

Twain Scanning ActiveX Control

User's Guide
ImageBASIC 3.1

IMAGE  *BASIC*

Diamond Head Software, Inc.
1217 Digital Drive Ste. 125
Richardson, Texas 75081
(972) 479-9205

COPYRIGHT NOTICES

Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of Diamond Head Software, Inc., except in the manner described in the documentation.

This software product contains proprietary software components developed by a number of different software companies, referred herein as "Third Party Licensors". This documentation and the software that you purchased are protected by one or more of the following copyright notices:

Copyright © 1996, 1997 Diamond Head Software, Inc. All rights reserved.

Company and product names mentioned in this documentation are trademarks or registered trademarks of their respective companies. Windows is a trademark and Microsoft is a registered trademark of Microsoft Corporation.

DIAMOND HEAD SOFTWARE INC. AND ITS THIRD PARTY LICENSORS MAKE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. DIAMOND HEAD SOFTWARE, INC. AND ITS THIRD PARTY LICENSORS DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL DIAMOND HEAD SOFTWARE INC. OR ITS THIRD PARTY LICENSORS AND/OR THEIR DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF DIAMOND HEAD SOFTWARE INC. OR ITS THIRD PARTY LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Diamond Head Software Inc.'s and its Third Party Licensors' liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort (including negligence), product liability or otherwise), will be limited to \$50.

Contents

Chapter 1 : Introduction	1
Linking Controls.....	1
Licensing Configuration and Verification.....	2
Chapter 2 : Scanning	5
Introduction to Scanning.....	5
Programming Considerations.....	5
Scanner Hardware Setup.....	7
Scanner Selection.....	8
Scanner Selection -- Graphical.....	8
Scanner Selection -- Programmatic.....	9
Scanner Setup.....	10
Scanner Setup.....	10
Scanning a Document.....	11
Chapter 3 : Reference	15
Technical Reference.....	15
Appendix A : File Formats	23
File and Compression Formats.....	23
File Formats Supported by the Twain Scan Control.....	23
Color Limitations of File Formats.....	24
Index	27

Chapter 1 : Introduction

Linking Controls

Many ImageBASIC controls can accept image data from other ImageBASIC controls. The process of designating where each ImageBASIC component gets its image data is referred to as linking the controls. With the exception of those controls that can directly access files or scanners, all ImageBASIC controls must be linked to another ImageBASIC control to get any image data.

Linking of controls is the primary method of moving an image through a series of processing steps. For example, an image may be originally captured through the Twain Scan control, passed to a TMS Display control for operator verification, optionally routed through a ScanFix control for enhancement, then to a TMS File control to be written to disk, and finally to a TextBridge control for OCR processing to generate indexing information for that file.

Creating the Link

Those ImageBASIC controls which accept image data from another ImageBASIC control have a property named **ImageDataSource**. To create the data link between controls, this property must specify the ImageBASIC control that will be supplying image data. Any image that is received by the source control will also be sent to the linked control. For example, if the **ImageDataSource** property of a TextBridge control is set to a TMS Display control, each time a new image is loaded into the display window, the TextBridge control will receive that image.

Linking at Design Time

As each ImageBASIC control is added to a Form at design time, its **ImageDataSource** property is automatically set to an ImageBASIC control already on that Form. The assignment may be changed at design time by selecting from the drop-down list of available ImageBASIC components. This list is shown with the **ImageDataSource** property in the Properties Window or Object Inspector.

Linking at Runtime

Linking controls at runtime requires only one line of code that can be executed at any time. The source of image data can be changed during program execution by naming another ImageBASIC control in the **ImageDataSource** property, as shown here:

```
TMSDispl.ImageDataSource = TwnScn1.Link
```

Each time the image data flowing from one ImageBASIC control changes, the receiving control's **ImageDataChanged** event is triggered. From this event, any procedure that is to be performed on each image can be started. For example, each time an OCR control's **ImageDataChanged** event occurs, an OCR attempt could be started on the new image data.

Licensing Configuration and Verification

Licensing in ImageBASIC is based on enabling a set number of concurrent seats. The number of seats that can use ImageBASIC is controlled by access to licenses. A license is an entry in a database that allows one computer to run a specific ImageBASIC component.

For example, there is a special type of license for the TMS Display control, another for TextBridge, another for ScanFix, etc. Each one of these licenses also comes in two varieties, runtime and development.

As is suggested by the names, *runtime licenses* are necessary for an *executable* to function properly, and *development licenses* are necessary to *develop* an application using ImageBASIC. A design time license will also function as a runtime license, which obviates the need to add runtime licenses for testing applications during development.

Where the Licenses are Kept

ImageBASIC will find licenses stored in either one of two locations -- in a licensing database or on a hardware key. The hardware key is plugged into the parallel port of the computer using ImageBASIC and will be automatically found each time an ImageBASIC component is used. The licensing database must be created on the site where it will be used and may not be moved from the location in which it is installed.

The inability to move a licensing database is one of its protection features. This attachment to a single location is formed when the licensing database is first created. Although it may not be moved once created, the licensing database can be written to a network drive to which all the machines running ImageBASIC have access. A file called IMGBASIC.INI must be on each of these networked machines. This file contains an entry pointing to the location of the licensing database. ImageBASIC can now search the licensing database for an available copy of each license it needs to run.

This licensing database may also be created on a local drive, activating ImageBASIC on only that one machine. The same INI file is still necessary to pinpoint the location of the database. If you opt to create a separate licensing

database on each individual machine using ImageBASIC, it will of course contain only one copy of each type of license. This one copy is sufficient to activate any number of ImageBASIC-based applications on this one computer.

How the Licenses are Used

As each application that uses ImageBASIC is initiated, or the control is loaded into the development environment, the ImageBASIC licensing server attempts to find the proper license for each component. For example, if an executable has been developed that uses ImageBASIC to scan new images using TWAIN and display the results, that application must be able to find one available *TMS Display runtime license* and one available *Twain Scan runtime license*.

If the application is successful in finding both these licenses, it will load normally and function exactly as it was programmed. When the application finds these licenses and is thereby informed that the correct licenses are available, it simultaneously locks the licenses so that no other application can use the same licenses to run at the same time. However, if two copies of these same licenses are available, another computer will be able to run the same application and will then lock the second license.

When an application that has locked one or more licenses terminates, the licenses are released and can immediately be taken by another user. This is the process by which concurrent licensing for any number of seats may be enabled. If the application ends abnormally -- the user might reboot or a concurrently running Windows application might lock up -- then the release of the licenses is conditional on the network/disk operating system.

For Novell networks, the default time for releasing locks made from a lost connection is five minutes. For other networks, your system administrator should be able to tell you how long the NOS takes to release the lock. In any case, the system administrator should be able to change this default release time to satisfy your particular needs.

If the licensing database has been installed on a stand-alone machine, the locks are immediately released and will again be available when the application is started again.

License Distribution Options

Because the ImageBASIC software is used in two distinct environments, development and runtime, different distribution methods are available to simplify the installation of licenses in each of these sites. As far as the license verification algorithms in the ImageBASIC software are concerned, the application can run as long as it is licensed. The following sections explain the different ways that ImageBASIC can be licensed.

Development Licenses

For developers, the options are hardware licensing and software licensing. Hardware licensing is accomplished through the use of a device called a hardware key that plugs into any parallel port on the development machine. The hardware key separately enables each ImageBASIC component. *The hardware key is the default method for distribution of development licenses*

Under certain circumstances, developers may be unable to use a hardware key. In these cases, the developer can install a licensing database that contains development licenses. Installing this database and changing the database or the hardware key require the developer to run the Licensing Configuration Manager and to call Diamond Head to receive an authorization code that enables the installation or change.

Runtime Licenses

For your clients who are running your compiled application, the same two sources of licensing authorization (the hardware key and the licensing database) are also available. Both have two primary control elements: a configuration program and an authorization program. The configuration program installs the licenses and returns a unique identification string for that installation. The authorization program provides a corresponding key string to complete the installation.

Diamond Head can provide you with the application that generates the authorization code. Because the developer will be able to generate authorization codes for the installation of a runtime licensing database, these clients will call the developer for the code, and Diamond Head Software need not be directly involved.

Another option called Component Licensing is available for special circumstances that make the installation of a licensing database or the distribution of a hardware key unusually difficult. In the Component Licensing system, the ImageBASIC components in an application are authorized as the executable is compiled. The application will then run without a hardware key or licensing database.

Chapter 2 : Scanning

Introduction to Scanning

The conversion of paper documents to digital media in an effort to make storage and retrieval of the information they hold more efficient and timely begins with the scanning process. Once the documents are captured in digital form, they may be further processed to enhance the appearance of the images, to extract the important data on the documents, and to store the images in a readily accessible format.

Beginning with the Twain Scan control to perform the initial capture, ImageBASIC offers controls to aid in the design and implementation of all steps in the storage and retrieval process.

Programming Considerations

All scanning operations, particularly operations based on batch scanning, are processor intensive and require significant system resources. This means that you must carefully consider the available resources when designing your scanning routines. The main issues that must be considered when choosing hardware and when designing the application are

- Memory requirements
- Scanner capabilities
- CPU speed
- Disk drive access time and storage capacity
- Display speed

Memory Required for Scanning, Display and Compression

Images are considerably larger than most other types of data objects that are managed in batch processing.

- A typical data record may reach 1500 bytes
- An optimally compressed bitonal image file is generally 40,000 to 50,000 bytes *per page*.

The raw data required to represent a single letter size page scanned in black and white at 300 DPI is just under one megabyte. This data can be compressed in several different formats, each with its own advantages and disadvantages.

Because of its speed, Run Length Encoding (RLE) compression is applied to all image data held in memory by ImageBASIC. RLE compresses the image data to approximately twenty-five percent of its original size, or about 250 kilobytes. This file size must be considered when transferring images across a network.

The file sizes discussed above are for a bitonal, letter size image scanned at 300 DPI. Higher resolutions and grayscale or color images result in significantly larger volumes of data.

- For example, the same 8.5" X 11" page scanned at 300 DPI, but in 256 level grayscale instead of bitonal, requires almost eight megabytes of raw data to represent it.
- If RLE compression performed as well on grayscale data as it does on bitonal data, a single image will still require approximately two megabytes of RAM.
- Unfortunately, RLE does not usually provide this much compression with grayscale, so that same image will require up to four megabytes of RAM.

When an image is rotated, ImageBASIC creates a copy of the image in the new orientation, thereby doubling the required memory. Aside from the storage space required for the data, additional RAM is needed to perform the compression and decompression algorithms.

All of this required memory is in addition to that used by

- Microsoft Windows
- Network or communication drivers
- Display and print drivers
- The many other files held in memory by the operating system and other applications

After the total requirements for all parts of a scanning system are considered, it can be seen how an application that simply drives a scanner and writes to file will require a PC with at least eight megabytes of RAM. Any additional functionality--even just the addition of several more controls to the project, even if they are not used during scanning--will necessarily require more memory and a more powerful processor.

Application Throughput vs. Scanner Speed

It has been demonstrated that if you need to do more than you are doing now, it will take longer than it does now. If a program updates a database, adding the update of a second or third database or table will increase the time necessary for the program to complete. This same fact must be considered when designing a scanning routine. Consider these observations:

- As discussed above in "Memory Required for Scanning, Display and Compression" on page 5, images are very large data objects. Recognizing this, applications must be designed to accommodate the larger object size.
- Scanning is a resource intensive task. A high speed scanner is capable of tying up a workstation all by itself, performing nothing but file saving, even without displaying the images as they are captured.
- The rated throughput of a scanner is determined by scanning alone, not in conjunction with database queries, network file transfers or image processing tasks.

The most common problem in scanner applications is being too complex. An application that scans, rotates images, does recognition of some kind, updates a database and writes files via network transfer is never going to approach the rated speed of high-end hardware. This is a critical point since scan throughput is normally the bottleneck in an imaging application. The key to a successful scanning application is to keep it simple.

- A scanning application should be limited as much as possible to just configuring and driving the scanner.
- Subsequent processing -- such as image enhancement, the extraction of indexing data through character or bar code recognition, form identification, etc. -- should be done in another program.
- In this way, scan throughput is maximized, and you gain the flexibility to process images independently of the scan operation, optimizing both processes.

Scanner Hardware Setup

In order to use the Twain Scan control, TWAIN drivers must be installed and configured. These drivers are commonly available from your hardware manufacturer, and each manufacturer creates a unique TWAIN driver that communicates with the hardware in question. The ImageBASIC Twain Scan control then communicates with this driver to instruct the scanner.

The installation and setup of the TWAIN drivers varies according to the source. Please refer to the installation instructions supplied with this software. After the TWAIN driver is installed and tested, the ImageBASIC Twain Scan control may be implemented.

Resolve any conflicts that you may have with the basic installation before moving on to the ImageBASIC dialog with the scanner. Once the scanner is properly installed, the rest of the process will work extremely well.

Scanner Selection

Adding scanning capability to an ImageBASIC application is usually simple. ImageBASIC offers an interface with standard dialog boxes and easy routing of the scanned images to file or to a display control. The first step, however, is to make sure that your scanner is properly installed, using whatever setup software is provided by the manufacturer.

After the hardware is setup, ImageBASIC offers two methods of selecting the scanner that you have attached. Once the scanner is selected and ImageBASIC has verified communication with it, you are ready to begin scanning. Selecting a scanner may be accomplished through either a built-in dialog or through a series of methods and properties. Both of these options are detailed below.

Scanner Selection -- Graphical

A standard scanner selection dialog is available for choosing the model of scanner that is being used. This dialog is opened by calling the **SelectScanner** method.

- The dialog box will display the names and models of all of the available TWAIN scanners.
- The user simply selects the correct model and clicks OK. The illustration below shows a typical selection dialog box.

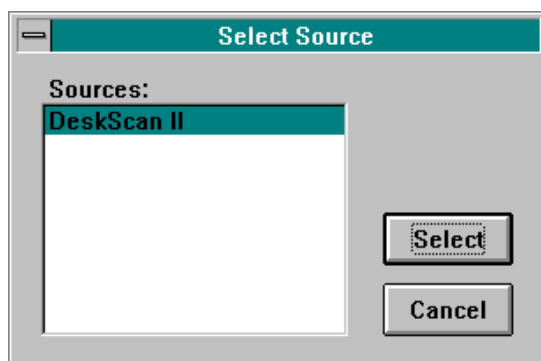


Figure 1 : Standard Scanner Selection dialog box listing all installed TWAIN drivers.

If the scanner is correctly attached and communication is established, ImageBASIC will load the scanner driver into memory and will set the **Scanner** property to the name of the scanner driver file.

Scanner Selection -- Programmatic

To programmatically control the selection and loading of scanner drivers, the following features of the Twain Scan control are available:

GetSavedScanner method	Returns the name of the most recently loaded scanner
Scanner property	Specifies the name of the scanner driver; may be set to the return value from the GetSavedScanner method
LoadScanner method	Loads the scanner driver specified in the Scanner property into memory
UnloadScanner method	Unloads the current scanner driver; does not change the value of the Scanner property

Specifying a Scanner Driver

To select a scanner without using the standard scanner selection dialog box, set the **Scanner** property to the name of the scanner driver. The entry for this property should be the name of the driver as shown in the selection dialog. For example, to use a Hewlett Packard scanner through the DeskScan TWAIN driver, set this property as shown below:

```
TwnScn1.Scanner = TwnScn1.GetSavedScanner
```

Setting this property will not load the driver. The driver will be loaded when setup or scanning starts or when the **LoadScanner** method is executed. Selecting a scanner driver may also be performed through a built-in dialog box as discussed in "Scanner Selection -- Graphical" on page 8.

Finding the Most Recently Used Scanner

To determine the most recently used scanner, execute the **GetSavedScanner** method. This method does not load the driver, nor does it change the value of the **Scanner** property.

```
TwnScn1.Scanner = TwnScn1.GetSavedScanner
```

The driver must be loaded before any attempt to configure or scan is made.

Loading a Scanner Driver

If the **Scanner** property is set to a valid value, the driver will be loaded when any action is performed that requires it. Alternatively, the driver may be explicitly loaded by executing the **LoadScanner** method.

The following code segment will find the most recently used scanner and attempt to load its driver:

```
TwnScn1.Scanner = TwnScn1.GetSavedScanner  
TwnScn1.LoadScanner
```

Unloading a Scanner Driver

The **LoadScanner** method will load the scanner driver specified in the **Scanner** property. The scanner driver will be unloaded when the application is ended or when the **UnloadScanner** method is executed.

```
TwnScn1.UnloadScanner
```

Scanner Setup

All scanners offer some degree of configuration for the scanning process. This may be as simple as setting the page size or as complex as activating and optimizing a scanner-mounted image processing board.

All of the features of each scanner can be accessed only through a built-in dialog that is customized for each scanner through the TWAIN driver.

Scanner Setup

A standard dialog box is supplied through the TWAIN driver that is being used. This dialog allows the user to set the options available for the selected scanner. This dialog box is displayed by calling the **SetupScanner** method:

```
TwnScn1.SetupScanner
```

The scanner setup dialog may be different for each scanner model, depending upon the features available in the hardware. Almost all scanners have options for resolution, page size, brightness and contrast, but many models offer more options. A typical scanner setup dialog box is shown below.

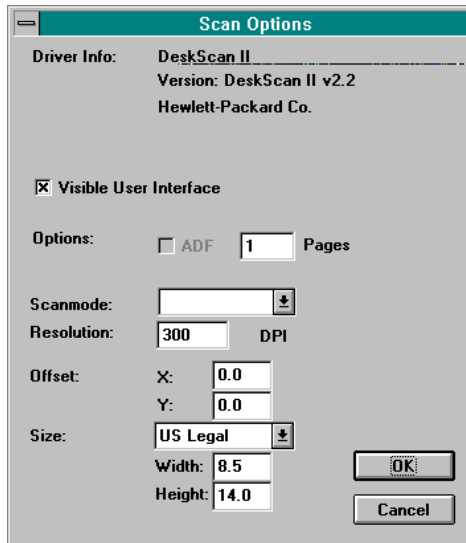


Figure 2 : Standard Scanner Setup dialog box showing basic options and driver versions.

Scanning a Document

When documents are scanned, the images can be sent to another ImageBASIC control for display or processing, or they can be sent directly to file. These two options are not mutually exclusive, however, and images can be simultaneously saved to file and passed to any ImageBASIC control as detailed in the following sections.

Scanning to Another ImageBASIC Control

As image data is created by the scanner and passed to the Twain Scan control, that data can be directly routed to any other ImageBASIC control for display, enhancement, or other processing. However, when designing an application, keep in mind that scanning is a processor intensive task and any additional load placed on the CPU will negatively impact the performance of all tasks.

Specify the Control to Receive the Image

To pass image data from a Twain Scan control to another ImageBASIC control as scanning is being performed, set the **ImageDataSource** of the recipient control to the Twain Scan control. For example, you may wish to display each image in a TMS Display control as it is scanned. Create the link between the controls before scanning begins by executing this command:

```
TMSDisp1.ImageDataSource = TwnScn1.Link
```

As new image files are generated by the scanner, the image data will be forwarded to the TMS Display control. As each new image arrives, the display control's **ImageDataChanged** event will be triggered. In this event, code can be executed to automatically perform any necessary processing on each image.

Scan the Page

To scan a single page into the Twain Scan control, execute the **ScanPage** method. This function will not load the scanner driver; the driver will be loaded when selected through the **SelectScanner** dialog or when the **LoadScanner** method is executed.

```
TwtnScn1.ScanPage
```

When this method is executed, control will pass to the particular TWAIN driver that you have installed. The ImageBASIC application will not regain control until the page is scanned.

Scanning to File

The Twain Scan control will save each image as it is generated. The following properties control the file location and format:

OutputToFile	If True, the new image will be written to the file specified in the following properties; if False, no file will be created
OutputFileFormat	Specifies the output compression type and file type
OutputFileName	Specifies the fully qualified path and file name
OutputFileAppend	If True and the output file already exists, the next image will be appended to the end of the file. If False, the output file will be overwritten if it exists. Only TIFF files support multiple pages.

Set the Output File Name

If the **OutputToFile** property is True, causing the Twain Scan control to write directly to file, the **OutputFileName** property must be set to the fully qualified path and file name to write.

Note: The file name that is specified here can be the fully qualified file name, including drive letter. If the path is not supplied, the file will be written in the current directory.

```
TwtnScn1.OutputFileName = "c:\data\output\010101.tif"
```

Set the File Format

The **OutputFileFormat** property specifies the format of the file written by the scanning control. For example, to create TIFF files using Group 4 compression, set the **OutputFileFormat** property as shown here:

```
TwainScn1.OutputFileFormat = 24
```

The Twain Scan control supports the following file formats:

- 0 Windows BMP
- 1 TIFF Uncompressed *Not yet supported*
- 2 JPEG (JFIF v. 1.1) *Not yet supported*
- 3 Machintosh PICT *Not yet supported*
- 4 UNIX XBM *Not yet supported*

Create Multiple Page Files

TIFF files allow the creation of a single file that contains multiple images. The **OutputFileAppend** property specifies whether the Twain Scan control will attempt to create these multi-page files.

- If **OutputFileAppend** is True and the **OutputFileName** property specifies a file that already exists, the next image will be added to the end of that file.
- If the specified file does not exist, it will be created.

Note: Only the TIFF file type support multiple pages in a single file. The **OutputFileAppend** property will be ignored for other file types, and any existing file will be overwritten.

Scan the Page

To scan a single page into the Twain Scan control, execute the **ScanPage** method. This function will not load the scanner driver; the driver will be loaded when selected through the **SelectScanner** dialog or when the **LoadScanner** method is executed.

```
TwainScn1.ScanPage
```

When this method is executed, control will pass to the particular TWAIN driver that you have installed. The ImageBASIC application will not regain control until the page is scanned. At this time, the **DidScan** event will occur.

Chapter 3 : Reference

Technical Reference

The following pages provide a technical reference guide to the properties, events and methods of the TWAIN ActiveX Control. The cross-references between the topics should give you a good idea of the ways in which these properties interconnect. For procedure-level descriptions of the uses of this control, refer to "Chapter 2 : Scanning" on page 5.

AboutBox Method

Definition:	Displays a message box containing the name of the control, a copyright message and an OK button. Pressing the button will unload the message box.
Parameters:	None
Syntax:	<code>TwnScn1>AboutBox</code>
Return Value:	None
Comments:	This message box provides information about the Annotation control.

Active Property

Definition:	<p>If set to True at design time, the control will fully initialize and verify licensing immediately upon initialization of the runtime application.</p> <p>If set to False at design time,full initialization of the control will be delayed at initialization of the runtime application. In this case, this property must be explicitly set to True at runtime before the control is used.</p>
Data Type:	Boolean
Design Access:	Read/Write
Runtime Access:	Read/Write (see limits below)
See Also:	"Error! Reference source not found!" on page Error! Bookmark not defined.
Comments:	If this property is set to True (the default) at design time, the control is fully initialized and licensing is verified immediately upon initialization of the application at runtime. The related technology libraries are loaded and the control is ready to be used.

If this property is set to False at design time, the control will only partially initialize when the application loads at runtime. By delaying these two actions, the application should be able to load more quickly:

- 1) The related technology libraries for the control will not be loaded.
- 2) The licensing server will not verify an available token for the control.

If the control initializes with **Active** set to False, this property must be explicitly set to True by the application. Until **Active** is set to True, the control will ignore all instructions to it.

If the control fails to find a license token, the **Active** property will be automatically set to False. The application can check this value on Form Load to determine if each control is licensed and can be used.

DidScan Event

- Definition:** Occurs after each page is scanned and image output is complete.
- Parameters:** None
- See Also:** ScanPage Method
- Comments:** This event occurs after the output file is created if **OutputToFile** is True and after any other ImageBASIC control that is linked to the scanner control receives the image data.

Error Event

- Definition:** Occurs for any error internal to this control.
- Parameters:** See Below
- Comments:** This event occurs only for errors internal to the specific control. If no code is placed in this event, the standard dialog is shown and displays the same message that is reported in the Description parameter.

Number	Reports the error number
Description	Reports a descriptive string of the error
SCode	ODBC error return code
Source	Reports a string of the source of the error
HelpFile	Reports the name of a help file that should explain this error
HelpContext	Reports the help context ID in the HelpFile that matches this error
CancelDisplay	If set to 0 in this event, the standard error reporting dialog will not be displayed

Note: If the routine in which the error occurs contains an *OnError* statement, and the runtime error is trapped through this statement, the standard error dialog, which is usually controlled through the *CancelDisplay* parameter, will not be displayed.

GetSavedScanner Method

Definition: Returns the name of the last scanner loaded.

Parameters: None

Syntax: `sScanName = TwnScn1.GetSavedScanner`

Return Value: Scanner driver name on success

Data Type: Long

See Also: Scanner Property, LoadScanner Method, SelectScanner Method

Comments: Successful execution of this method does not change the currently loaded scanner driver, if any, and does not set the **Scanner** property. To find and then load the most recently used scanner, execute code similar to the following:

```
sScanName = TwnScn1.GetSavedScanner
TwnScn1.Scanner = sScanName
TwnScn1.LoadScanner
```

Alternatively, a scanner may be selected and loaded by calling the **SelectScanner** method and choosing a scanner in the dialog.

Link Property

Definition: Unique identification string used in control linking.

Data Type: Long

Design Access: Not Available

Runtime Access: Read-only

See Also: "Linking Controls" on page 1

Possible Values: Any nine-character Link ID

Comments: As each ImageBASIC control is created, a unique Link ID is calculated for the control. This ID is stored in the **Link** property.

When any other ImageBASIC control must link to this Twain Scan control to get image data, the **ImageDataSource** of that recipient control is set to the Link ID of the Twain Scan control. Other ImageBASIC linking is based on this same ID string as well, but the Twain Scan control can supply only image data.

For example, to display images as they are scanned, a TMS Display control may be linked to the Twain Scan control as shown here:

```
TMSDispl.ImageDataSource = TwnScn1.Link
```

In Visual Basic, **Link** is the default property of this control, so the same control communication may be enabled without specifying the **Link** property, as shown here:

```
TMSDispl.ImageDataSource = TwnScn1
```

LoadScanner Method

- Definition:** Loads the scanner driver specified in the **Scanner** property.
- Parameters:** None
- Syntax:** `TwnScn1.LoadScanner`
- Return Value:** 1 on success
0 on failure
- See Also:** Scanner Property, UnloadScanner Method
- Comments:** If another scanner driver is already loaded, it will be unloaded by this method. The loaded driver may also be unloaded by executing the **UnloadScanner** method.
The **Scanner** property may be explicitly set to any valid scanner driver name. The most recently used scanner driver can be found using the **GetSavedScanner** method.
Note: The specified scanner must be available and communicating properly to load the driver.

OutputFileAppend Property

- Definition:** If True, any image saved to file will be appended to the output file if it already exists.
- Data Type:** Boolean
- Design Access:** Read/Write
- Runtime Access:** Read/Write
- See Also:** OutputToFile Property, OutputFileName Property
- Possible Values:** True
False
- Comments:** Some of the file formats supported by ImageBASIC do not support the creation of multi-page files. The file formats that can maintain multiple pages are TIFF, PDA, and DCX. Attempting to append to any other file will cause the file to be overwritten.

OutputFileFormat Property

Definition:	Specifies the format of the output file.
Data Type:	Enumerated
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	OutputToFile Property, OutputFileName Property
Possible Values:	<ul style="list-style-type: none">0 BMP (Windows uncompressed bitmap)1 TIFF (TIFF uncompressed format)2 JPEG (JPEG interchange format JFIF v1.1)3 PICT (Macintosh PICT format)4 XBM (Unix XBM format)
Comments:	<p>Note: As of this writing, only option 0--BMP is supported. The other output formats are scheduled for addition future releases.</p> <p>Different file format are limited in the level and type of color images that they support. Files saved by the Twain Scan control will be of the same color definition as the image received from the scanner. Ensure that the file format selected supports the specified color format. Refer to "Color Limitations of File Formats" on page 24 for a detailed chart.</p>

OutputFileName Property

Definition:	Specifies the file name of the image file to save if the OutputToFile property is True.
Data Type:	String
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	OutputToFile Property, OutputFileFormat Property, OutputFileAppend Property
Possible Values:	Any valid path and file name
Comments:	This property may be set to any valid path and file name. The path may be relative or absolute. As with most file name, the extension may be arbitrarily assigned, but it is recommended that the standard file extensions be applied; e.g., <i>.tif</i> for all TIFF files, <i>.pcx</i> for PCX files, etc.

OutputToFile Property

Definition:	If True, all scanned images will be saved to file. If False, scanned images will be loaded into memory and made available to any other ImageBASIC control whose ImageDataSource is set to this Twain Scan control.
Data Type:	Boolean
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	OutputFileName Property, OutputFileFormat Property, OutputFileAppend Property
Possible Values:	True False
Comments:	All images will be passed to any other ImageBASIC control that is linked to the Twain Scan control (by setting the ImageDataSource of the recipient control). The image may also be saved to file by setting the OutputToFile property to True before scanning is begun.

Scanner Property

Definition:	Specifies the name of the selected scanner driver.
Data Type:	String
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	LoadScanner Method, SelectScanner Method
Possible Values:	Any valid scanner driver name
Comments:	This property may be explicitly set to the name of a valid scanner driver, and the driver will then be loaded when the LoadScanner method is executed. In order to load the most recently used scanner driver, the GetSavedScanner method may be used.

ScanPage Method

Definition:	Instructs the currently selected scanner to scan one page.
Parameters:	None
Syntax:	TwainScan1.ScanPage
Return Value:	0 on failure 1 on success
See Also:	SelectScanner Method, SetupScanner Method, OutputToFile Property

Comments: A scanner driver must be selected before scanning is initiated. Refer to the **Scanner** property or the **SelectScanner** method for details on selecting a scanner driver.

All images that are scanned are made available to other ImageBASIC controls. These controls must name the Twain Scan control in their **ImageDataSource** property to access the images.

If the **OutputToFile** property is True, the scanned images will be saved directly to the file and in the format specified in these properties:

- OutputFileName
- OutputFileFormat
- OutputFileAppend

The **DidScan** event occurs after each page is scanned and saved, if the **OutputToFile** property is True.

SelectScanner Method

Definition: Opens a standard scanner selection dialog box listing all of the installed scanner drivers.

Parameters None

Syntax: TwnScn1.SelectScanner

Return Value: 0 on failure or 'Cancel'
1 on success

See Also: SetupScanner Method, Scanner Property

Comments: Selecting a scanner in this dialog sets the **Scanner** property but does not load the driver. The driver will be loaded when required or when the **LoadScanner** method is executed.

SetupScanner Method

Definition: Opens a dialog box for manipulating the currently selected scanner's options.

Parameters None

Syntax: TwnScn1.SetupScanner

Return Value: 0 on failure or 'Cancel'
1 on success

See Also: SelectScanner Method, ConfigureScanner Method

Comments: The dialog box displays standard setup information such as page size, brightness and contrast, and resolution. Those scanners that offer advanced features will generally have the 'Configure...' button enabled in this dialog. The advanced options dialog that

is displayed when the 'Configure...' button is clicked may also be accessed through the **ConfigureScanner** method.

UnloadScanner Method

Definition:	Unloads any scanner driver currently loaded in memory.
Parameters:	None
Syntax:	<code>TwainScn1.UnloadScanner</code>
Return Value:	0 on failure 1 on success
See Also:	LoadScanner Method
Comments:	This method does not change the value in the Scanner property. Executing this method will cause errors if the scanner driver is currently in use. Before unloading a scanner driver, ensure that all scanning processes and setup dialogs are closed.

Appendix A : File Formats

File and Compression Formats

Most of us do not have a lot of experience with image data. Many of the features in ImageBASIC will be much more easily understood if we present a short primer on how image data is captured, stored and maintained.

When an image is captured at the scanner, it is held as raster data, or a stream of pixels that map together to form a picture. This data stream is relatively large; e.g., the pixels making up a bitonal 8.5" X 11" image scanned at 300 DPI (dots per inch) resolution will take up about 1 megabyte. The scanner usually sends this data either to the CPU or to a special board for compression.

As the data reaches the CPU there are numerous compression options. Each compression option presents different trade-offs with respect to compactness and decompression speed. For data that is being sent to the screen for display, we use a form of compression called Run Length Encoding (RLE) Run Length Encoding is a form of compressing the data that provides for very rapid decompression for quick display. RLE reduces the file size from 1 megabyte to about 250K, so it is somewhat more costly in terms of memory size than some other options.

By default, Diamond Head Software uses TIFF Group 4 compression in files and RLE in memory. Other options are available, however, and an overview of the various compression methods is given below.

File Formats Supported by the Twain Scan Control

BMP File Format

A common Windows graphics format, BMP files can be read by almost any Windows-based image viewer. BMP files are palatted, and can represent up to 24-bit color. However, these files are not compressed and can therefore be very large when storing high color or high resolution images.

PCX File Format

PCX was developed and distributed by Microsoft and ZSoft and is common throughout all Windows installations. Image data is compressed using an RLE scheme, and is therefore quick but not markedly efficient, particularly when storing high color images.

According to its specifications, PCX can encode bitonal, 4-bit, 8-bit, or 24-bit color images up to 64,000 by 64,000 pixels. Typical compression for images or 16 colors or fewer is approximately 50%, while high color and complex images are compressed less.

TIFF File Format

TIFF was initially developed by Aldus Corporation to standardize the storage of bitonal images in document imaging and desktop publishing applications. TIFF is a versatile format composed of a header record listing details of the image followed by the compressed image data. Image data in a TIFF can be compressed using a number of different algorithms, and the exact compression scheme is recorded in the header to allow for easier decompression.

TIFF Group 4, the default format for ImageBASIC, compresses typical bitonal documents to approximately 5% of the original size. Group 4 is a bitonal standard and will not reliably store any grayscale or color images.

Run Length Encoding (RLE)

RLE (Run Length Encoding) is a simple compression algorithm that encodes a run of identical pixels as a count and the pixel value. This compression will generally result in a file size approximately one-quarter the original size. RLE compression is very fast but relatively inefficient; however, because of its speed advantages, it is used as the compression method for images held in memory by ImageBASIC and also in BMP files.

Uncompressed Image Files

Uncompressed images are simply that -- no compression has been applied to the raw image data. Therefore, the image files tend to be very large. For example, the raw data to describe a bitonal letter size image at 300 DPI is about 1 MB. Adding to the area of the image or to the level of color can drastically increase the file size and the time required to transfer and display it.

Color Limitations of File Formats

In the following tables, the entries in the top row of each table indicate the color format of an image file. The three numbers indicate, respectively, BitsPerSample, SamplesPerPixel, and PhotometricInterpretation. The left-most column indicates the type of file format. By finding the intersection of the file type and color format, you can find the most appropriate, fully functional combination for your needs.

BitsPerSample(BPS) indicates how many different values are allowed for each sample (see **SamplesPerPixel**) that defines a pixel's color. Valid values are 1, 4, 8.

SamplesPerPixel(SPP) indicates how many individual values are used to define a pixel's color. Valid values are 1 and 3. 1 indicates that this image is not RGB, leaving bitonal, grayscale, and paletted as possibilities. **BitsPerSample** and **PhotometricInterpretation** will indicate which of these possible formats is used.

PhotometricInterpretation(PI) indicates the interpretation of color values for display. The valid values of **PhotometricInterpretation** are as follows:

- 0 White 0 Typical bitonal/gray interpretation in which 0 indicates white and higher numbers are darker shades.
- 1 White 1 The inverse of White 0, in which 0 is interpreted as black and higher numbers are lighter shades.
- 2 RGB Red, Green, and Blue values of each pixel are specified; this setting requires a value of 3 for **SamplesPerPixel**
- 3 Paletted An array is constructed with a color assigned to each value in than array. This value is generally only applied to color images and is limited to few values than RGB.

The key to the results codes in the tables is as follows:

- Y The combination file type and color definition is fully functional
- N This combination does not work at all

Bitonal and Grayscale Image File Formats

	Binary BPS 1 SPP 1 PI 0/1	16-level gray BPS 4 SPP 1 PI 0/1	256-level gray BPS 8 SPP 1 PI 0/1
TIFF Group 4	Y	N	N
PCX	Y	N	Y
BMP	Y	N	Y

Paletted and RGB Image File Formats

	16-color palette BPS 4 SPP 1 PI 3	256-color palette BPS 8 SPP 1 PI 3	24-bit RGB BPS 8 SPP 3 PI 2
TIFF Group 4	N	N	N
PCX	N	Y	Y
BMP	Y	Y	N

Index

A

- AboutBox Method 15
- Active Property 15
- Appending Output File 13
- Appending Output Files 18
- Application Design Guidelines 5

B

- BMP File Format 23

C

- Configuring the Scanner
 - SetupScanner Method 10

D

- Development Licensing 4
- Dialog
 - Scanner Select 8
 - Scanner Setup 10
- DidScan Event 16
- Driver Loading
 - LoadScanner Method 18
- Driver Name 20
 - Scanner Property 8, 9

E

- Error Event 16
- Events
 - DidScan 16
 - Error 16

F

- File Name of Output 19
- File Writing
 - OutputFileAppend 12
 - OutputFileFormat Property 12
 - OutputFileName Property 12
- Format of Output Files 19

G

- GetSavedScanner Method 9, 17

I

- ImageDataSource Property 11
- Initiating Scanning
 - ScanPage Method 12, 13, 20

L

- Licensing
 - Active Property 15
- Linking Controls
 - ImageDataSource Property 11
- Load Time Improvement
 - Active Property 15
- Loading Scanner Driver
 - LoadScanner Method 9, 10
- LoadScanner Method 9, 10, 18

M

- Memory Requirements 5
- Methods
 - AboutBox 15
 - GetSavedScanner 9, 17
 - LoadScanner 9, 10, 18
 - ScanPage 12, 13, 20
 - SelectScanner 21
 - SetupScanner 10, 21
 - UnloadScanner 9, 10, 22
- Most recently used scanner 9, 17
- Multi-Page Files 13
 - OutputFileAppend 12

O

- OutputFileAppend Property 12, 13, 18
- OutputFileFormat Property 12, 19
- OutputFileName Property 12, 19
- OutputToFile Property 12, 20

P

- PCX File Format 23
- Programming Considerations 5
- Properties
 - Active 15

- OutputFileAppend 12, 13, 18
- OutputFileFormat 12, 19
- OutputFileName 12, 19
- OutputToFile 12, 20
- Scanner 8, 9, 20

R

- RLE Compression 6, 23, 24
- Run Length Encoding Compression 6, 23, 24
- Runtime Licensing 4

S

- Scan Destination 11
 - OutputToFile Property 12
- Scanner Driver, Unloading 22
- Scanner Property 8, 9, 20
- Scanner Selection 21
- Scanner Selection -- GUI 8
- Scanner Selection -- Programmatic 9
- Scanner Setup 10, 21
- Scanner Speed 6
- Scanning Events
 - DidScan 16
- Scanning to File 20
- ScanPage Method 12, 13, 20
- Selecting Scanner Driver
 - Scanner Property 9
- SelectScanner Method 21
- SetupScanner Method 10, 21
- Single Page Scanning 20
- System Requirements of Scanning 5

T

- Throughput vs. Scanner Speed 6
- TIFF File Format 24
- Timing of Licensing Verification
 - Active Property 15

U

- Uncompressed Image Files 24
- Unloading Scanner Driver
 - UnloadScanner Method 9, 10
- UnloadScanner Method 9, 10, 22

V

- Version Information
 - AboutBox Method 15

W

- Writing Files
 - OutputToFile Property 12