A	A	A
SetFileProperties()					A	A	A
SetSystemFileAt-tributes()	A	A +	A ++	HD OB ++	A ++	A ++	HD OB ++
Show1xServerOptDlg()							A
ShowFileDialog()	A	A	A	A	A	A	A
ShowFileProperties()			A	A	A	A	A
ShowFindDialog()							A
ShowPrintDialog()	A	A	A	A	A	A	A
VerifyImage()	A	A	A	A	A	A	A
EVENTS							
FilePropertiesClose					A	A	A

Extender Properties, Methods, and Events of the Image Admin Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Index	RO	RO	RO	RO	RO	RO	RO
Name	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Object	RO	RO	RO	RO	RO	RO	RO
Parent	RO	RO	RO	RO	RO	RO	RO
Tag	A	A	A	A	A	A	A
METHODS							
No Methods							
EVENTS							
No Events							

Image Annotation Tool Button Control

This table lists the properties, methods, and events of the Image Annotation Tool Button control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
AnnotationBackColor	A	A	A	A	A	A	A
AnnotationFillColor	A	A	A	A	A	A	A
AnnotationFillStyle	A	A	A	A	A	A	A
AnnotationFont	A	A	A	A	A	A	A
AnnotationFontColor	A	A	A	A	A	A	A
AnnotationImage	A	A	A	A	A	A	A
AnnotationLineColor	A	A	A	A	A	A	A
AnnotationLineStyle	A	A	A	A	A	A	A
AnnotationLineWidth	A	A	A	A	A	A	A
AnnotationOcrType					A	A	A
AnnotationStampText	A	A	A	A	A	A	A
AnnotationTextFile	A	A	A	A	A	A	A
AnnotationType	A	A	A	A	A+	A+	A+
DestImageControl	A	A	A	A	A	A	A
Enabled	A	A	A	A	A	A	A
hWnd	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
PictureDisabled	A	A	A	A	A	A	A
PictureDown	A	A	A	A	A	A	A
PictureUp	A	A	A	A	A	A	A

Image Annotation Tool Button Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
ReadyState			RT RO	RT RO			RT RO
StatusCode	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Value	RT	RT	RT	RT	RT	RT	RT
METHODS							
AboutBox	A	A	A	A	A	A	A
Draw	A	A	A	A	A	A	A
GetVersion	A	A	A	A	A	A	A
EVENTS							
Click	A	A	A	A	A	A	A
Error	A	A	A	A	A	A	A
KeyDown	A	A	A	A	A	A	A
KeyPress	A	A	A	A	A	A	A
KeyUp	A	A	A	A	A	A	A
MouseDown	A	A	A	A	A	A	A
MouseMove	A	A	A	A	A	A	A
MouseUp	A	A	A	A	A	A	A
ReadyStateChange			A	A +			A +

Extender Properties, Methods, and Events of the Image Annotation Tool Button Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Container	RT	RT	RT	RT	RT	RT	RT
DataBindings	RO	RO	RO	RO	RO	RO	RO
DragIcon	A	A	A	A	A	A	A
DragMode	A	A	A	A	A	A	A
Height	A	A	A	A	A	A	A
HelpContextID	A	A	A	A	A	A	A
Index	RO	RO	RO	RO	RO	RO	RO
Left	A	A	A	A	A	A	A
Name	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Object	RO	RO	RO	RO	RO	RO	RO
Parent	RO	RO	RO	RO	RO	RO	RO
TabIndex	A	A	A	A	A	A	A
TabStop	A	A	A	A	A	A	A
Tag	A	A	A	A	A	A	A
ToolTipText	A	A	A	A	A	A	A
Top	A	A	A	A	A	A	A
Visible	A	A	A	A	A	A	A
WhatsThisHelpID	A	A	A	A	A	A	A
Width	A	A	A	A	A	A	A

Extender Properties, Methods, and Events of the Image Annotation Tool Button Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
Drag	A	A	A	A	A	A	A
Move	A	A	A	A	A	A	A
SetFocus	A	A	A	A	A	A	A
ShowWhatsThis	A	A	A	A	A	A	A
ZOrder	A	A	A	A	A	A	A
EVENTS							
DragDrop	A	A	A	A	A	A	A
DragOver	A	A	A	A	A	A	A
GotFocus	A	A	A	A	A	A	A
LostFocus	A	A	A	A	A	A	A

Image Edit Control

This table lists the properties, methods, and events of the Image Edit control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
AnnotationBackColor	A	A	A	A	A	A	A
AnnotationFillColor	A	A	A	A	A	A	A
AnnotationFillStyle	A	A	A	A	A	A	A
AnnotationFont	A	A	A	A	A	A	A
AnnotationFontColor	A	A	A	A	A	A	A
AnnotationGroupCount	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
AnnotationImage	A	A	A	A	A	A	A
AnnotationLineColor	A	A	A	A	A	A	A
AnnotationLineStyle	A	A	A	A	A	A	A
AnnotationLineWidth	A	A	A	A	A	A	A
AnnotationOcrType					A	A	A
AnnotationStampText	A	A	A	A	A	A	A
AnnotationTextFile	A	A	A	A	A	A	A
AnnotationType	A	A	A	A	A+	A+	A+
AutoRefresh	A	A	A	A	A	A	A
BorderStyle	DT	DT	DT	DT	DT	DT	DT
CompressionInfo	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES							
CompressionType	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ContinuePrinting			RT WO	RT WO	RT WO	RT WO	RT WO
ContinueWithoutUndo			RT WO	RT WO	RT WO	RT WO	RT WO
DisplayICMEnabled			A	A		HD	A
DisplayScaleAlgorithm	A	A	A	A	A	A	A
Enabled	A	A	A	A	A	A	A
FileType	RT RO	RT RO +	RT RO ++	RT RO ++	RT RO ++	RT RO ++	RT RO ++
hWnd	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Image	A	A	A	A	A	A	A
ImageControl	A	A	A	A	A	A	A
ImageDisplayed	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ImageHeight	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ImageModified	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ImagePalette	A	A	A	A	A	A	A
ImageResolutionX	RT	RT	RT	RT	RT	RT	RT

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES							
ImageResolutionY	RT	RT	RT	RT	RT	RT	RT
ImageScaleHeight	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ImageScaleWidth	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ImageWidth	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
MagnifierZoom							A
MouseIcon	A	A	A	A	A	A	A
MousePointer	A	A	A	A	A+	A+	A+
OcrZoneVisibility					A	A	A
Page	A	A	A	A	A	A	A
PageCount	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
PageType	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
ReadyState			RT RO				RT RO
ScrollBars	A	A	A	A	A	A	A
ScrollPositionX	RT	RT	RT	RT	RT	RT	RT
ScrollPositionY	RT	RT	RT	RT	RT	RT	RT
ScrollShortcutsEnabled	A	A	A	A	A	A	A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
SelectionRectangle	A	A	A	A	A	A	A
StatusCode	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
UndoLevels			A	A	A	A	A
UseCheckContinue Printing			A	A	A	A	A
Zoom	A	A	A	A	A	A	A
METHODS							
AboutBox	A	A	A	A	A	A	A
AddAnnotationGroup	A	A	A	A	A	A	A
AutoCrop					A	A	A
AutoDeskew					A	A	A
BurnInAnnotations	A	A	A	A	A	A	A
ClearDisplay	A	A	A	A	A	A	A
ClipboardCopy	A	A	A	A	A	A	A
ClipboardCut	A	A	A	A	A	A	A
ClipboardPaste	A	A	A	A	A	A	A
CompletePaste	A	A	A	A	A	A	A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
ConvertPageType	A	A	A	A	A	A	A
Crop					A	A	A
DeleteAnnotationGroup	A	A	A	A	A	A	A
DeleteImageData	A	A	A	A	A	A	A
DeleteSelected Annotations	A	A	A	A	A	A	A
Despeckle					A	A	A
Display	A	A	A	A	A	A	A
DisplayBlankImage	A	A	A	A	A	A	A
Draw	A	A	A	A	A	A	A
DrawSelectionRect	A	A	A	A	A	A	A
EditSelected AnnotationText	A	A	A	A	A	A	A
ExecuteTextEdit Command			A	A			A
FitTo	A	A	A	A	A	A	A
Flip	A	A	A	A	A	A	A
GetAnnotationGroup	RT	RT	RT	RT	RT	RT	RT
GetAnnotation MarkCount	RT	RT	RT	RT	RT	RT	RT
GetCurrent AnnotationGroup	RT	RT	RT	RT	RT	RT	RT

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
METHODS							
GetRubberStampItem			A	A	A	A	A
GetRubberStamp MenuItems			A	A	A	A	A
GetSelected AnnotationBackColor	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation FillColor	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation FillStyle	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation Font	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation FontColor	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation Image	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation LineColor	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation LineStyle	RT	RT	RT	RT	RT	RT	RT
GetSelectedAnnotation LineWidth	RT	RT	RT	RT	RT	RT	RT
GetSelected AnnotationOcrType					RT	RT	RT
GetVersion	HD	HD	HD	A	HD	HD	A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
HideAnnotationGroup	A	A	A	A	A	A	A
HideAnnotationTool Palette	A	A	A	A	A	A	A
Invert							A
IsClipboardData Available	A	A	A	A	A	A	A
LoadAnnotations							A
ManualDeskew					A	A	A
PrintImage	A	A	A +	A +	A +	A +	A ++
Redo			A	A	A	A	A
Refresh	A	A	A	A	A	A	A
RemoveAllOCRMarks					A	A	A
Rotate					A	A	A
RotateAll	HD	HD	HD	HD	HD +	HD +	A +
RotateLeft	A	A	A +	A +	A ++	A ++	A ++
RotateRight	A	A	A +	A +	A ++	A ++	A ++
Save	A	A	A	A	A	A	A
SaveAnnotations							A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
METHODS							
SaveAs	A	A	A	A	A	A	A
SavePage			A	A	A	A	A
ScrollImage	A	A	A	A	A	A	A
SelectAnnotationGroup	A	A	A	A	A	A	A
SelectFirstOcrZone					A	A	A
SelectNextOcrZone					A	A	A
SelectTool	A	A	A	A	A +	A +	A +
SetCurrentAnnotation Group	RT	RT	RT	RT	RT	RT	RT
SetImagePalette	A	A	A	A	A	A	A
SetRubberStampItem			A	A	A	A	A
SetSelectedAnnotation BackColor	A	A	A	A	A	A	A
SetSelectedAnnotation FillColor	A	A	A	A	A	A	A
SetSelectedAnnotation FillStyle	A	A	A	A	A	A	A
SetSelectedAnnotation Font	A	A	A	A	A	A	A
SetSelectedAnnotation FontColor	A	A	A	A	A	A	A
SetSelectedAnnotation LineColor	A	A	A	A	A	A	A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
SetSelectedAnnotation LineStyle	A	A	A	A	A	A	A
SetSelectedAnnotation LineWidth	A	A	A	A	A	A	A
SetSelectedAnnotation OcrType					A	A	A
ShowAnnotationGroup	A	A	A	A	A	A	A
ShowAnnotationTool Palette	A	A	A	A	A	A	A
ShowAttribsDialog	A	A	A +	A +	A ++	A ++	A ++
ShowMagnifier							A
ShowPageProperties			A	A	A	A	A
ShowRubberStamp Dialog	A	A	A	A	A	A	A
Undo			A	A	A	A	A
ZoomToSelection	A	A	A	A	A	A	A
EVENTS							
BadDocumentFileType							A
CheckContinuePrinting			A	A	A	A	A
Click	A	A	A	A	A	A	A
Close	A	A	A	A	A	A	A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
EVENTS							
DblClick	A	A	A	A	A	A	A
EditingTextAnnotation			A	A			A
Error	A	A	A	A	A	A	A
ErrorSavingUndo Information			A	A	A	A	A
HyperlinkGoToDoc			A	A			A
HyperlinkGoToPage			A	A	A	A	A
KeyDown	A	A	A	A	A	A	A
KeyPress	A	A	A	A	A	A	A
KeyUp	A	A	A	A	A	A	A
Load	A	A	A	A	A	A	A
MagnifierStatus							A
MarkEnd	A	A	A	A	A	A	A
MarkMove			A	A	A	A	A
MarkSelect	A	A	A	A	A	A	A
MouseDown	A	A	A	A	A	A	A
MouseMove	A	A	A	A	A	A	A
MouseUp	A	A	A	A	A	A	A
PagePropertiesClose			A	A	A	A	A
PasteClip				A			A

Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
EVENTS							
PasteCompleted	A	A	A	A	A	A	A
ReadyStateChange			A				A+
Scroll	A	A	A	A	A	A	A
SelectionRectDrawn	A	A	A	A	A	A	A
StraightenPage					A	A	A
ToolPaletteHidden	A	A	A	A	A	A	A
ToolSelected	A	A	A	A	A	A	A
ToolTip	A	A	A	A	A	A	A

Extender Properties, Methods and Events of the Image Edit Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Container	RT	RT	RT	RT	RT	RT	RT
DataBindings				RO			RO
DataChanged				RT			RT
DataField				A			A
DragIcon	A	A	A	A	A	A	A
DragMode	A	A	A	A	A	A	A
Height	A	A	A	A	A	A	A
HelpContextID	A	A	A	A	A	A	A
Index	RO	RO	RO	RO	RO	RO	RO
Left	A	A	A	A	A	A	A
Name	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Object	RO	RO	RO	RO	RO	RO	RO
Parent	RO	RO	RO	RO	RO	RO	RO
TabIndex	A	A	A	A	A	A	A
TabStop	A	A	A	A	A	A	A
Tag	A	A	A	A	A	A	A
ToolTipText	A	A	A	A	A	A	A
Top	A	A	A	A	A	A	A
Visible	A	A	A	A	A	A	A

Extender Properties, Methods and Events of the Image Edit Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
WhatsThisHelpID	A	A	A	A	A	A	A
Width	A	A	A	A	A	A	A
METHODS							
Drag	A	A	A	A	A	A	A
Move	A	A	A	A	A	A	A
SetFocus	A	A	A	A	A	A	A
ShowWhatsThis	A	A	A	A	A	A	A
ZOrder	A	A	A	A	A	A	A
EVENTS							
DragDrop	A	A	A	A	A	A	A
DragOver	A	A	A	A	A	A	A
GotFocus	A	A	A	A	A	A	A
LostFocus	A	A	A	A	A	A	A

Image OCR Control

This table lists the properties, methods, and events of the Image OCR control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES[a]							
CopyToClipBoard					A	A	A
Image					A	A	A
Language					A	A	A
LaunchApplication					A	A	A
Layout					A	A	A
OcrFromClipBoard					A	A	A
OutputFile					A	A	A
OutputType					A	A	A
Pages					A	A	A
ProcessingMode					A	A	A
ProgressDialogCaption					A	A	A
ProgressNotification					A	A	A
Quality					A	A	A
ReadyState							RO
RetainPageLayout					A	A	A
RetainPictures					A	A	A
ShowProgress					A	A	A

a. Several hidden properties named Reserve1, Reserve2, etc. also appear when viewing Image OCR properties in the Object Browser of Visual Basic 5.0. Ignore these properties.

Image OCR Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
StatusCode					RO	RO	RO
TrainingFile					A	A	A
TrainingFileOptions					A	A	A
TrainingThreshold					A	A	A
METHODS							
AboutBox()					A	A	A
GetVersion()							A
LoadDictionary()					A	A	A
SetDefaultValues()					A	A	A
ShowOcr()					A	A	A
ShowOcrOptions()					A	A	A
StartOcr()					A	A	A
StopOcr()					A	A	A
EVENTS							
OcrComplete()					A	A	A
OcrProgress()					A	A	A
ReadyStateChange()							A

Extender Properties, Methods and Events of the Image OCR Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
DataBindings					RO	RO	RO
DataChanged					RT	RT	RT
DataField					A	A	A
Index					RO	RO	RO
Name					RT RO	RT RO	RT RO
Object					RO	RO	RO
Parent					RO	RO	RO
Tag					A	A	A
METHODS							
No Methods							
EVENTS							
No Events							

Image Scan Control

This table lists the properties, methods, and events of the Image Scan control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
CompressionInfo	A	A	HD OB	HD OB	HD OB	HD OB	HD OB
CompressionType	A	A	HD OB +	HD OB +	HD OB +	HD OB +	HD OB +
DestImageControl	A	A	A	A	A	A	A
FileType	A	A	A	A	A	A	A
Image	A	A	A	A	A	A	A
MultiPage	A	A	A	A	A	A	A
Page	A	A	A	A	A	A	A
PageCount	A	A	A	A	A	A	A
PageOption	A	A	A	A	A	A	A
PageType	A	A	HD OB	HD OB	HD OB	HD OB	HD OB
ScanTo	A	A	A	A	A	A	A
Scroll	A	A	A	A	A	A	A
ShowSetupBeforeScan		A	A	A	A	A	A
StatusCode	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
StopScanBox	A	A	A	A	A	A	A
Zoom	A	A	A	A	A	A	A

Image Scan Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
METHODS							
AboutBox()	A	A	A	A	A	A	A
CloseScanner()	A	A	A	A	A	A	A
GetCompression Preference()			A	A	A	A	A
GetPageType CompressionInfo()			A	A	A	A	A
GetPageType CompressionType()			A	A	A	A	A
GetScanCapability()					A	A	A
GetVersion()	HD	HD	HD	A	HD	HD	A
OpenScanner()	A	A	A	A	A	A	A
ResetScanner()	A	A	A	HD OB	A	A	HD OB
ScannerAvailable()	A	A	A	A	A	A	A
SetExternalImage Name()	A	A	HD OB	HD OB	HD OB	HD OB	HD OB
SetPageType CompressionOpts()			A	A	A	A	A
SetScanCapability()					A	A	A
ShowScannerSetup()	US	US	HD US	HD US	US	US	HD US
ShowScanNew()	A	A	A	A	A	A	A
ShowScanPage()	A	A	A	A	A	A	A

Image Scan Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
ShowScanPreferences()	A	A	A	A	A	A	A
ShowSelectScanner()	A	A	A	A	A	A	A
StartScan()	A	A	A	A	A	A	A
StopScan()	A	A	A	A	A	A	A
EVENTS							
PageDone()	A	A	A	A	A	A	A
ScanDone()	A	A	A	A	A	A	A
ScanStarted()	A	A	A	A	A	A	A
ScanUIDone()			A	A			A

Extender Properties, Methods, and Events of the Image Scan Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Index	RO	RO	RO	RO	RO	RO	RO
Name	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Object	RO	RO	RO	RO	RO	RO	RO
Parent	RO	RO	RO	RO	RO	RO	RO
Tag	A	A	A	A	A	A	A
METHODS							
No Methods							
EVENTS							
No Events							

Image Thumbnail Control

This table lists the properties, methods, and events of the Image Thumbnail control by product version.

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
AutoSelect			A	A	A	A	A
BackColor	A	A	A	A	A	A	A
BorderStyle	DT	DT	DT	DT	DT	DT	DT
DataField				A			A
DropFailed			A	A		A	A
Enabled	A	A	A	A	A	A	A
EnableDragDrop			A	A	A	A	A
FirstSelectedThumb	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
HighlightColor	A	A	A	A	A	A	A
HighlightSelected Thumbs	A	A	A	A	A	A	A
hWnd	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Image	A	A	A	A	A	A	A
LastSelectedThumb	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
MouseIcon	A	A	A	A	A	A	A
MousePointer	A	A	A	A	A	A	A
ReadyState			RO	RO			RO

Image Thumbnail Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	Win 95	NT 4.0	Win 98	NT 5.0	Pro 1.0	Pro 1.1	Pro 2.0
PROPERTIES							
ScrollDirection	A	A	A	A	A	A	A
SelectedThumbCount	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
StatusCode	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
ThumbBackColor	A	A	A	A	A	A	A
ThumbCaption	A	A	A	A	A	A	A
ThumbCaptionColor	A	A	A	A	A	A	A
ThumbCaptionFont	A	A	A	A	A	A	A
ThumbCaptionStyle	A	A	A	A	A	A	A
ThumbCount	HD RT RO	HD RT RO	RT RO	RT RO	HD RT RO	HD RT RO	RT RO
ThumbDrop Names(Item)			RT RO	RT RO	RT RO	RT RO	RT RO
ThumbDrop Pages(Item)			RT RO	RT RO	RT RO	RT RO	RT RO
ThumbHeight	A	A	A	A	A	A	A
ThumbSelected (PageNumber)	RT	RT	RT	RT	RT	RT	RT
ThumbWidth	A	A	A	A	A	A	A

Image Thumbnail Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
AboutBox()	A	A	A	A	A	A	A
ClearThumbs()	A	A	A	A	A	A	A
DeleteThumbs()	A	A	A	A	A	A	A
DeselectAllThumbs()	A	A	A	A	A	A	A
DisplayThumbs()	A	A	A	A	A	A	A
GenerateThumb()	A	A	A	A	A	A	A
GetManualThumb Filename()							A
GetManualThumb Page()							A
GetMaximumSize()	A	A	A	A	A	A	A
GetMinimumSize()	A	A	A	A	A	A	A
GetScrollDirection Size()	A	A	A	A	A	A	A
GetVersion()	HD	HD	HD	A	HD	HD	A
InsertThumbs()	A	A	A	A	A	A	A
Refresh()	A	A	A	A	A	A	A

Image Thumbnail Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
METHODS							
SaveAs()							A
ScrollThumbs()	A	A	A	A	A	A	A
SelectAllThumbs()	A	A	A	A	A	A	A
SetManualMode()							A
SetManualThumb()							A
SetSaveAsBackColor()							A
SetSaveAsCaption()							A
SetSaveAsCaption Font()							A
SetSaveAsDimensions()							A
SetSaveAsOrientation()							A
SetSaveAsThumbType()							A
UISetThumbSize()	A	A	A	A	A	A	A
EVENTS							
ReadyStateChange			A	A+			A+
ThumbDrag()			A	A	A	A	A
ThumbDrop()			A	A	A	A	A
Standard Events							
Error	A	A	A	A	A	A	A
KeyDown	A	A	A	A	A	A	A

Image Thumbnail Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
EVENTS							
Standard Events							
KeyUp	A	A	A	A	A	A	A
Standard Events in which ThumbNumber Parameter is added							
Click	A	A	A	A	A	A	A
DblClick	A	A	A	A	A	A	A
MouseDown	A	A	A	A	A	A	A
Standard Events in which ThumbNumber Parameter is added							
MouseMove	A	A	A	A	A	A	A
MouseUp	A	A	A	A	A	A	A

Extender Properties, Methods and Events of the Image Thumbnail Control

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Container	RT	RT	RT	RT	RT	RT	RT
DataBindings				RO			RO
DataChanged				RT			RT
DragIcon	A	A	A	A	A	A	A
DragMode	A	A	A	A	A	A	A
Height	A	A	A	A	A	A	A
HelpContextID	A	A	A	A	A	A	A
Index	RO	RO	RO	RO	RO	RO	RO
Left	A	A	A	A	A	A	A
Name	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO	RT RO
Object	RO	RO	RO	RO	RO	RO	RO
Parent	RO	RO	RO	RO	RO	RO	RO
TabIndex	A	A	A	A	A	A	A
TabStop	A	A	A	A	A	A	A
Tag	A	A	A	A	A	A	A
ToolTipText	A	A	A	A	A	A	A
Top	A	A	A	A	A	A	A
Visible	A	A	A	A	A	A	A
WhatsThisHelpID	A	A	A	A	A	A	A

Extender Properties, Methods, and Events of the Image Thumbnail Control (continued)

	Microsoft Products				Eastman Software Products		
COMPONENT	**Win 95**	**NT 4.0**	**Win 98**	**NT 5.0**	**Pro 1.0**	**Pro 1.1**	**Pro 2.0**
PROPERTIES							
Width	A	A	A	A	A	A	A
METHODS							
Drag	A	A	A	A	A	A	A
Move	A	A	A	A	A	A	A
SetFocus	A	A	A	A	A	A	A
ShowWhatsThis	A	A	A	A	A	A	A
Zorder	A	A	A	A	A	A	A
EVENTS							
DragDrop	A	A	A	A	A	A	A
DragOver	A	A	A	A	A	A	A
GotFocus	A	A	A	A	A	A	A
LostFocus	A	A	A	A	A	A	A

Index

Symbols

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z