

Nestor ICR ActiveX Control

User's Guide

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 are protected by one or more of the following copyright notices:

Portions of this product,© 1995, 1996, 1997 Diamond Head Software, Inc. All rights reserved.

Portions of this product,© 1994, 1995, 1996 Nestor Incorporated. 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 to Nestor	1
Introduction to the Nestor Control.....	1
System Memory Requirements.....	1
Installed Files and File Structure.....	1
Supported File Formats.....	2
Image Processing Work Flow.....	2
Form Design Suggestions.....	6
Linking Nestor to an Image Source.....	8
Licensing Configuration and Verification.....	10
Chapter 2 : Form Definition	15
Overview of Form and Zone Definition.....	15
Reading a ZDF File into Memory.....	15
Saving a ZDF File to Disk.....	15
Document Definition.....	16
Zone Definition.....	20
Zone Types.....	21
Registration Zone Definition.....	22
OMR Zone Definition.....	23
Data Zone Definition.....	24
Data Zone Syntax Definition.....	29
User-Defined Dictionaries.....	35
Chapter 3 : Recognition and Verification	39
Recognition Using the Nestor Control.....	39
Recognition Results.....	40
Format of Results.....	40
Chapter 4 : Technical Reference	43
Reference to Properties, Methods and Events.....	43
Appendix A : Error Codes	69
Error Reporting.....	69
Runtime Error Codes.....	69
NestorReader Engine Error Codes.....	69
Appendix B : File Formats	103
Notice.....	103

Chapter 1 : Introduction to Nestor

Introduction to the Nestor Control

The Nestor ActiveX control performs Intelligent Character Recognition (ICR), the conversion of handwritten image text into machine readable text. In order to make the recognition process as fast and accurate as possible, the document that is being processed must be defined and zones for recognition selected on the form. The process of defining zones and processing options must be performed before ICR begins, but the form definition can be saved and used for all forms of the same type.

This chapter provides an overview of the installation and system requirements of the Nestor control. Also included is an introduction to some common concepts in ImageBASIC, including licensing and linking of controls.

System Memory Requirements

Recognition using the Nestor engine requires at least 6.5 MB of free RAM. Recognition of dirty or distorted images or images with many lines or other graphics can require several more MB of RAM.

If the images being processed are clean and well-defined, the amount of memory used by Nestor can be reduced by changing the `WORK_BUFFER_SIZE` entry in `NESTOR.INI` which may be found in the `IMGBASIC\BIN` subdirectory. This variable defaults to 5000000 but can be set as low as 2000000 to reduce memory use. Increase this value if error message saying that the image is too complex occur.

Installed Files and File Structure

When Nestor is installed, its files are installed in the following directories:

In `AUTOEXEC.BAT`:

- a. The Nestor installation directory is added to the `PATH` statement.
- b. An environment variable called `NESTORPATH` is populated with the installation directory.

Under the ImageBASIC installation directory, the following subdirectories are created:

LIB	Holds the libraries and other necessary files
-----	---

BIN	Holds NESTOR.INI and executables
MEMORIES	Holds the font memories

This directory structure should be duplicated when installing Nestor on a runtime system.

Supported File Formats

The Nestor control can process image files received either from another ImageBASIC control (see the **ImageDataSource** property) or directly from file. *In all cases, only bitonal images can be processed by Nestor* The file must be defined as bitonal. Even if only black and white are present in an image, the file cannot be defined as grayscale or color. The bits per sample and samples per pixel must both be 1 (one).

When processing image from another ImageBASIC control, any image format that can be loaded into ImageBASIC can be recognized.

When processing images directly from file, the following formats are supported:

- DCX
- PCX
- PDA Group 3
- PDA Group 4
- PDA Uncompressed
- TIFF Group 3
- TIFF Group 3 Modified
- TIFF Group 4
- TIFF LZW
- TIFF Packbits
- TIFF Uncompressed

Image Processing Work Flow

The process of using the Nestor control to perform any recognition -- Intelligent Character Recognition (ICR) or Optical Mark Recognition (OMR) -- is essentially a four-step process:

- 1) Define the Form and Zones on the Form
- 2) At the start of recognition, perform preprocessing enhancement
- 2) Perform Recognition and capture the results

3) Verify Recognition Results

Define the Form and Zones on the Form

In order to accurately process an image, the Nestor control must be given one or more zone definitions which include at least the coordinates of the zone and the type of zone (Text or OMR). Additionally, Nestor can perform some image clean-up such as registration and form extraction if the form is adequately defined. All form and zone definition can be made in memory or stored in a file for future use.

The entire set of parameters for recognition is called a Zone Definition File (ZDF). Methods are available to modify information in both the memory-based ZDF and the ZDF file and to update one from the other. Each recognition attempt allows the option of using either the memory-based ZDF or the file-based ZDF.

The exact parameters that can be set for each zone depend upon the type of zone:

Data Zone	For text recognition
OMR Zone	For mark recognition
Registration Zone	For document positioning
Identification Zone	For automated form identification

The sample application that is included with the Nestor control includes a graphical tool that allows the user to view a sample image. The user can then specify the preprocessing that is to be done on similar images, define marks to be used in that preprocessing, and define data zones for data extraction. The interface to zone definition is described in the section titled 'Overview of Form and Zone Definition' on page 15.

After the zones for recognition are defined, new images with data that you want to capture can be processed. This processing begins with preprocessing enhancement to ensure the image is as good as possible before recognition actually begins.

Preprocess the Image

Images coming from a scanner or fax machine are often less than perfect. Some distortions have their origins in the mechanics of scanning the paper to produce the image; examples of this include a skewed image because the paper did not feed into the scanner or fax machine correctly, or noise added to an image due to either an incorrect scanner setting or transmission difficulties while sending a fax. Any distortion of the form in the image from its original size and positioning in the master that was used to define zones for recognition will result in those zones being in the wrong places on the image.

Other image problems are related to the original from which the image was taken. Printed forms introduce problems when the information that was printed on the form overlaps with the information that was written on the form.

Nestor provides an assortment of techniques that deal with different aspects of image preprocessing. These techniques include:

- *Removal of preprinted information* This requires an image of a master form to compare with the image being processed.
- *Noise detection and removal* Both random specks and lines can be removed. The sensitivity of the noise removal is adjustable for the type of noise encountered on a particular type of form.
- *Form deskewing* The image being processed is compared to a master form, and the image being processed is rotated to match the original.
- *Form identification.* Each form that is to be identified must have unique text in a known location.

Not every application will require each of these techniques, so their implementation should be contingent on the balance of processing time versus the accuracy required by the application. The result of preprocessing may be written to disk (refer to 'OUTPUT_IMAGE_NAME' on page 118) so that the effectiveness of the preprocessing may be judged.

Although the preprocessing and the recognition are a single step to the user of the Nestor control -- because the preprocessing enhancement options are part of the form definition -- the next step after enhancement is to perform recognition

Perform Recognition

The initial part of recognition when using the Nestor control is the preprocessing enhancement that is performed by the control when ICR is started. Documents frequently contain specified fields with well-defined character types. Examples include a ZIP code field which will only contain numbers, a date field which might contain numbers and slashes, and a name field which may only contain alphabetic characters. It is often desired to process these fields on many documents with the same basic layout. This may be accomplished by defining a *zone* for each field to be processed on documents of that type. A zone is defined by a region of a document which contains a field to be processed and *zone properties*. Zone properties can be described as specifications of the type of data expected within the zone.

Once a form and the zones on the form are defined, the recognition process can begin. Recognition is always based on a ZDF, whether memory-based or file-based. The options for recognition output and recognition are described in detail

in Chapter 3 beginning with the section titled "Recognition Using the Nestor Control" on page 39. After recognition is performed, the information that is retrieved must be verified for accuracy.

Verify Recognition Results

After recognition is performed and the results are captured, one step that is important in your application is data verification. No automatic form reader is able to read 100% of every form (indeed, two people may disagree as to what a given form says), so there will be some things that will be flagged as possibly needing a human's attention. Nestor assigns a *confidence value* to every character that is recognized. Any character that has a "low" confidence value can then be flagged by the application for a human to decide upon. Whether a character's confidence is "low enough" to require human intervention is usually determined by setting a value called a *confidence threshold*. Characters with a confidence value below this threshold are verified by humans, while characters with higher confidence values are deemed to be correct.

The exact value of the confidence threshold is determined by how "costly" errors are deemed to be in a particular application. In situations where a single error is very expensive, such as the amount of a check, the confidence threshold is usually set to a relatively high value, such as 95 (out of 100). In situations where errors are not very costly, such as a street address, the confidence threshold is usually set to a relatively low value, such as 70, to reduce the amount of data with which humans need to interact. Your application may set a different confidence threshold for each data zone, thus allowing the user to fine-tune his data verification effort.

The Nestor control can report confidence information, alternate characters, and character position for each character in its return string. The verification information is available in the ZRF output format.

Using the Extracted Data

Once the data have been extracted and verified, the data will typically be used for some purpose. Nestor provides programmatic access to the data and also provides several file formats for output so that the information can be used by other programs. Two formats that are quite useful for such purposes are *text* (TXT) format and the *comma separated value* (CSV) format. The TXT format can easily be imported into word processor programs, while the CSV format is a widely-supported import format for databases such as Microsoft Access or Borland Paradox.

For many programmers, though, the most convenient method is to be able to access the extracted data directly from within their program. The results of

recognition are provided through several properties as described in "Recognition Results" on page 40.

Form Design Suggestions

The following advice about how to design forms to achieve high recognition accuracy is reproduced by permission from Nestor Systems, Inc., and provides some of the insights they have developed.

Properly designed forms will help make your ICR installation operate more smoothly and efficiently and save you a great deal of money in the long run. To achieve the highest accuracy in your forms for ICR applications, consider the following elements that contribute to successful installations:

Forms Design

Application Context

Image Quality

Video Correction

Recognition Engine

Despite the most up to date recognition systems, networks, optical disks, GUIs, scanners, and other innovative devices, many recognition problems occur simply because the forms customers use are not well designed. Sometimes the smallest design detail can greatly impact results. For example, the area code of a telephone number can be recorded within parentheses or separated by a slash. Which format is more easily and accurately scanned and which is most likely to be incorrectly recognized as extra "1's" ?

There are some situations in which one cannot design the forms used for ICR. In these cases, one may use a forms removal engine that separates form from content. There are several interesting and useful software packages that serve this purpose.

If, however, the forms can be redesigned, recognition problems can be solved before they occur. For example, most forms have a date field. Even with perfect line removal, customers can be expected to use printed words, cursive words, abbreviations, military formats, and characters that are easily misread, like slashes that resemble the numeral "1". Printing the date field with instructions such as

Date (mm/dd/yy) ____/____/____

can prevent most occurrences of such problems.

The form is a critical part of transferring data into the computer system, Accordingly, the forms we design should take the greatest advantage of the technology available in ICR. Only then can databases be created that empower the user and are efficient and as error free as possible.

Guidelines for Forms Design

Here are some basic guidelines for forms modification that can enhance the success of your installation:

- Use dropout ink colors on recognition fields.

Most companies that print business forms are familiar with dropout colors. Moore Business Forms and Standard Register, for example, have specialized in ICR-readable forms for many years. Today, even the local printing company can provide dropout inks with little difficulty.

Pastel blues and greens drop out of most scanners, whereas other scanners have built-in filters to ignore particular dropout colors. Refer to your scanner manufacturer for information on dropout ink colors.

- Place black lines or corners at the top and bottom of the form.

These provide anchors for proper alignment, de-skewing, and re-sizing fax images that are sometimes distorted during transmission. These unique features can be the best places to locate Registration Zones for your form.

- Use character separation guides on recognition fields.

Either boxes or tickmarks along a straight line (which look like a comb) are helpful in guiding authors to separate characters. For 200-300 DPI images, a good rule of thumb is five (5) characters per inch horizontally, three (3) characters per inch vertically, and one (1) character space (one-fifth of an inch) between horizontal fields.

- Use special application level guides, edits, and context.

Some examples are pre-printed dollar signs in fields specifying monetary amounts, the parentheses and dash for telephone numbers and separated areas for a Social Security number, for example:

Telephone (____) ____ - ____

Total \$ ____, ____. __

Social Security Number ____ - ____ - ____

Whenever you limit the range of acceptable characters within a field, you improve recognition accuracy.

In addition, look for ways to use relationships among multiple fields for context, such as city and state vs. zip code; subtotals versus grand total;

and unit price and units ordered vs. total price. These contextual relationships will enable you to increase accuracy, often without human intervention.

- Place the signature line as far as possible from any recognition field.
John Hancock made a statement with his large signature when he put his name on the Declaration of Independence. People often complete forms with just such a flourish and allow their signatures to extend into an adjacent data field. This contributes to problems in recognition for both fax machines and scanners.
- Properly designing forms will help make your ICR installation operate more smoothly and efficiently, saving you a great deal of time and money in the process. When your customers are involved in the redesign of the forms, their partnership allows them to have a sense of ownership in the application.

Linking Nestor to an Image Source

The Nestor control can read an image for processing directly from file or it can accept it from another ImageBASIC control. The following sections describe the concepts of linking ImageBASIC controls to route information and images within your application.

Types of Inter-Control Communication

The ImageBASIC controls communicate amongst themselves in a unique manner to get information and data to the control that needs it. There are two times that this communication is performed:

- 1) Image data is passed, or
- 2) Services are performed

Each ImageBASIC control can be either a source or a recipient of one or both of these types of communication. The act of specifying the source and recipient of this communication is called *linking* the controls.

All ImageBASIC controls that can accept image data from another ImageBASIC control have a property called **ImageDataSource**. The ImageDataSource property specifies the ImageBASIC control that is the source for all incoming images.

- For example, the TMSSequoia Display control cannot directly access image files on disk.

- The TMSSequoia File control must read the image files into memory where they can be retrieved by the Display control.
- Therefore, the source of images for the TMSSequoia Display control is the TMSSequoia File control. The communication between the controls is enabled by setting the TMSSequoia Display control's ImageDataSource to specify the TMSSequoia File control.

Some ImageBASIC controls offer services other than image processing. The communication for these controls is enabled in a similar manner, but through properties other than the ImageDataSource property.

- For example, the Annotation control does not process image data and, therefore, does not have an **ImageDataSource** property.
- Instead, the Annotation control performs a service -- creating and maintaining annotations. The Annotation control must have a Display control on which to show its annotations.
- The **DisplaySource** property must specify the Display control that will be performing this function.

Methodology of Linking

As each instance of an ImageBASIC control is created on a Form, a unique identification string is calculated for that control. This string, or Link ID, is stored in the **Link** property of each control. This property is not visible at design time and cannot be changed by the application designer. The unique Link ID is calculated each time a control is created, whether it is created statically at design time or dynamically at runtime.

The assignment of a source control is made by setting the Source property of the receiving control -- perhaps the **ImageDataSource** of a TMSSequoia Display control -- to the **Link** property of the source control. For example, for a TMSSequoia Display control to accept image data from a TMSSequoia File control, set the **ImageDataSource** property as shown here for Delphi:

```
TMSDispl.ImageDataSource := TMSFile1.Link
```

The **Link** property is the default property of all ImageBASIC control, so when developing in Visual Basic, is not necessary to specify the property name. In this case, the same assignment shown above may be made as shown here for Visual Basic:

```
TMSDispl.ImageDataSource = TMSFile1
```

Other forms of inter-control communication are established in the same manner. For example, allowing an Annotation control to display its objects on a

TMSSequoia Display control is performed by setting the Annotation control's DisplaySource property as shown here:

```
Annotel.DisplaySource = TMSDispl.Link
```

The links between ImageBASIC controls are made either at design time or at runtime. Changes may be made at any time during runtime if your application requires receiving image data or services from multiple controls.

Changes of Image Data

Each time the image data supplied by one ImageBASIC control changes, the recipient control will automatically refresh with the new image. A change in the image will occur when the File control opens a new file or changes to another page in the same image file. Changes in the supplied image data also occur in other controls when scanning, pasting from the clipboard, and other actions. Each time the image data that is being supplied changes, the recipient control's **ImageDataChanged** event occurs. The **ImageDataChanged** event has no parameters but simply occurs once each time the incoming image changes.

Licensing Configuration and Verification

In order to run, each ImageBASIC control must be able to verify the presence of a valid license token. These license tokens are stored on either a hardware key (shipped with each toolkit purchase) or in a licensing database (the typical toolkit evaluation and runtime distribution format).

Using these tokens, licensing in ImageBASIC is based on enabling a set number of concurrent seats. Each license token allows a single PC to run any number of instances of a single control. For example, a single token for a Display will allow one workstation to concurrently run multiple applications, each of which employs any number of Display controls.

A unique token is required for each different type of ImageBASIC component that you have licensed:

- There is a special type of token for the TMSSequoia Display control, another for the TextBridge control, another for the ScanFix control, and yet more token types for each additional control. The 16-bit and 32-bit versions of each control are also licensed separately.
- Each one of these licenses also comes in two varieties: *runtime* and *development*.

As suggested by the names, runtime licenses are necessary for an executable to function, and development licenses are necessary to develop an application using ImageBASIC.

A design time license will function as a runtime license, removing the need to add runtime tokens for application testing during development.

Where the Licenses are Kept

The ImageBASIC licensing server 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 a parallel port on the computer using ImageBASIC and will be automatically found each time an ImageBASIC component is used.

Note: When using Windows NT, the parallel port must be configured using the Sentinel drivers that are shipped with ImageBASIC.

- 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. Alternatively, each workstation can have its own licensing database installed locally.

A file called IMGBASIC.INI must be on each of PC that is using an license database. This file contains an entry pointing to the location of the licensing database. Based on the entries in this file, ImageBASIC can search the licensing database for an available copy of each license it needs to run.

Creating a Licensing Database

Licensing databases can be installed in two ways:

- 1) If you are installing an evaluation copy of the ImageBASIC CD, run the registration utility that is accessible through the CD Navigator.
- 2) If you are installing a runtime database or are modifying an existing database, run a utility called the License Configuration Manager (LCM). The LCM is installed along with all ImageBASIC toolkit in the TOOLS subdirectory.

How the License Tokens 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 token for each component.

For example, if an executable has been developed that uses ImageBASIC to display existing images, and then calls on TextBridge to perform OCR on that image, that application must be able to find one *Display runtime license*, one *File runtime license* and one *TextBridge runtime license*.

- If the application is successful in finding 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. 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.

- If the application cannot verify the presence of the required tokens, the ImageBASIC components will not operate.

The **Active** property of the controls that did not find license tokens will be set to False. The state of the **Active** property can be verified during Form Load.

Delaying the Locking of Tokens

When the ImageBASIC controls are loaded into a development environment, the licensing server will always immediately attempt to verify that the controls are licensed.

When the controls are loaded at runtime, however, the licensing server can be instructed to delay the verification of available licenses. For this purpose, each control has a property named **Active**. The **Active** property can be changed to False only at design time. If the property is False when a control initializes at runtime, licensing will not be verified and the control's technology libraries will not be loaded. This will make the initialization of an application somewhat faster.

Before using any control that was loaded without full initialization, the **Active** property must be set to True by the application. At this time, the licensing server will find and lock the requisite token and the technology libraries will be loaded. If the proper token cannot be found, **Active** will be set back to False and the control cannot be used.

Licensing Token Release

When an application that has locked one or more tokens terminates normally, the tokens are released and can immediately be taken by another user if a network licensing database is being used. 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 or disk operating system.
- For Novell networks, the default time for releasing a file lock after a connection is lost is 5 (five) minutes.

For other networks, your system administrator should be able to tell you how long the NOS takes to release the lock.

The system administrator should be able to change the 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.

Chapter 2 : Form Definition

Overview of Form and Zone Definition

This chapter provides a description of the definition of a form and zones for recognition. Before recognition can be performed, the zones for recognition must be defined. One way to achieve this is to use the Nestor Demo which provides a graphical user interface to the help in the definition of *Zone Definition File*. Such a file usually has a file extension of ZDF, so this is also called *ZDF file*.

It is also possible to define a ZDF file by using Nestor methods and properties or to read a ZDF file into the session and modify the ZDF information by using the same methods and properties. When doing this, *ZDF buffer* is used to maintain the information associated with a ZDF file in the current session's data.

The definition of a form and the zones on the form should usually be performed graphically so that the user can choose the optimum registration zones and is assured that the data and OMR zones are properly identified. Because the full form and zone definition can be written to file, any given form needs to be defined only once. All recognition on that particular form can then be based on the saved copy of the ZDF file.

Reading a ZDF File into Memory

The full definition of a ZDF file is described in the following sections, but always keep in mind that a ZDF file that is already available can be loaded and modified for a given recognition session. The **LoadZDFFile** method loads a ZDF file into memory and updates all related zone and form definition properties.

Saving a ZDF File to Disk

Because of the complexity of the form and zone definition, it is frequently convenient to design one or more complete zone definition files. Each of these form definition files can be created in memory using the methods and properties of the Nestor control and then saved to a ZDF file using the **SaveZDFFile** method. These ZDF files are then available to process all documents that match the form.

Document Definition

The document definition options are global options which apply to the entire form. This section will describe the process of activating or deactivating the various properties which apply to the image as a whole. The order in which the properties are set is unimportant.

Each option can be defined using properties and is thus maintained in the ZDF buffer in memory. The ZDF buffer contents can be written to a ZDF file for future use in processing multiple forms.

The most commonly used document definition options are available through properties of the Nestor control. Those options that are not exposed as properties but which are available in the Nestor engine can still be accessed through the ZDF file. A full description of the global options in the ZDF file format may be found in "Appendix B : File Formats" in the section titled "ZDF Global Definitions" on page 104.

The following sections describe the document definition options. In addition to the form definition and enhancement options described here, multiple zones within a form can be defined as described in the section titled "Zone Definition" on page 20. When all global and zone options are set and a ZDF is available, recognition can begin as described in the section "Recognition Using the Nestor Control" on page 39.

Long Line Removal

Nestor has a limited ability to remove long vertical or horizontal lines on an image to improve recognition. The following properties activate and set the parameters to Line Removal:

OptLineRemovalActivate	If True, lines that match the following two parameters will be removed before recognition is performed. If False, line removal is not performed.
OptLineRemovalMinLength	Specifies the minimum length that a line must reach to be removed, in image pixels. The default is 75. If the image contains tall or wide letters, this parameter should be increased to avoid removing the letters. Small gaps in the line can be ignored when determining its length.

OptLineRemovalMaxGap

Specifies the maximum length of a gap in a line that can be ignored when determining if the line is long enough to remove.

Noise Processing and Speck Removal

Nestor allows some control over what will be called noise on an image by means of three different values. Generally, the higher the settings for these values, the more "specks" on the image will be called noise. These values are specified in the following properties:

MaxSpeckArea

MaxSpeckWidth

MaxSpeckHeight

If the height of a speck is smaller than the allowed height and its width is less than the allowed width, or if its total area is less than the value specified by the **MaxSpeckArea** property, it will be ignored as noise.

A related value is that of Smart Noise Classification. If this feature is enabled as illustrated below through the **OptSmartNoiseClassification** property, pieces of noise will be evaluated as potentially being detached pieces of characters. If this feature is disabled, specks satisfying the noise criteria will be totally ignored.

```
Nestor1.MaxSpeckArea = 20
```

```
Nestor1.MaxSpeckHeight = 4
```

```
Nestor1.MaxSpeckWidth = 4
```

```
Nestor1.OptSmartNoiseClassification = True
```

Registration

Even though registration zones may be defined for an image, these zones will not be used to try to register or deskew an image unless that registration is activated. Refer to Registration Zone Definition for details on the selection of these zones. The **OptRegisterActivate** property must be set to True to enable registration:

```
Nestor1.OptRegisterActivate = True
```

If the **OptRegisterActivate** property is False, registration will not be performed. Refer to "Registration Zone Definition" on page 22 for details on the configuration of the zones that will be used for registration when the **OptRegisterActivate** property is True.

Form Removal

Nestor allows the removal of forms as part of the preprocessing of an image. To perform form removal specify the name of the "master" form (an image of a blank form) in the **OptFormRemovalTemplate** property. If form removal is not desired, this property should be set to null.

To perform form removal, specify the master form to use:

```
Nestor1.OptFormRemovalTemplate =  
    "c:\imgbasic\master001.tif"
```

To disable form removal, do not supply a master form file name:

```
Nestor1.OptFormRemovalTemplate = ""
```

The best results for form removal are obtained when registration is also being used. If this is the case, the same image that was used to define the registration zones should be used as the master form. Form removal works best when at least three registration zones are defined; it is recommended that four registration zones be used in this case.

Logical Constraints and Syntax Elements

The application may define *logical constraints* which are subsets of the recognition memory's character set. A logical constraint is a definition of a character type, for example *lower case*, *machine print letter* or an *upper case*, *handprint letter*. If the logical constraint specifies that the characters are of mixed or unknown print type the recognition engine will automatically attempt to detect the print type -- machine print or handprint -- on a word by word basis during recognition.

Logical constraint definitions are stored in NESTOR.INI and in a ZDF file. Full details are available in the section titled **Data Zone Syntax Definition** on page 29.

- Logical constraints can be defined using the Nestor Sample Application. This application will write the definitions to the current ZDF.
- The user can directly edit NESTOR.INI to make a definition global for all ZDF files. The entries in this INI file are used if the logical constraints are not defined in the ZDF file.

A logical constraint may specify all characters in a recognition memory's character set. All data zones and syntax definitions must refer to a defined logical constraint rather than to a recognition memory. Logical constraints are, of necessity, defined for an entire ZDF definition rather than on a zone by zone basis, though different zones may of course use only a subset of the defined logical constraints.

Whenever a Registration Zone or Identification Zone is a part of the ZDF definition, it is essential that logical constraint 0 be defined as using a machine print memory such as `ansm0087.me2` or `anhm0096.me2` before the registration or identification information is analyzed. Memory `ansm0087.me2` is defined by default as logical constraint 0 in `NESTOR.INI`.

Based upon the logical constraints that are defined, syntax elements to be used in zone syntax definitions for Data zones are defined as global values:

- 1) Word types are defined based on logical constraints.
- 2) Line types are defined based on word types.
- 3) Zone syntax is defined based on line types.

Creating new logical constraints and syntax elements as a part of the process of building up a zone syntax definition is described in the section "Data Zone Syntax Definition" on page 29.

Output Formats

The **OutputFormat** property specifies the format of the output text that results from recognition, as illustrated here:

```
Nestor1.OutputFormat = 13
```

The following options are available for the enumerated **OutputFormat** property:

- 10 ASCII Database
- 13 ASCII Standard
- 71 Zone Recognition File (ZRF)
- 72 Specified in ZDF

For all options, the **OutputTo** property specifies whether the results will be available in memory or will be written directly to file.

10--ASCII Database outputs a plain text string with all white space compressed.

- If the **OutputTo** property specifies *0--Memory*, the **ResultCount** property will be set to the number of recognition zones in the image. The **ResultIndex** property may be used to loop through all recognition zones to recover the recognition results.
- If the **OutputTo** property specifies *1--File*, the **OutputFileName** property must be set to a valid path and file name. The recognition results for each zone will begin a new line.

13--ASCII Standard outputs a plain text string with all line breaks included and a new line for each zone.

71--ZRF writes the output results to a ZRF file format.

- A ZRF file is almost identical to a ZDF file except that it includes the recognition results in an additional block inside each Data Zone.
- It is strongly recommended that this option be used only when the **OutputTo** property specifies *I--File*.
- If writing directly to file, the **OutputFileName** property must be set to a valid path and file name.

72--Specified in ZDF will read the RESULTS_FORMAT keyword in the ZDF file to determine the output format. Note that this option is valid only when using a ZDF file; i.e., when the **ZDFInputFrom** property is *I--File*. The options for the RESULTS_FORMAT keyword are as follows:

TXT	Outputs plain text and allows an additional value in quotes to determine the delimiter character to use between words; the default delimiter is a space.
CSV	(Comma Separated Value) is a file format which contains the names of the data zones on the first line, with the next line or lines containing the complete recognition results for those zones from one or more images. This is a file format commonly used for import into databases.
ZRF	Causes output to the ZRF format.

Zone Definition

The general concept of Zone Definition includes the definition of characteristics of the document and the image enhancement that can be performed by the Nestor engine. The Nestor engine uses a set of zone and document definitions grouped together as a Zone Definition File (ZDF).

- The ZDF is composed of a set of global variables (such as whether to remove lines) and a series of individual zone definitions. The definition of these options is described in the section titled **Document Definition** on page 16.
- Each zone definition includes data such as the position of the zone and the type of the zone (character or mark recognition). These options are described in the following sections.

The ZDF can be maintained in memory or in a disk file. When held in memory, the attributes of the ZDF are reported through a series of properties of the Nestor control. The Nestor control implements two methods to communicate between the file-based ZDF and the memory-based ZDF. These methods are as follows:

LoadZDFFile	Reads a ZDF file into memory in the ZDF buffer and populates the zone definition properties of the control. The ZoneCount property is updated and the ZoneIndex property may be used to loop through all defined zones.
SaveZDFFile	Writes all zone definitions in the ZDF buffer to a ZDF file. This file will be available for future ICR attempts.

Because multiple zones are typically defined within a single ZDF, the Nestor control creates and tracks zones through the following properties and methods:

ZoneCount Property	Reports the total number of zones that are defined. This property is updated when a ZDF file is loaded or when a zone is added or deleted in memory.
ZoneIndex Property	Specifies which of the currently defined zones is active. The attributes of the current zone are reported and may be set through a series of properties as described in "Field Definition Options", below.
ZoneName Property	Specifies a user-defined name for the zone. A particular zone in memory may be made the active zone by setting either the ZoneIndex property or the ZoneName property.
CreateZone Method	<p>Allocates a new zone and updates the ZoneCount property. This new zone may be selected through the ZoneIndex property and then described through the field-level options.</p> <p>To make a new zone, set the ZoneIndex property to 0 (zero), set the zone coordinate properties and then ZoneType property, then execute the CreateZone method. Full details on the process may be found in the following sections.</p>
DeleteZone Method	Deletes the current zone and updates the ZoneCount property. The ZoneIndex property will be changed only if the last zone was deleted. Otherwise, the remaining zones will be renumbered to fill the gap left by the zone deletion.

Zone Types

Each zone that is defined within the Nestor zone definition structure can be individually optimized for character recognition. Three different types of zones can be defined:

Data Zones

OMR Zones

Registration Zones

These three zone types share some common characteristics but will be described separately in the following sections.

- At least two and up to four Registration Zones should be defined to optimize the accurate positioning of the image so that the Data and OMR Zone coordinates match as closely as possible to the image that is being recognized.

The Registration zones are used to register and deskew the image. It is strongly recommended that four registration zones be defined if Form Removal is enabled through the **OptFormRemovalTemplate** property.

See "Registration Zone Definition" on page 22 for details.

- A Data Zone must be defined for each text region. To optimize the recognition of text, the field may be strictly defined for the type and style of text that is expected.

See "Data Zone Definition" on page 24 for details.

- An OMR Zone should be defined for each region on which the application performs Optical Marks Recognition (OMR).

See "OMR Zone Definition" on page 23 for details.

Registration Zone Definition

Registration Zones are used to deskew incoming images and to correct for some degree of stretching and shrinking that can occur during image capture. The definition of good Registration Zones is absolutely necessary for form removal and for accurate positioning of the Data and OMR zones.

To define a Registration Zone:

- 1) Load a copy of the form that is being defined into an ImageBASIC Display control.
- 2) Link the Nestor control to the Display control.
- 3) Select a region of the image that contains a unique graphic or string. Because it is advisable to specify three or four Registration Zones, select a region that is toward one of the corners or edges. Any type of mark on the image may be used for registration, but it is recommended that the mark be relatively large and separated from other marks.

- 4) Set the **ZoneIndex** property to 0 (zero) to indicate that a new zone is being defined.
- 5) Set the **ZoneType** property to *0--Registration Zone*
- 6) Set the **ZoneName** property to any printable string name for the zone. Zone names cannot include spaces.
- 7) Enter the region coordinates in the **ZoneTop**, **ZoneLeft**, **ZoneRight** and **ZoneBottom** properties.
- 8) Execute the **InitZone** method. This instructs the Nestor engine to scan the region and to analyze the selected mark. The Nestor engine will record its analysis of the mark in the ZDF buffer. After all form definition is complete, the ZDF buffer should be written to file using the **SaveZDFFile** method.
- 9) Execute the **CreateZone** method. This method will allocate a new zone index and update the **ZoneCount** property. The method will return the index of the newly created zone.

OMR Zone Definition

An OMR Zone must be defined for each image region that contains OMR check boxes or marks. If dropout ink is to be used for the form, an image of the form that is printed with regular ink should be used to select the OMR Zones. In this case, the **OptDropoutInk** property should be set to True when defining the zone and when performing recognition.

To define an OMR Zone:

- 1) Load a copy of the form that is being defined into an ImageBASIC Display control.

```
TMSDispl.ImageDataSource = TMSFile1.Link
TMSFile1.LoadFile "c:\images\master001.tif"
```
- 2) Link the Nestor control to the Display control.

```
Nestor1.ImageDataSource = TMSDispl.Link
```
- 3) Select a region of the image that contains OMR check boxes.

Typically, a user will want to visually select the region. One of the Display control's mouse buttons may be set to define the Working Region. This region's coordinates can be passed to the Nestor control to record the position of the OMR Zone, as illustrated in step (7).
- 4) Set the **ZoneIndex** property to 0 (zero) to indicate that a new zone is being defined.

```
Nestor1.ZoneIndex = 0
```

- 5) Set the **ZoneType** property to *1--OMR Zone*.
`Nestor1.ZoneType = 1`
- 6) Set the **ZoneName** property to any printable string name for the zone.
Zone names cannot include spaces.
`Nestor1.ZoneName = "OMR_Zone_1"`
- 7) Enter the region coordinates in the **ZoneTop**, **ZoneLeft**, **ZoneRight** and **ZoneBottom** properties.
`Nestor1.ZoneLeft = TMSDispl.RegLeft`
`Nestor1.ZoneTop = TMSDispl.RegTop`
`Nestor1.ZoneRight = TMSDispl.RegRight`
`Nestor1.ZoneBottom = TMSDispl.RegBottom`
- 8) Set the **OMRUseWeightedCenter** property to True if the check boxes will be marked so that most of the mark is in the center of the check box. For example, filled ovals will not require this option, but X-ed check boxes usually will.
`Nestor1.OMRUseWeightedCenter = True`
- 9) If the forms that will be scanned and recognized will be printed in dropout ink, set the **OptDropoutInk** property to True.
`Nestor1.OMRDropoutInk = True`
- 10) Set the **OMRMinFillForChecked** property to specify how well the check boxes must be filled to count as intentional marks. Higher values in this property will cause incompletely erased or X-ed check boxes to be ignored or considered blank.
`Nestor1.OMRMinFillForChecked = 20`
- 11) Execute the **InitZone** method. This instructs the Nestor engine to scan the region and to analyze the selected check boxes. The Nestor engine will record its analysis of the OMR zone in the ZDF buffer. After all form definition is complete, the ZDF buffer should be written to file using the **SaveZDFFile** method.
- 12) Execute the **CreateZone** method. This method will allocate a new zone with all of the options that were just set. The **ZoneCount** property will be updated, and the method will return the index of the newly created zone.

Data Zone Definition

A Data Zone must be defined for each region of the form that contains text that needs to be captured. Because of the variation of the types of data that may need to be captured in each zone, several limiting parameters are available to optimize recognition.

All Data Zones must be defined using the following properties:

ZoneName	Specifies any valid string name for the zone; no spaces are allowed
ZoneBottom	Specifies the image pixel coordinate of the bottom edge of the zone
ZoneLeft	Specifies the image pixel coordinate of the left edge of the zone
ZoneRight	Specifies the image pixel coordinate of the right edge of the zone
ZoneTop	Specifies the image pixel coordinate of the top edge of the zone

To improve the accuracy of recognition in the Data Zone, the expected contents of the zone may be defined in either of two ways:

- 1) If the exact format of the zone contents is known -- for example, it is an address field -- a Zone Syntax can be defined. Refer to "Data Zone Syntax Definition" on page 29 for details.
- 2) If the exact format of the zone's contents is not known, the expected results can still be limited by specifying the use of a dictionary, a letter type (logical constraint) and context.

After selecting a Data Zone through the **ZoneIndex** or **ZoneName** property, the following attributes of that zone can be set:

Character Repair
Zone Segmentation
Minimum Number of Characters Expected in Zone
Reporting of Punctuation
Minimum Acceptable Confidence
Number of Alternate Characters that are Reported
Zone-Specific Dictionary
Zone Syntax

Character Repair

Character repair may be necessary after line removal or when the text in a zone is low resolution (such as a fax) or fragmented (such as dot matrix). Two Data Zone keywords are available to control character repair:

CHARACTER_FATTENING_INCREMENT

CHARACTER_REPAIR_THRESHOLD

CHARACTER_FATTENING_INCREMENT specifies the number of pixels by which characters will be expanded during repair. Valid only if character repair is enabled. This value must be an integer in the range 1 (one) to 3 (three). This keyword is set using the **SetZoneParam** method as shown below:

```
Nestor1.SetZoneParam "CHARACTER_FATTENING_INCREMENT", "2"
```

CHARACTER_REPAIR_THRESHOLD specifies the threshold of character degradation at which repair is performed. The threshold is compared to a calculated value that indicates how fragmented the characters in an image are, and repair will be done if the fragmentation metric exceeds the threshold. A threshold of 0 (zero) means that character repair will never be done, while a threshold of 100 means that character repair will always be done. The default value is 0 (off).

This keyword is set using the **SetZoneParam** method as shown below:

```
Nestor1.SetZoneParam "CHARACTER_REPAIR_THRESHOLD", "75"
```

Zone Segmentation

Segmentation is the specification of how much text is expected in the zone -- a single character, a single word, a single line, or multiple lines. The expected segmentation of a zone is set using the SEGMENTATION keyword.

Note: Segmentation and Zone Syntax are mutually exclusive. Only one or the other may be specified for a given zone.

This keyword accepts two parameters:

Segmentation Type	Valid values for this parameter are as follows:
-------------------	---

SINGLE_CHARACTER

SINGLE_WORD

SINGLE_LINE

MULTI_LINE

Segmentation Mode	Valid values for this parameter are as follows:
-------------------	---

MIN_SEGMENTATION

STD_SEGMENTATION

MAX_SEGMENTATION

Each mode implies that more processing time will be devoted to separating characters than is spent with the lower setting.

This keyword is set using the **SetZoneParam** method as shown below:

```
Dim strKeyword as String, strKeywordValue as String
strKeyWord = "SEGMENTATION"
strKeywordValue = "MULTI_LINE MAX_SEGMENTATION"
Nestor1.SetZoneParam strKeyword, strKeywordValue
```

In the ZDF file, this keyword entry will appear as shown below:

```
SEGMENTATION SINGLE_LINE STD_SEGMENTATION
```

Minimum Number of Characters in Zone

If the zone is known to contain at least a certain number of characters, or if you wish to return an error if a field is not filled, the minimum number of characters to accepts in a field may be set using the MIN_CHARS keyword. This keyword is set using the **SetZoneParam** method as shown below:

```
Nestor1.SetZoneParam( "MIN_CHARS" , "100" )
```

This keyword accepts integer values from 0 to 500.

If this number of characters is not found, the Recognition Server first searches to the left of the zone and then to the right of the zone for the missing characters. The search stops in each direction when a space big enough to be considered a word break is found. It is an error to return less than the specified number of characters is found for the zone. When MIN_CHARS is used, the zone's Segmentation type must be SINGLE_LINE or SINGLE_WORD.

Reporting Punctuation

Under most circumstances, the punctuation in a handprint field is not important and can be safely ignored. Therefore, the default behavior of Nestor is to treat punctuation as noise and exclude it from the output string. If it is necessary to report punctuation, the FIND_PUNCTUATION keyword may be set. This keyword is set using the **SetZoneParam** method as shown below:

```
Nestor1.SetZoneParam( "FIND_PUNCTUATION" , "TRUE" )
```

This keyword accepts a value of TRUE or FALSE. The parameter value is case-sensitive.

Minimum Acceptable Confidence

Because each zone on a form may require a different level of confidence, the CONFIDENCE_THRESHOLD keyword can be used to define the lowest acceptable confidence. For example, a zone that contains a billing amount may require much more accuracy than a zone that contains the description of the items

ordered. Each character in the zone that falls below this confidence level will be replaced in the output string by the character specified in the global option property **OptUncertainCharReplacement**(and the global keyword **FLAG_UNCERTAIN_CHAR_REPLACEMENT**).

The **CONFIDENCE_THRESHOLD** keyword accepts integer values in the range 0 (zero) to 100, where 0 indicates that no letters will be verified and 100 that almost all characters will be verified. This keyword is set using the **SetZoneParam** method as shown below:

```
Nestor1.SetZoneParam("CONFIDENCE_THRESHOLD", "90")
```

Alternate Characters

The Nestor engine will supply up to two additional characters that it considered as possible alternative to the most confident character. In many cases in which the primary return string is incorrect, one of the alternate characters is correct. Each zone may be individually configured to report 0, 1 or two alternate characters for each character in the return string. The **ALTERNATE_IDS** keyword is used to make this assignment.

The **ALTERNATE_IDS** keyword accepts integer values of 0 (zero) to 2 (two) indicating how many alternate characters to reports:

```
Nestor1.SetZoneParam("ALTERNATE_IDS", "2")
```

Dictionary Usage

Each zone can be assigned a different dictionary to aid in recognition. The dictionary can be either a user-defined dictionary or the standard English language dictionary.

Specification of Dictionary File

The dictionary to be used for a zone is specified by the **DICTIONARY** keyword. This keyword accepts either 1 or 2 additional values. The keyword **DICTIONARY** must always be followed by one of the following parameters:

OFF

ASSIST

CLOSEST_MATCH

- The **ASSIST** keyword specifies that the dictionary will assist recognition, but that the results will not be forced to match a word in the dictionary. This keyword must be followed by the path and filename of the dictionary to use.

- The CLOSEST_MATCH keyword specifies that the output word must be in the dictionary. This keyword must be followed by the path and filename of the dictionary to use.

The DICTIONARY keyword is set using the SetZoneParam method, as shown below:

```
Nestor1.SetZoneParam( "DICTIONARY", "OFF" )
Nestor1.SetZoneParam( "DICTIONARY", "ASSIST
c:\ib\user1.dic" )
```

The creation and maintenance of user-defined dictionaries is detailed in the section "User-Defined Dictionaries" on page 35.

Context Matching

User-defined dictionaries are one of two types -- normal dictionaries and special dictionaries. Normal dictionaries use the same character context statistics as the standard English dictionary. Special dictionaries use only the character context statistics of the words in the special dictionary.

Character context statistics are based on the likelihood of any given character following or preceding another. For example, in the English context, it is unlikely that *aq* will be followed by any letter other than *a*.

The use of context matching is controlled through the CONTEXT keyword. This keyword accepts values of ON or OFF as set through the SetZoneParam method:

```
Nestor1.SetZoneParam( "CONTEXT", "ON" )
```

Zone Syntax

If the exact format and layout of the text in the zone is known -- for example, a date field or an address field -- the Zone Syntax may be defined. Zone Syntax definition is based on building up the word and line types that are expected in the zone. Zone Syntax definition is detailed in the section "Data Zone Syntax Definition" on page 29.

Note: Any one zone may use either Zone Syntax or Segmentation, Zone Constraints, Dictionary, and Context Matching.

Data Zone Syntax Definition

When the exact format or syntax of a zone is known, the best recognition accuracy can be obtained by restricting the output to the known format. For example, if the zone that you are defining is known to be a data field and the form

is preprinted so that the date must be written in a certain format, the zone syntax can be defined so that only that date format will be recognized.

In order to define the syntax for a zone, line and word types must first be defined:

- A zone syntax is built from predefined line types.
- Line types are built from predefined word types.
- Word types are built up of predefined letter types, or logical constraints.

Logical constraints, word types and line types are all global definitions that are available to all zones in a given ZDF. The following sections describe the definition of each of these syntax elements.

Note: If a syntax is defined for a zone, the defined syntax overrides the settings for CONSTRAINT, DICTIONARY, and CONTEXT for that zone. Neither context checking nor dictionaries may be used in conjunction with zone syntax.

Defining Logical Constraints

A logical constraint is essentially the definition of a letter type. For example, a logical constraint can be created that represents only uppercase handprint letters; another logical constraint may represent handprint numerals plus common mathematical symbols such as plus, minus, and equal signs.

A set of default logical constraints is defined in NESTOR.INI which is installed to the IMGBASIC\BIN subdirectory. The logical constraints in this file may be used by any ZDF file to build up word types. The logical constraints in this file use the labels 0 through 9. These labels should not be redefined.

In NESTOR.INI, the default logical constraints are defined as shown below. The syntax of the definition is similar to that used in the ZDF file to define new logical constraints.

```
SYSTEM_CONSTRAINT 0=ansm0087.me2 MP 0-9A-Za-z#$%()*+,-./
SYSTEM_CONSTRAINT 1=ansm0087.me2 MP 0-9
SYSTEM_CONSTRAINT 2=ansm0087.me2 MP A-Za-z#$%()*+,-./
SYSTEM_CONSTRAINT 3=ansh0095.me3 HP 0-9A-Z$( )+-/
SYSTEM_CONSTRAINT 4=ansh0095.me3 HP 0-9
SYSTEM_CONSTRAINT 5=ansh0095.me3 HP A-Z$( )+-/
SYSTEM_CONSTRAINT 6=unhp0093.me3 HP A-Z
SYSTEM_CONSTRAINT 7=unhp0093.me3 HP 0-9
SYSTEM_CONSTRAINT 8=unhp3075.me2 HP 0-9A-Z-
SYSTEM_CONSTRAINT 9=unhp3075.me2 HP A-Z
```

User-defined logical constraints are defined in the ZDF file using the **DEFINE_CONSTRAINT** keyword in the Global section. This keyword uses the following syntax:

```
DEFINE_CONSTRAINT  label memory print_type class_list
```

This keyword is followed at a minimum by a single character which is to be used as the label for the constraint and the name of the recognition memory to be used for this logical constraint. Optionally, the print type (machine print (**MP**), hand print (**HP**), or automatic selection of machine print and hand print (**MH**)) and a list of the allowed classes for this logical constraint may be defined.

The parameters with this keyword are as follows:

- | | |
|-------------------|--|
| <i>label</i> | A single character in the range 0-9, A-Z, or a-z. The <label> is case-sensitive. |
| <i>memory</i> | The name for the recognition memory that is to be used for this logical constraint. NestorReader OT API will look for the recognition memory in the <NESTORPATH>\MEMORIES directory. |
| <i>print_type</i> | May be MP (machine print), HP (hand print), or MH (unknown/mixed). The absence of the <print type> results in the use of the broadest range of print types available in the named <memory> . |
| <i>class_list</i> | May be the word ALL , or a list of allowed classes. The absence of the <class list> is equivalent to ALL . The list of allowed classes may include ranges for contiguous character values such as A-F or 0-9 . The only default contiguous ranges are 0-9 , A-Z , and a-z . Other characters, such as punctuation, hyphens, ampersands, etc. must be listed explicitly. The hyphen (or dash) will be interpreted as a character class when it is seen outside one of these valid range declarations. For instance, a-9 specifies three characters: a , - , and 9 . |

In the Global section of a ZDF file, the following entries could be found:

```
DEFINE_CONSTRAINT A ANHM0079.ME2
DEFINE_CONSTRAINT B ANHM0079.ME2 MP
DEFINE_CONSTRAINT C ANHM0079.ME2 MP 0-9A-Z,.-
DEFINE_CONSTRAINT D ANHM0079.ME2 MP 0123456789ABCD
```

To define a new logical constraint in code, rather than directly in the ZDF file, the **SetZoneParam** method should be used. This method is called as illustrated below:

```
Nestor1.SetZoneParam param_name, param_value
```

For example, to define the same logical constraint that is shown in the samples immediately above, this method may be called as shown here:

```
Dim strConstValue as String
strConstValue = "C ANHM0079.ME2 MP 0-9A-Z,.-"
Nestor1.SetZoneParam "DEFINE_CONSTRAINT", strConstValue
```

Defining a Word Type

A word type is defined by specifying an expected sequence of characters. The characters that make up the word are defined by (1) the number of characters and (2) the logical constraint to be applied to each character.

- Each character in the word is represented by the label of a predefined logical constraint
- Machine print and hand print may be mixed within a word type definition.
- A word type definition may include a single “wild card” character specification. The “wild card” specification is indicated by placing parentheses around the constraint character; for example, the following would indicate that any number of characters of the set defined for logical constraint D are expected for that word.

(D)

- All logical constraint characters which are used in defining a word type must be previously defined with the **DEFINE_CONSTRAINT** keyword in the ZDF file.

In the ZDF file, the word types are defined in the global section using the keyword **WORD_TYPE**, using the syntax shown below:

```
WORD_TYPE label definition
```

The parameters to the **WORD_TYPE** keyword are as follows:

<i>label</i>	The name of this word type. The name can contain no spaces.
<i>definition</i>	The string of logical constraint labels that indicate the type and number of characters in this word.

Examples of *definition* parameters follow:

(A)BBC

Specifies any number of characters to be recognized using logical constraint **A**, followed by two characters to be recognized with logical constraint **B** and one character to be recognized using logical constraint **C**.

ABABAB

Specifies a word type which consists of characters alternating between logical constraint **A** and logical constraint **B**.

The following examples show the definition of several word types:

```
WORD_TYPE CANADIAN_ZIP ABABAB
WORD_TYPE 5DIG_ZIP BBBBBB
WORD_TYPE 9DIG_ZIP BBBBBBCBBBB
WORD_TYPE CITY_WORD (A)
WORD_TYPE STATE_ABBR AA
WORD_TYPE TEXT_WORD (A)
```

A word type may be defined in the ZDF buffer in memory using the **SetZoneParam** method. This method is called as illustrated below:

```
Nestor1.SetZoneParam param_name, param_value
```

For example, to define one of the word types that is shown in the samples immediately above, this method may be called as shown here:

```
Nestor1.SetZoneParam "WORD_TYPE", "5_DIGIT_ZIP 11111"
```

Where 1 is the logical constraint label for numerals. This is the default definition of the system constraint that is defined in NESTOR.INI.

Using the word types that are defined, the next step is to define line types

Defining a Line Type

Line types are defined based on a series of word types. Just as a word type is built up from a series of logical constraints

- The expected format of a line is specified by defining the sequence of word types that might be found on that line using the global keyword **LINE_TYPE**
- Wild cards (indicated by placing a word type in parentheses) may be used to indicate the possibility of multiple words of a given type
- The vertical bar (|) may be used to indicate that there are two or more possible formats for a word in a given position.

- As in the word type specification, only one wild card may be used per line type definition so that ambiguous definitions may be avoided.

An example of a line type that describes the last line of an address which contains the city state, and ZIP code is:

```
LINE_TYPE ZIP_LINE (CITY_WORD) STATE_ABBR
          5DIG_ZIP|9DIG_ZIP
```

Where,

LINE_TYPE is the keyword defining a new line type for use in a zone syntax,

ZIP_LINE is the name of the line type that is being defined,

The remainder of the line is the list of the word types that can be present.

Note that the city name uses a wild card because city names may use two or more words (New York, Salt Lake City, etc.). The ZIP code may be either 5 digits or 9 digits with a hyphen.

Defining a Zone Syntax

A zone's syntax is defined by using one or more line types.

- The beginning of the zone's syntax definition consists of a line containing the keyword **ZONE_SYNTAX_START**

This is followed by one or more lines which specify the names of previously-defined line types, possibly enclosed by parentheses to indicate a wild card.

The final line of the zone's syntax definition consists of a line containing the keyword **ZONE_SYNTAX_END**

- If more than a single line is expected in the zone, the position of the line type in the zone syntax definition indicates the position of the line in the zone.
- Wild cards may be used in the definition of a zone, and such wild cards are indicated by placing parentheses around the line type name. A single wild card line is allowed in each Zone Syntax.
- Alternative line types may not be specified.

A sample definition of a zone which has an address consisting of a line with a name, one or more lines of street address, and a line with the city, state, and ZIP code is:

```
ZONE_SYNTAX_START
NAME_LINE
(ADDRESS_LINE)
```

ZIP_LINE

ZONE_SYNTAX_END

Zone Syntax Output

The zone syntax logic will attempt to fit the results of the recognition into the format specified for each word in the output. In doing so, the zone syntax logic takes into account not only the output of the recognition, but the syntax definition (which may include multiple possible interpretations of a single word) and other internal information. The overall confidence for every possible combination of alternative word types is used to select a valid combination of word types (as defined by the line type) which has the highest possible confidence.

For an individual word, it is possible that there will be either more or fewer characters than is allowed by the word type definition. If there are more characters than the syntax allows, the extra characters on the end of the word will have their classes set to "*" and confidence values set to 0, but their coordinates will be preserved. If there are fewer segmented characters than the syntax demands, additional characters will be artificially added to the word to make it correspond to the minimum length required. The added characters will have their classes set to "*" and confidences of 0, and their coordinates will be the same as the last character in the word.

User-Defined Dictionaries

There are two types of user-defined dictionaries used by Nestor: Normal dictionaries and Special dictionaries. The difference between the two types is that Normal dictionaries use the character context statistics of the English language dictionary, while the Special dictionary calculates its own character context statistics based only on the words in the dictionary.

For most recognition tasks, a normal dictionary would probably be most useful. For special applications in which all possible inputs are known in advance, a special dictionary is likely to provide better accuracy.

The application designer has the option of creating, editing, and combining both Normal and Special dictionaries. The following sections show how both of these dictionary types can be created and managed.

Make a User-Defined Dictionary

To create a new dictionary:

- 1) Make an ASCII text file containing all the words you want, one word per line with no spaces on a line
- 2) Execute the **CreateDictionary** or **CreateSpecialDictionary** method to create the dictionary based on the specified text file.

The **CreateDictionary** method creates a Normal dictionary

The **CreateSpecialDictionary** method creates a Special dictionary

These methods accept two parameters, the name of the text file and the name of the dictionary file to create:

```
Nestor1.CreateDictionary "medical.txt", "n_user.dic"
Nestor1.CreateSpecialDictionary "medical.txt",
"s_user.dic"
```

An existing dictionary created using these methods can be edited as described in the following sections. Words may be added to a dictionary, deleted from a dictionary, or two dictionaries can be combined into a third.

Add Words to a Dictionary

To add more words to an existing user-defined dictionary, the **AddWordToDict** method should be used. This method accepts two parameters: the word to add and the path and file name of the dictionary:

```
Dim strWord as String, strDictName as String
strWord = Text1.Text
strDictName = "c:\imgbasic\user001.dic"
Nestor1.AddWordToDict strWord, strDictName
```

An application could perform the addition based on the contents of a text file similar to one used to create the dictionary; i.e., a plain text file with one word to remove one each line.

Remove Words from a Dictionary

To remove words from an existing user-defined dictionary -- for example, a dictionary of employee names may need to be updated -- use the **RemoveWordFromDictionary** method. This method accepts two parameters: the word to add and the path and file name of the dictionary:

```
Dim strWord as String, strDictName as String
strWord = Text1.Text
strDictName = "c:\imgbasic\user001.dic"
Nestor1.RemoveWordFromDict strWord, strDictName
```


Combine Multiple Dictionaries

Two normal dictionaries or two special dictionaries can be combined into a third dictionary of the same type. Two dissimilar dictionary types cannot be combined. Dictionaries are combined using one of the following methods:

CombineDictionaries

CombineSpecialDictionaries

Both methods accepts three parameters:

First source dictionary path and filename

Second source dictionary path and filename

Path and filename of new dictionary

For example, to combine two dictionary files named user01.dic and user02.doc into a third dictionary, the **CombineDictionaries** method would be called as shown below:

```
Dim strDict1 as String, strDict2 as String
Dim strDict3 as String
strDict1 = "c:\imgbasic\user01.dic
strDict2 = "c:\imgbasic\user02.dic
strDict3 = "c:\imgbasic\user09.dic
Nestor1.CombineDictionaries strDict1, strDict2, strDict3
```


Chapter 3 : Recognition and Verification

Recognition Using the Nestor Control

Recognition Events

During the recognition attempt, the Nestor engine can trigger certain events. These events can report errors or request information from the operator. Using these events, the developer can create a custom look and processing system based on individual requirements. The events that can occur as follows:

Progress	Occurs frequently throughout the processing performed by the control. Reports the current progress in the recognition process and allows canceling of the attempt.
Error	<p>Occurs each time an error internal to the Nestor control is detected. This event reports the following parameters:</p> <p><i>Number</i> A long error code that identifies the error</p> <p><i>Description</i> Descriptive string of the error</p> <p><i>SCode</i> The Visual Basic runtime error number which is equivalent to the local error code – reported in the <i>Number</i> parameter – plus the Visual Basic constant vbObjectError (-2147221504).</p> <p><i>Source</i> Descriptive string of the source of the error</p> <p><i>HelpFile</i> Suggested help file name that should have a detailed explanation of the error</p> <p><i>HelpContext</i> Context ID of the appropriate topic in the help file named above</p> <p><i>CancelDisplay</i> If set to True during this event, the standard error dialog will not be displayed. This dialog should be turned off if the application is trapping errors (through the On Error statement) or if you prefer to allow Visual Basic's error message box to be displayed.</p>

Recognition Results

Format of Results

The results will be reported as specified in the **OutputFormat** property or in the RESULTS_FORMAT keyword in the ZDF file. When output is to any format that will be parsed or imported into another application (e.g., ZRF or CSV formats), it is advisable to write the output to file.

- If the **OutputTo** property is *1--File*, the recognition results will be written to the file specified in the **OutputFileName** property.
- If the **OutputTo** property is *0--String*, the recognition results will be reported in the **Results** property.

Output To File

When the recognition results are written directly to file, the format of the text depends upon the output format:

ZRF	This file format is a plain text format that is very similar to the ZDF file. The same keywords and information that were in the ZDF that was used to process the image will also be recorded in the ZRF file. The ZRF file differs by the addition of recognition results which are included in each zone definition section of the file. The recognition results may be found between the keywords START_RESULTS and END_RESULTS within each zone. The results include confidence information and alternate characters. For more information on the format of the results output, refer to "START_RESULTS, END_RESULTS" on page 129.
CSV	This format lists all of the Zone names on the first line, separated by commas. The results of all zones are on the next line, each separated from the last by a comma. No alternate characters or character information is available in this format.
TXT	A plain text file of all zone recognition results. No alternate characters or character information is available in this format.

Output to String

After each successful recognition attempt, the **ResultCount** property will be updated. The **ResultCount** property indicates the number of zones reported for

the most recent recognition. The **ResultIndex** property may be set to any integer value from 1 (one) to **ResultCount**.

When the **ResultIndex** property is set to a valid value, the following properties are updated:

Result	Reports the string of the recognition results for the zone. If a Data Zone is being reported, the recognized text will be shown. If an OMR Zone is being reported, the text assigned to the checked box in the , if any, will be reported. Otherwise, the check box will be reported as checked or not.
ResultZoneName	Reports the name assigned to the current zone when the zone was initially defined, or as specified in the ZDF that was used to perform this recognition.
ResultZoneCharCount	Reports the number of characters in the current zone. Additional information for each character may be available.
ResultZoneLineCount	Reports the number of lines of text in the current zone.

Chapter 4 : Technical Reference

Reference to Properties, Methods and Events

AboutBox Method

Definition:	Displays a message box showing version and copyright information when queried.
Syntax:	<code>Nestor1.AboutBox</code>
Design Access:	Read-only
Runtime Access:	Read-only
Comments:	The message box is application modal and contains a single OK button. Clicking the button will close the message box.

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
Syntax:	<code>Nestor1.Active = True</code>
Design Access:	Read/Write
Runtime Access:	Read/Write (see limits below)
See Also:	"Licensing Configuration and Verification on page 10
Comments:	<p>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.</p> <p>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:</p> <ol style="list-style-type: none">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.

AddWordToDict Method

Definition: Adds a single word to an existing dictionary file; can be used only with normal dictionaries.

Parameters: *dictionary*
word

Syntax: `Nestor1.AddWordToDict dictionary, word`

Return Values: True on success
False on error

Data Type: Boolean

See Also: CreateDictionary Method, CreateSpecialDictionary Method

Comments:

CombineDictionaries Method

Definition: Combines two normal dictionary files into a single dictionary.

Parameters: *dict1* Path and file name of one of the source dictionary files
dict2 Path and file name of the second source dictionary file
dict3 Path and file name of the output dictionary that is the combination of the first two

Syntax: `Nestor1.CombineDictionaries dict1, dict2, dict3`

Return Values: True on success
False on error

Data Type: Boolean

See Also:

Comments:

CombineSpecialDictionaries Method

Definition: Combines two special dictionary files into a single dictionary.

Parameters: *dict1* Path and file name of one of the source dictionary files
 dict2 Path and file name of the second source dictionary file
 dict3 Path and file name of the output dictionary that is the combination of the first two

Syntax: `Nestor1.CombineSpecialDictionaries dict1, dict2, dict3`

Return Values: True on success
 False on error

Data Type: Boolean

See Also:

Comments:

ConvertDictionaryToTextFile Method

Definition: Converts an existing dictionary to a plain text file.

Parameters: *dict* Path and file name of the dictionary file
 text Path and file name of the text file to write

Syntax: `Nestor1.CoconvertDictionaryToTextFile dict, text`

Return Values: True on success
 False on error

Data Type: Boolean

See Also:

Comments:

CreateDictionary Method

Definition: Accepts a plain text file and makes a normal dictionary from it.

Parameters: *text* Path and file name of the text file that contains the words that compose the dictionary
 dict Path and file name of the dictionary file to write

Syntax: `Nestor1.CreateDictionary text, dict`

Return Values: True on success
 False on error

Data Type: Boolean

See Also:

Comments:

CreateSpecialDictionary Method

Definition: Accepts a plain text file and makes a special dictionary from it.

Parameters: *text* Path and file name of the text file that contains the words that compose the dictionary
dict Path and file name of the dictionary file to write

Syntax: Nestor1.CreateSpecialDictionary *text, dict*

Return Values: True on success
False on error

Data Type: Boolean

See Also:

Comments:

CreateZone Method

Definition: Allocates a new zone based on the parameters set for **ZoneIndex0** (zero).

Parameters: None

Syntax: Nestor1.CreateZone

Return Values: True on success
False on error

Data Type: Boolean

See Also:

Comments:

DebugPath Property

Definition: Specifies a path and filename for a log file of the operation of the Nestor engine.

Data Type:

Syntax: Nestor1.DebugPath = "c:\imgbasic\nestor.log"

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

DeleteWordFromDictionary Method

Definition: Deletes a single word from an existing dictionary; valid only for normal dictionaries.

Parameters:	<i>dict</i>	Path and file name of the dictionary file
	<i>word</i>	String of the word to delete from the dictionary; no spaces are allowed
Syntax:	Nestor1.DeleteWordFromDictionary <i>dict, word</i>	
Return Values:	None	
Data Type:	Long	
See Also:		
Comments:		

DeleteZone Method

Definition:	Deletes the zone at the current ZoneIndex	
Parameters:	None	
Syntax:	Nestor1.DeleteZone	
Return Values:	None	
Data Type:	Long	
See Also:		
Comments:		

Error Event

Definition:	Occurs for each error internal to the control.	
Parameters:	Number	A long error code that identifies the error
	Description	Descriptive string of the error
	SCode	A composite long number indicating the severity of the error, the facility code, the origin of the error, and the native error code
	Source	Descriptive string of the source of the error
	HelpFile	Suggested help file name that should have a detailed explanation of the error
	HelpContext	Context ID of the appropriate topic in the help file named above
	CancelDisplay	If set to True during this event, the standard error dialog will not be displayed

GetZoneParam Method

Definition:		
Parameters:	<i>parameter</i>	Name of the parameter whose value is to be retrieved

Syntax: `strParamValue = Nestor1.GetZoneParam(parameter)`
Return Values: Current value of the specified parameter in the ZDF buffer
Data Type: String
See Also: SetZoneParam Method
Comments:

ImageDataSource Property

Definition: Specifies the ImageBASIC control that will supply image data for ICR by this control. Valid only when **InputFrom** is set to *0--ImageDataSource*

Data Type: String

Syntax: `Nestor1.ImageDataSource = TMSDispl.Link`

Design Access: Read/Write

Runtime Access: Read/Write

See Also: InputFrom Property, RegionSource Property

Comments: **ImageDataSource** must specify an ImageBASIC control that can supply image data if ICR is to be attempted and the **InputFrom** property is *0--ImageDataSource*

When an ImageBASIC control is added to a Form at design time, the **ImageDataSource** property is automatically populated with a source ImageBASIC control if one already exists on the Form.

At runtime, this property may be set to the **Link** value of any ImageBASIC control that is an image source; for example,
`Nestor1.ImageDataSource = TMSDispl.Link`

InputFileName Property

Definition: Specifies the path and filename to an image file to process; valid for both form definition and recognition; valid only when the **InputFrom** property is set to *1--File*.

Data Type:

Syntax: `Nestor1.InputFileName = "c:\images\form0001.tif"`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

InputFrom Property

Definition: Specifies the source of images for processing -- from memory or from a disk file.

Data Type:

Syntax: `Nestor1.InputFrom = intOption`

Possible Values: 0 ImageDataSource
1 File

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

LoadMasterImage Method

Definition: Accepts the image specified in the **InputFrom** property and analyzes the registration and OMR zones.

Parameters: *file* Path and file name of the master image to analyze
snippet Boolean

Syntax: `Nestor1.LoadMasterImage file, snippet`

Return Values: True
False

Data Type: Boolean

See Also:

Comments:

LoadZDFFile Method

Definition: Loads the specified ZDF file into memory.

Parameters: None

Syntax: `Nestor1.LoadZDFFile`

Return Values: None

Data Type: Long

See Also: SaveZDFFile Method

Comments:

OCRPage Method

Definition: Initiates a recognition attempt on the current image page.

Parameters: None

Return Values: None

Data Type: Long

Syntax: Nestor1.OCRPage

See Also: ZDFInputFrom Property, InputFrom Property, OutputTo Property

Comments: The image on which recognition will be performed is specified as follows:

- 1) If the **InputFrom** property is set to *0--ImageDataSource*, the image for ICR will be read from the ImageBASIC control specified in the **ImageDataSource** property.
If the **InputFrom** property is set to *1--File*, the image for ICR will be read from the file(s) specified in the **InputFileName** property.
- 2) If the **ZDFInputFrom** property is *0--ZDF Buffer*, the current zone definitions in the ZDF buffer will be used to process the image.
If the **ZDFInputFrom** property is *1--ZDF File*, the ZDF file specified in the **ZDFInputFileName** property will be read to supply the processing parameters.

OptAutoNoiseDetect Property

Definition: If True,

Data Type:

Syntax: Nestor1.OptAutoNoiseDetect = True

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OptDropOutInkUsed Property

Definition: Should be set to True during both form/zone definition and recognition if the forms that are being recognized will be printed in dropout ink. If False, the engine will assume that regular ink was used in the form.

Data Type: Boolean

Syntax: Nestor1.OptDropOutInkUsed = True

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OptFormRemovalTemplateFile Property

Definition: Specifies the path and file name of the file to use for Form Removal during recognition. If null, Form Removal will not be performed.

Data Type:

Syntax: `Nestor1.OptFormRemovalTemplateFile = strFilename`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OptImageResolutionDPI Property

Definition: Specifies the original resolution of the input image. Must be set if using the Characters Per Inch specification in Data Zones.

Data Type:

Syntax: `Nestor1.OptImageResolutionDPI = 300`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OptMaxGapInLongLine Property

Definition: Specifies the maximum gap in a long line which will be ignored when determining how long each line is.

Data Type: Long

Syntax: `Nestor1.OptMaxGapInLongLine = 3`

Design Access: Read/Write

Runtime Access: Read/Write

See Also: `OptMinLongLineLength` Property

Comments:

OptMaxSpeckArea Property

Definition: Specifies the largest clump of pixels that can be considered a speck and removed. If 0 (zero), speck removal will not be performed.

Data Type:

Syntax: `Nestor1.OptMaxSpeckArea = 20`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: Any clump of pixels that is smaller than the area defined by the **OptMaxSpeckArea**, **OptMaxSpeckHeight** and **OptMaxSpeckWidth** properties will be removed in the preprocessing step.

OptMaxSpeckHeight Property

Definition: Specifies the maximum height, in image pixels, of a clump of pixels that can be considered noise.

Data Type:

Syntax: `Nestor1.OptMaxSpeckHeight = 4`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: Any clump of pixels that is smaller than the area defined by the **OptMaxSpeckArea**, **OptMaxSpeckHeight** and **OptMaxSpeckWidth** properties will be removed in the preprocessing step.

OptMaxSpeckWidth Property

Definition: Specifies the maximum width, in image pixels, of a clump of pixels that can be considered noise.

Data Type:

Syntax: `Nestor1.OptMaxSpeckWidth = 4`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: Any clump of pixels that is smaller than the area defined by the **OptMaxSpeckArea**, **OptMaxSpeckHeight** and **OptMaxSpeckWidth** properties will be removed in the preprocessing step.

OptMinLineHeightBeforeSplit Property

Definition: Specifies the minimum height of a line which will have line splitting attempted.

Data Type: Long

Syntax: `Nestor1.OptMinLineHeightBeforeSplit = intHeight`

Possible Values: Integers, 0 to 300

Default: 75

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: Line splitting works in the multiple line segmentation mode and attempts to make a horizontal split between 2 or more lines which have been incorrectly connected due to the excursion of one or more characters from one line to the other. For example, the bottom of a “y” dropping into some characters in the line below.

OptMinLongLineLength Property

Definition: Specifies the minimum length, in image pixels, of a horizontal or vertical line for Line Removal to be performed.

Data Type: Long

Syntax: `Nestor1.OptLongLineLength = 600`

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: The value of this property should be greater than the maximum expected size (in pixels) of the largest character in the image. The default value is 75 pixels.

Line Removal must be activated through the

OptRemoveLongLines property.

OptMinOMRFilledForChecked Property

Definition: Specifies the percentage of an OMR box that must be filled for the box to be considered checked.

Data Type:

Syntax: `Nestor1.OptMinOMRFilledForChecked = intPercent`

Possible Values: Integers, typically less than 10

Design Access: Read/Write

Runtime Access: Read/Write

See Also: **OptOMRUseWeightedCenter** Property

Comments: The **OptOMRUseWeightedCenter** property must be False for this setting to be used. The default value is 5.

OptOMRUseWeightedCenter Property

Definition: If True, the recognition engine will determine if an OMR box is checked by searching the center of the check box. If False, the overall percentage of the fill will determine if the box is considered checked.

Data Type: Boolean

Syntax: Nestor1.OptOMRUseWeightedCenter = True

Possible Values: True
False

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: This setting should be True if the amount that is filled is more towards the center of the box (“weighted” towards the box’s center), such as when a “X” is used.

OptRemoveGraphics Property

Definition: If True, large blobs on the image that are considered graphics will be removed. If False, the graphics will be converted to text.

Data Type: Boolean

Syntax: Nestor1.OptRemoveGraphics = True

Possible Values: True
False

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OptRemoveLongLines Property

Definition: If True, horizontal and vertical lines will be removed from the input image before recognition. If False, lines will not be removed.

Data Type: Boolean

Syntax: Nestor1.OptRemoveLongLines = True

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: The determination of which lines to remove is based on the **OptMinLongLineLength** and **OptMaxGapInLongLine** properties.
Line Removal should be disabled when Form Removal is enabled.

OptUncertainCharMaxHeight Property

Definition: Specifies the maximum percentage of the height of the zone that a character can be in order to be considered a character and possibly marked as uncertain.

Data Type: Long

Syntax: `Nestor1.OptUncertainCharMaxHeight = intHeight`

Possible Values: Integers, 0 to 1000
Default: 350

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: For example, a value of 350 means that the character can be up to 3.5 times as high as the zone.

Note: The OptUncertainChar... properties deal with the situation where the segmentation process sometimes overlooks characters. There are 2 typical examples of this behavior. The first is when a character from one line drops down into another line—such as a lower case “g” in the first line touching the top of an “l” in the second line. Without these settings, either the “g” and the “l” would not appear in the recognition results, or they would be combined and represented by the wrong character (usually with a high confidence value) in either the first or the second line. The other typical example of overlooked characters is best explained by a “y” that is written with a tiny top and a long tail that extends beyond the current line or zone. In this case, the “y” would not appear in the recognition results. In both examples, nothing is presented to the verification operator to correct.

For a single word or single line zone, the height percentage calculated below is relative to the zone. For a multiple line zone, the height percentage is relative to the height of each line found by the segmentation code.

The noise width and noise height settings also interact with these settings. If something matches the noise criteria, there will be no attempt to flag it as an uncertainty character.

With the following settings, even though the recognition process does not know what the correct characters are, they are at least identified with an uncertainty character on a particular line within a zone and with a low confidence. This guarantees that they will be flagged as uncertain and the verification operator will be given a chance to correct them.

These settings are intended for advanced users and only need to be modified for very special cases.

OptUncertainCharMinOverlap Property

Definition:	Specifies the percentage of the height of the zone or line (for multiple line zones) that a character must be before it can be considered a character.
Data Type:	Long
Syntax:	<code>Nestor1.OptUncertainCharMinOverlap = <i>intOverlap</i></code>
Possible Values:	Integers, 0 to 100 Default: 30
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	
Comments:	See Note on OptUncertainCharMaxHeight Property.

OptUncertainCharReplacement Property

Definition:	Specifies the character to place in the output string for each character whose confidence falls below the minimum threshold.
Data Type:	String
Syntax:	<code>Nestor1.OptUncertainCharReplacement = "?"</code>
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	OptUncertainCharMaxHeight Property
Comments:	The minimum acceptable confidence for characters in the return string is set using the Data Zone keyword <code>CONFIDENCE_THRESHOLD</code> . See Note on OptUncertainCharMaxHeight Property.

OptUncertainCharSplitOverlap Property

Definition:	Specifies the percentage of a given line or zone that a character must be before it will be considered for splitting into two characters.
Data Type:	Long
Syntax:	<code>Nestor1.OptUncertainCharSplitOverlap = <i>intSplit</i></code>
Possible Values:	Integers, 0 to 100 Default: 40
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	

Comments: See Note on **OptUncertainCharMaxHeight** Property.

OptUseSmartNoiseClassification Property

Definition: If True, the recognition engine will attempt to match noise to characters of which it might be a fragment. If False, all image features which are determined to be noise will be ignored.

Data Type: Boolean

Syntax: `Nestor1.OptUseSmartNoiseClassification = intBool`

Possible Values: True
False

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: This property should be set to True when the image has many fragmented characters whose interpretation may be degraded by disregarding all noise in the image. This property works in conjunction with the **OptNoiseMaxSpeckArea** and related properties to determine what is noise and what is character fragments.

OptUseSpecialPunctuationMem Property

Definition: If True, the engine will attempt to identify and return punctuation in the Data Zones. If False, punctuation will normally be disregarded and treated as noise.

Data Type: Boolean

Syntax: `Nestor1.OptUseSpecialPunctuationMemory = intBool`

Possible Values: True
False

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments:

OutputFileName Property

Definition: Specifies the path and file name of the file to write with the recognition results.

Data Type: String

Syntax: `Nestor1.OutputFileName = "c:\capture\form01.txt"`

Design Access: Read/Write
Runtime Access: Read/Write
See Also: OutputToProperty, OutputFileAppend Property
Comments: This property is valid only if the **OutputTo** property specifies *!-File*. The **OutputFileAppend** property controls whether the text is appended to the file or replaces any existing contents.

OutputFormat Property

Definition: Specifies the format of the recognition output string.
Data Type: Enumerated
Syntax: Nestor1.OutputFormat = *intOption*
Design Access: Read/Write
Runtime Access: Read/Write
See Also: OutputFileNameProperty, OutputToProperty
Comments: The valid enumerated values for this property are shown in the list below.
Note: All output formats numbered six and higher must be written to file instead of to a Visual Basic string variable or property. This is because all of these formats are binary output rather than text, and Visual Basic string properties and variables cannot accept this data.
10 ASCII Database
13 ASCII Standard
71 ZRF
72 Specified in ZDF
Analogous to the ZDF global keyword **RESULTS_FORMAT**.

OutputTo Property

Definition: Specifies the destination for ICR results.
Data Type: Integer (Enumerated)
Syntax: Nestor1.OutputTo = *intOption*
Design Access: Read/Write
Runtime Access: Read/Write
See Also: InputFromProperty, OutputFileNameProperty
Comments: Valid options for this property are as follows:
0 String
1 File
0-String causes the recognition results to be reported in the **Result** property.

!--File causes the recognition results to be written to the file specified in the **OutputFileName** property.

Progress Event

Definition:	Occurs frequently during recognition and provides continued updates that the engine is still operating; allows the option to cancel recognition.				
Parameters:	<table><tr><td>Percent</td><td>Percentage completion of current ICR attempt</td></tr><tr><td>Cancel</td><td>If set to True, this ICR attempt is canceled</td></tr></table>	Percent	Percentage completion of current ICR attempt	Cancel	If set to True, this ICR attempt is canceled
Percent	Percentage completion of current ICR attempt				
Cancel	If set to True, this ICR attempt is canceled				
Comments:	<p>The only way to cancel a recognition attempt is to set the <i>Cancel</i> parameter to this event to True during the event. The cancellation may be passed to the event as a global variable set in a button click or through some other method, as shown here:</p> <pre>Private Sub cmdCancel_Click() gnStop = True End Sub Private Sub Nestor1_Progress(ByVal Percent As Integer, Cancel As Boolean) DoEvents Cancel = gnStop End Sub</pre>				

Result Property

Definition:	If OutputTo specifies <i>!--String</i> , reports the final recognition results after a successful ICR attempt.
Data Type:	String
Syntax:	<i>sResults</i> = Nestor1.Result
Design Access:	Not Available
Runtime Access:	Read-only
See Also:	OutputToProperty
Comments:	<p>The string that is reported in this property will be the final recognition results. The string will be formatted as specified in the OutputFormat property.</p> <p>Valid only when OutputTo is set to <i>!--String</i>. After recognition is complete, this property will report the total number of regions that were processed. When the ResultIndex property is set to any integer value between 1 (one) and ResultCount inclusive, the Result property is populated with the output string for the specified region.</p>

ResultCharInfo Property

Definition:

Data Type: Boolean

Syntax: `strResults = Nestor1.Result`

Design Access: Not Available

Runtime Access: Read-only

See Also:

Comments:

ResultCount Property

Definition: After a successful recognition attempt, reports the total number of zones that were recognized. This value should match the number of Data Zones and OMR Zones for the image.

Data Type: Long

Syntax: `intZoneCount = Nestor1.ResultCount`

Design Access: Not Available

Runtime Access: Read-only

See Also: ResultIndex Property

Comments:

ResultIndex Property

Definition: After a successful recognition attempt, specifies which zone's recognition results will be reported.

Data Type: Long

Syntax: `Nestor1.Result = intIndex`

Design Access: Not Available

Runtime Access: Read/Write

See Also:

Comments: Immediately after a successful recognition, the **ResultIndex** property may be set to any value from 1 (one) to **ResultCount**. Setting **ResultIndex** to a valid value will update the following properties with information pertaining to a single recognition zone:

Result	Reports the recognition string for the selected zone
ResultCharInfo	If True, the Result property will show all character information; if

	False, only the recognition string will be available
ResultZoneName	Reports the name assigned to the Data or OMR zone whose results are now being reported
ResultZoneCharCount	Reports the number of characters that were recognized in the current zone
ResultZoneLineCount	Reports the number of lines that were recognized in the current zone

ResultZoneCharCount Property

Definition:	After a successful recognition attempt, reports the number of characters that were recognized in the currently reported zone.
Data Type:	Long
Syntax:	<code>intCharCount = Nestor1.ResultZoneCharCount</code>
Design Access:	Not Available
Runtime Access:	Read-only
See Also:	ResultIndex Property
Comments:	Immediately after a successful recognition, the ResultIndex property may be set to any value from 1 (one) to ResultCount . Setting ResultIndex to a valid value will update the following properties with information pertaining to a single recognition zone:

Result	Reports the recognition string for the selected zone
ResultCharInfo	If True, the Result property will show all character information; if False, only the recognition string will be available
ResultZoneName	Reports the name assigned to the Data or OMR zone whose results are now being reported
ResultZoneCharCount	Reports the number of characters that were recognized in the current zone
ResultZoneLineCount	Reports the number of lines that were recognized in the current zone

ResultZoneLineCount Property

Definition: After a successful recognition attempt, reports the number of lines that were identified in the currently reported zone.

Data Type: Long

Syntax: `intLineCount = Nestor1.ResultZoneLineCount`

Design Access: Not Available

Runtime Access: Read-only

See Also:

Comments: Immediately after a successful recognition, the **ResultIndex** property may be set to any value from 1 (one) to **ResultCount**. Setting **ResultIndex** to a valid value will update the following properties with information pertaining to a single recognition zone:

Result Reports the recognition string for the selected zone

ResultCharInfo If True, the Result property will show all character information; if False, only the recognition string will be available

ResultZoneName Reports the name assigned to the Data or OMR zone whose results are now being reported

ResultZoneCharCount Reports the number of characters that were recognized in the current zone

ResultZoneLineCount Reports the number of lines that were recognized in the current zone

ResultZoneName Property

Definition: After a successful recognition attempt, reports the name assigned to the currently reported zone.

Data Type: String

Syntax: `strZoneName = Nestor1.ResultZoneName`

Design Access: Not Available

Runtime Access:	Read-only	
See Also:	ResultIndex Property	
Comments:	Immediately after a successful recognition, the ResultIndex property may be set to any value from 1 (one) to ResultCount . Setting ResultIndex to a valid value will update the following properties with information pertaining to a single recognition zone:	
	Result	Reports the recognition string for the selected zone
	ResultCharInfo	If True, the Result property will show all character information; if False, only the recognition string will be available
	ResultZoneName	Reports the name assigned to the Data or OMR zone whose results are now being reported
	ResultZoneCharCount	Reports the number of characters that were recognized in the current zone
	ResultZoneLineCount	Reports the number of lines that were recognized in the current zone

SaveZDFFile Method

Definition:	Saves the information in the current ZDF buffer to the file name specified in the ZDFInputIFFileName property.
Parameters:	None
Syntax:	Nestor1.SaveZDFFile
Return Values:	None
Data Type:	Boolean
See Also:	ZDFInputIFFileName Property
Comments:	

SetZoneParam Method

Definition:	Sets a processing parameter for the current zone; the parameter is stored in the current ZDF buffer.	
Parameters:	<i>param</i>	String name of the parameter to set
	<i>value</i>	Value to be assigned to the specified parameter
Syntax:	Nestor1.SetZoneParam <i>param, value</i>	

Return Values: True on success
False on Error

Data Type: Boolean

See Also: ZoneIndex Property, 'Appendix B : File Format' on page 103

Comments:

ShutdownEngine Method

Definition: Closes all currently open instances of the Nestor engine.

Parameters: None

Syntax: Nestor1.ShutdownEngine

Return Values: None

Data Type: String

See Also:

Comments:

ZDFInputFileName Property

Definition: Specifies the name of the ZDF file to use for recognition and for loading and saving zone definition information.

Data Type: String

Syntax: Nestor1.ZDFInputFileName = *strFilename*

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: The ZDF file specified in this property will be used for recognition only when the **ZDFInputFrom** property is *1--ZDF File*.

When saving the contents of the current ZDF buffer to file using the **SaveZDFFile** method, this property should specify the path and file name of the file to write. If the specified file exists, it will be overwritten. The file should be assigned an extension of *.zdf*.

When loading the contents of an existing ZDF file into the ZDF buffer using the **LoadZDFFile** method, this property should include the full path and file name of the file to load.

ZDFInputFrom Property

Definition:	When recognition is performed, this property specifies whether the zone definitions will be read from a ZDF file or from the ZDF buffer.
Data Type:	Integer (Enumerated)
Syntax:	<code>Nestor1.ZDFInputFrom = <i>intOption</i></code>
Possible Values:	0 ZDF Buffer 1 ZDF File
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	
Comments:	If this property is set to <i>1--ZDF File</i> , the ZDFInputFileName property must specify the ZDF file to read.

ZoneBottom Property

Definition:	Specifies the image pixel coordinate of the bottom edge of the current zone.
Data Type:	Long
Syntax:	<code>Nestor1.ZoneBottom = <i>lngPosition</i></code>
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	ZoneIndex Property
Comments:	In a ZDF, the coordinates of all zone types are defined by the following properties. All values in these properties are based on image pixels, where (0,0) is the top, left corner of the image. ZoneBottom ZoneLeft ZoneRight ZoneTop The current zone is selected through the ZoneIndex or ZoneName property. When defining a new zone ZoneIndex must be 0 (zero).

ZoneCount Property

Definition:	Reports the total number of zones that are defined in the current ZDF buffer.
Data Type:	Integer
Syntax:	<code><i>intCount</i> = Nestor1.ZoneCount</code>
Design Access:	Read/Write
Runtime Access:	Read/Write

See Also: ZoneIndex Property, LoadZDFFile Method

Comments:

ZoneIndex Property

Definition: Specifies the current zone.

Data Type: Integer

Syntax: Nestor1.ZoneIndex = *intIndex*

Design Access: Read/Write

Runtime Access: Read/Write

See Also: CreateZone Method

Comments: The ZoneIndex property may be set to any integer value from 1 (one) to ZoneCount. When set to a valid value, the following descriptive properties for the specified zone will be refreshed:

ZoneBottom

ZoneLeft

ZoneName

ZoneRight

ZoneTop

ZoneType

In addition, the **SetZoneParam** and **GetZoneParam** methods will operate on the options for the specified zone.

Note: When creating a new zone using the **CreateZone** method, ZoneIndex must be set to 0 (zero) before defining the same properties listed above. After these descriptive properties are set, the **CreateZone** method may be executed.

ZoneLeft Property

Definition: Specifies the image pixel coordinate of the left edge of the current zone.

Data Type: Long

Syntax: Nestor1.ZoneLeft = *lngPosition*

Design Access: Read/Write

Runtime Access: Read/Write

See Also:

Comments: In a ZDF, the coordinates of all zone types are defined by the following properties. All values in these properties are based on image pixels, where (0,0) is the top, left corner of the image.

ZoneBottom

ZoneLeft

ZoneRight

ZoneTop

The current zone is selected through the **ZoneIndex** or **ZoneName** property. When defining a new zone **ZoneIndex** must be 0 (zero).

ZoneName Property

Definition:	Specifies the name of the current zone. May be set to any valid value when defining a new zone. When set to a defined zone name, the specified zone will be made the current zone.
Data Type:	String
Syntax:	<code>Nestor1.ZoneName = strZName</code>
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	ZoneIndex Property
Comments:	

ZoneRight Property

Definition:	Specifies the image pixel coordinate of the right edge of the current zone.
Data Type:	Long
Syntax:	<code>Nestor1.ZoneRight = lngPosition</code>
Design Access:	Read/Write
Runtime Access:	Read/Write
See Also:	
Comments:	In a ZDF, the coordinates of all zone types are defined by the following properties. All values in these properties are based on image pixels, where (0,0) is the top, left corner of the image. ZoneBottom ZoneLeft ZoneRight ZoneTop The current zone is selected through the ZoneIndex or ZoneName property. When defining a new zone ZoneIndex must be 0 (zero).

ZoneTop Property

Definition:	Specifies the image pixel coordinate of the top edge of the current zone.
Data Type:	Long
Syntax:	<code>Nestor1.ZoneTop = lngZoneTop</code>

Design Access: Read/Write
Runtime Access: Read/Write
See Also:
Comments: In a ZDF, the coordinates of all zone types are defined by the following properties. All values in these properties are based on image pixels, where (0,0) is the top, left corner of the image.
 ZoneBottom
 ZoneLeft
 ZoneRight
 ZoneTop
 The current zone is selected through the **ZoneIndex** or **ZoneName** property. When defining a new zone **ZoneIndex** must be 0 (zero).

ZoneType Property

Definition: Specifies the type-- Data, OMR or Recognition -- of the current zone.
Data Type: Integer (Enumerated)
Syntax: Nestor1.ZoneType = *intOption*
Possible Values: 0 Registration Zone
 1 OMR Zone
 2 Data Zone
Design Access: Read/Write
Runtime Access: Read/Write
See Also: ZoneIndex Property
Comments: The zone parameters that may be set vary according to the type of zone under consideration. All zone types share the following properties:
 ZoneBottom
 ZoneLeft
 ZoneName
 ZoneRight
 ZoneTop

Appendix A : Error Codes

Error Reporting

Runtime Error Codes

NestorReader Engine Error Codes

The following error codes indicate that NestorReader OT API has run out of memory. If one of these errors occurs, no further processing may be done until NestorReader OT API has been re-started.

- 0001 "Memory allocation failed (malloc)"
- 0002 "Memory reallocation failed"
- 0003 "Memory allocation failed (calloc)"

The following error codes have various causes. Please note the complete information in your application so that the exact location of the error's occurrence may be reported to Nestor. Messages identified **WARNINGS** allow continued use of the system; the effects of other errors may vary.

- 5561** "ERROR: opening recognizer memory file."
This error indicates that the recognition or segmentation memory file named in a **DEFINE_CONSTRAINT** line in a ZDF file could not be opened. Check the appropriate ZDF file to see that it correctly indicates where the **ME1** or **ME2** file provided by NCS resides.
- 5563** "ERROR: reading recognizer memory file."
This error indicates that the recognition or segmentation memory file may be corrupted. Reload the desired **ME1** or **ME2** file from the NCS distribution media. If the error still occurs, contact NCS.
- 5568** "ERROR: using fseek on disk file."

	This is a system or disk error. NestorReader OT API was unable to seek to a desired location on a disk file. This could be caused by a corrupted copy of DOS in RAM or by a hard disk error; rebooting may help.
5569	<p>"ERROR: using ftell on disk file."</p> <p>NestorReader OT API was unable to determine the current location in a disk file. This could be caused by a corrupted copy of DOS in RAM or by a hard disk error; rebooting may help.</p>
5740	<p>"ERROR: reading from memory file."</p> <p>This error is similar to error 5563.</p>
6003	<p>"ERROR: Failed to open file."</p> <p>This is a generic error which may occur while opening a file.</p>
6007	<p>"ERROR: While reading from file."</p> <p>This is a generic error which may occur while reading a file.</p>
6008	<p>"ERROR: While writing to file."</p> <p>This is a generic error which may occur while writing to a file.</p>
6009	<p>"WARNING: Character image too large to process."</p> <p>One or more of the characters presented for recognition was physically too large for NestorReader OT API to process after segmentation. Check the image(s) concerned for three or more connected characters. The image may be rescanned with a lighter setting so that characters are separated from light smudges and other image defects.</p>
6012	<p>"ERROR: One of the dimensions of an image was 0."</p> <p>This error indicates that NestorReader OT API was unable to read the TIFF image format. It may be that the image has been stored in a non-standard TIFF format.</p>
6020	<p>"ERROR: No memory loaded for specified constraint."</p> <p>Recognition was attempted on a zone or character with a logical constraint which did not specify a valid ME1 or ME2 file. Ensure that the memories specified by the DEFINE_CONSTRAINT keyword in the ZDF file have correct path information.</p>
6025	<p>"ERROR: No zones found in ZDF file."</p> <p>The ZDF file exists, but is empty.</p>

- 6026** "ERROR: Unrecognized zone format in ZDF file."
The ZDF file exists, and has information in it, but at least one of the zone definitions is not of the proper format. Check the ZDF file to ensure it is a valid ZDF file, and ensure that it has not been corrupted.
- 6027** "ERROR: Unable to open REC file."
The recognition results (**REC**) file could not be created for output. This may happen if the disk is read-only (such as a CD-ROM), or if the disk is full.
- 6031** "ERROR: Unable to open temporary file."
An attempt was made to open a temporary file, but the attempt failed. This error may occur if the image resides on a CD-ROM, or if the disk is full.
- 6033** "ERROR: Unable to seek in image file."
This is a system or disk error. NestorReader OT API was unable to seek to a desired location in an image file. This could be caused by a corrupted copy of DOS in RAM or by a hard disk error; rebooting may help.
- 6034** "ERROR: Unable to read from image file."
This is a system or disk error. NestorReader OT API was unable to read from an open image file. This could be caused by a corrupted copy of DOS in RAM or by a hard disk error; rebooting may help.
- 6035** "ERROR: Unable to determine position in image file."
NestorReader OT API was unable to determine the current location in an image file. This could be caused by a corrupted copy of DOS in RAM or by a hard disk error; rebooting may help.
- 6036** "ERROR: Specified memory is not supported by NestorReader."
The most likely cause for this error is the specification of a non-NestorReader file with a **DEFINE_CONSTRAINT** keyword in a ZDF file. Make sure that the specified file name is a NestorReader-supplied **ME1** or **ME2** file.
- 6037** "ERROR: Too many characters in zone."
The number of characters in a zone exceeded the internal capacity of NestorReader OT API. Break up the zone into multiple zones if possible.

6038	<p>"WARNING: Could not open .SPC file."</p> <p>See error #6023.</p>
6039	<p>"WARNING: Could not read zone definitions from ZDF file."</p> <p>See error #6026.</p>
6045	<p>"WARNING: Character too large; not processed."</p> <p>Segmentation of the zone produced a character larger than 21000 pixels. The recognition of other characters will continue, but the results for the character in question will be marked with a confidence value of 0.</p>
6046	<p>"WARNING: Bad file name; cannot open file."</p> <p>The most likely reason for this error is an incorrect pathname in the .SPC file. Check the .SPC file for correct pathnames for the correct ZDF file and the image files.</p>
6047	<p>"WARNING: Blob larger than max allowable..."</p> <p>Touching characters or graphics in the zone were too large. The area affected will be segmented, but no recognition will be performed on it.</p>
6049	<p>"ERROR: Selected image area is too large and complex."</p> <p>The zone being processed has either too many characters in it, or has many separate blobs in it. This zone cannot be processed with the current zone definitions, but recognition will be attempted on other zones if present.</p>
6052	<p>"WARNING: Image file smaller than zone definition."</p> <p>The zones specified in the ZDF file do not fall within the bounds of the image being processed. Check to ensure that the correct ZDF file is being used for the image files named, or that the DATA_ZONE coordinates being set in your application are correct.</p>
6054	<p>"WARNING: Function argument out of range."</p> <p>An arithmetic function received an argument out of the acceptable range; processing will continue.</p>
6056	<p>"ERROR: Memory constraint has invalid value."</p> <p>The constraint specified in the ZDF file may be negative</p>
6057	<p>"ERROR: Memory with duplicate constraint value."</p> <p>An attempt was made to read in a memory with a constraint value that duplicates a constraint value for a previously-read</p>

memory. Check the constraint values of memories being read.

- 6065** "ERROR: Invalid CCITT Group 3, 2D compression mode."
The **.TIF** file's header specified a compression mode that is not supported by NestorReader OT API. Try to generate the **.TIF** file in a different compression mode.
- 6069** "ERROR: Error reading VB file"
The **.ME1** or **.ME2** being read may be corrupted. Reinstall the memory from the NestorReader OT API installation diskettes, CD ROM, or tape.
- 6070** "ERROR: Attempt to install dictionary over existing dictionary."
An attempt was made to read a dictionary with the same number as a previous dictionary.
- 6071** "ERROR: Error opening dictionary."
The file name specified for the dictionary could not be opened. Check to ensure that the dictionary file name being used (either in application code or in a Data zone specification) has the correct pathname.
- 6073** "ERROR: Invalid run length code in tiff file"
The **.TIF** file's compressed image contains an invalid value. The file may be corrupted, or it may have been generated by software that does not follow the TIFF Rev. 5.0 specification.
- 6084** "ERROR: File is physically too large to process."
An image file is too large (as measured by bytes on the disk) to be processed. If feasible, the file may be scanned piecewise or separated into parts by a graphics program.
- 6091** "ERROR: opening recognition memory file."
A **.ME1** or **.ME2** file does not exist at the pathname specified in a logical constraint definition.
- 6093** "ERROR: Writing results file."
See error **6027**.
- 6099** "ERROR: Dictionary format is incorrect"
- 6100** "ERROR: Dictionary version is incorrect."
The dictionary file's format is not recognized. Check to ensure that the file is either a dictionary file distributed by

- NCS or is the output of one of NestorReader OT API dictionary maintenance functions.
- 6101** "WARNING: Word too long for context checking."
One of the words in the zone contains more letters than the maximum number expected by NestorReader OT API. One of the words in the zone may contain cursive handwriting. Trying a different segmentation type may alleviate this for some zones.
- 6102** "WARNING: Specified dictionary not loaded."
No dictionary has been loaded for the dictionary index indicated. Be sure all dictionaries were load properly. When specifying dictionary numbers using the **DICTIONARY** keyword in a ZDF file, be sure they correspond to these dictionaries.
- 6104** "WARNING: No results file name supplied to NR_WriteResults. No results file was written."
This error happens when using an image in RAM with results type **FILE_RESULTS**. In this cases, the user must specify a proper filename for the results to be written.
- 6110** "Could not open dictionary file for output."
The dictionary utilities could not open the specified dictionary file. This may happen if the disk is read-only (such as a CD-ROM), or if the disk is full.
- 6111** "Could not open text file for dictionary operations."
The dictionary utilities could not open the specified dictionary file. Be sure that the dictionary file specified exists and has a valid filename.
- 6124** "ERROR: Unable to open TXT file"
The results directory specified for output may not exist, or there may be a write-protected file of the same name already in existence.
- 6126** "ERROR: invalid constraint label"
A character that was not a digit or an upper or lower case letter was used in defining a logical constraint or with a Data Zone's **CONSTRAINT** keyword in a ZDF file.
- 6200** "WARNING: No segmentation type specified."
The **SEGMENTATION** keyword was specified in a ZDF file with no additional parameter.

- 6201** "WARNING: Unknown segmentation type specified."
The **SEGMENTATION** keyword was specified in a ZDF file with an invalid parameter.
- 6202** "WARNING: Specify ON or OFF with **DICTIONARY** keyword."
The **DICTIONARY** keyword was specified in a ZDF file with no following parameters.
- 6203** "WARNING: Invalid value used with **DICTIONARY ON** keywords."
The modifier which must be specified with the **DICTIONARY** keyword was not recognized.
- 6204** "WARNING: Invalid keyword used with **DICTIONARY** keyword."
The dictionary mode keyword which must be specified with the **DICTIONARY** keyword was missing or invalid. See Appendix A for acceptable values.
- 6205** "WARNING: No action specified with **CONTEXT** keyword."
The **CONTEXT** keyword was specified in a ZDF file with no additional parameter. This keyword expects the parameter to be **ON** or **OFF**.
- 6206** "WARNING: Invalid keyword used with **CONTEXT** keyword."
The **CONTEXT** keyword was specified in a ZDF file with an invalid parameter. This keyword expects the parameter to be **ON** or **OFF**.
- 6209** "WARNING: No action specified with **CONSTRAINT** keyword."
The **CONSTRAINT** keyword was specified in a ZDF file with no following parameter.
- 6210** "WARNING: Invalid keyword used with **CONSTRAINT** keyword."
The **CONSTRAINT** keyword was specified in a ZDF file with an invalid parameter.
- 6211** "WARNING: No value specified with **ALTERNATE_IDS** keyword."

The **ALTERNATE_IDS** keyword was specified in a ZDF file with no following parameter. This keyword expects a value 0, 1 or 2 as a parameter.

6212 "ERROR: Unexpected EOF encountered in the ZDF file"

The start of a zone was found in a ZDF file with no corresponding end keyword. Check to ensure that the ZDF file is not corrupted.

6213 "WARNING: No value specified with TOP_LEFT_X keyword."

6214 "WARNING: No value specified with TOP_LEFT_Y keyword."

6215 "WARNING: No value specified with BOTTOM_RIGHT_X keyword."

6216 "WARNING: No value specified with BOTTOM_RIGHT_Y keyword."

One of these keywords was specified in a ZDF file with no additional parameter. These keywords expect a non-negative integer as a parameter.

6217 "WARNING: Invalid value used with TOP_LEFT_X keyword."

6218 "WARNING: Invalid value used with TOP_LEFT_Y keyword."

6219 "WARNING: Invalid value used with BOTTOM_RIGHT_X keyword."

6220 "WARNING: Invalid value used with BOTTOM_RIGHT_Y keyword."

One of these keywords was specified in a ZDF file with an invalid parameter. These keywords expects a non-negative integer as a parameter.

6221 "WARNING: No TOP_LEFT_X keyword found in zone definition."

6222 "WARNING: No TOP_LEFT_Y keyword found in zone definition."

6223 "WARNING: No BOTTOM_RIGHT_X keyword found in zone definition."

6224 "WARNING: No BOTTOM_RIGHT_Y keyword found in zone definition."

These keywords are required in a ZDF file for each zone definition.

- 6225** "WARNING: BOTTOM_RIGHT_X was less than TOP_LEFT_X value."
- 6226** "WARNING: BOTTOM_RIGHT_Y was less than TOP_LEFT_Y value."
- The values of the X and Y coordinates for the zone must be consistent with a valid image rectangle.
- 6227** "WARNING: No CONSTRAINT keyword found for zone definition."
- 6228** "WARNING: No SEGMENTATION keyword found for zone definition."
- 6229** "WARNING: No DICTIONARY keyword found for zone definition."
- 6231** "WARNING: No CONTEXT keyword found for zone definition."
- 6232** "WARNING: No ALTERNATE_IDS keyword found for zone definition."
- One of the settings needed for a complete zone definition was not defined in either the global settings or in the individual zone.
- 6233** "WARNING: Invalid value used with ALTERNATE_IDS keyword."
- The **ALTERNATE_IDS** keyword was specified in a ZDF file with an invalid parameter. This keyword expects a value 0, 1 or 2 as a parameter.
- 6234** "WARNING: Unable to read features for registration zone."
- The **FEATURES** keyword in the ZDF file was detected for a Registration Zone, but there were fewer than the expected number of features listed. The ZDF file may be corrupted.
- 6235** "WARNING: Unknown global keyword encountered."
- NestorReader OT API encountered a keyword in the global definition portion of the ZDF file which was either unknown or invalid as a global keyword.
- 6236** "WARNING: Unknown data zone keyword encountered."

- NestorReader OT API encountered a keyword in a ZDF file's data zone definition which was either unknown or invalid in the context of a data zone definition.
- 6237** "WARNING: Unknown registration zone keyword encountered."
- NestorReader OT API encountered a keyword in a ZDF file's registration zone definition which was either unknown or invalid in the context of a registration zone definition.
- 6238** "WARNING: Could not open ZDF file"
- The most likely reason for this error is an incorrect pathname in the **.SPC** file.
- 6239** "WARNING: No action specified with REGISTRATION keyword."
- 6240** "WARNING: No action specified with REMOVE_LINES keyword."
- One of these keywords was specified in a ZDF file with no additional parameter. These keywords expects the parameter to be **ON** or **OFF**.
- 6241** "WARNING: Invalid value used with REGISTRATION keyword."
- 6242** "WARNING: Invalid value used with REMOVE_LINES keyword."
- One of these keywords was specified in a ZDF file with an invalid parameter. These keywords may only be followed by **ON** or **OFF**.
- 6244** "WARNING: Invalid value used with ALTERNATE_IDS keyword."
- The **ALTERNATE_IDS** keyword was specified in a ZDF file with an invalid parameter. The only valid values for this keyword's parameter are 0, 1, and 2.
- 6248** "ERROR: Character format unknown in WORD_TYPE definition."
- This error is most likely caused by an invalid syntax for a **WORD_TYPE** definition when specifying a Word Syntax.
- 6249** "ERROR: Redefinition of a WORD_TYPE tag."
- NestorReader OT API has detected more than one **WORD_TYPE** definition with the same tag value. Be sure all Word Syntax entries have a unique tag.

- 6250** "ERROR: No character constraints in **WORD_TYPE** definition."
The character constraint value is a required field for a Word Syntax definition.
- 6251** "ERROR: Unknown keyword found in **WORD_TYPE** definition."
NestorReader OT API found a keyword during parsing a Word Syntax definition in a ZDF file which it was unable to interpret.
- 6252** "ERROR: Missing word type tag after **WORD_TYPE** keyword."
NestorReader OT API detected a **WORD_TYPE** definition in a ZDF file with the no tag value. Be sure all Word Syntax entries have a unique tag.
- 6253** "ERROR: Missing line type tag after **LINE_TYPE** keyword."
NestorReader OT API has detected a **LINE_TYPE** definition with the no tag value. Be sure all Line Syntax entries have a unique tag.
- 6254** "ERROR: Redefinition of a **LINE_TYPE** tag."
NestorReader OT API has detected more than one **LINE_TYPE** definition with the same tag value. Be sure all Line Syntax entries have a unique tag.
- 6255** "ERROR: Undefined **WORD_TYPE** used in **LINE_TYPE** definition."
A Line Syntax entry has been found which refers to an undefined Word Syntax tag. All Word Syntax entries used in a Line Syntax definition must be specified before they are referenced.
- 6256** "ERROR: Close paren found without matching open paren."
6257 "ERROR: Invalid character found following close paren."
6258 "ERROR: No close paren found to match open paren."
These errors are caused by an invalid syntax grammar when specifying a Word Syntax.
- 6259** "ERROR: Redefinition of zone grammar with **ZONE_DEFINITION**."
A second instance of the **ZONE_SYNTAX_START** keyword was found before an ending **ZONE_SYNTAX_END**

keyword. Each Zone Syntax definition must begin with **ZONE_SYNTAX_START** and end with **ZONE_SYNTAX_END**

6260 "ERROR: Unknown LINE_TYPE tag in ZONE_DEFINITION."

A Zone Syntax entry has been found which refers to an undefined Line Syntax tag. All Line Syntax entries must be specified before they can be referenced by a Zone Syntax definition.

6261 "WARNING: Missing close paren in ZONE_DEFINITION."

A Line Syntax tag was found in the Zone Syntax definition which had an opening parenthesis but no closing parenthesis.

6262 "WARNING: Segmentation does not match the syntax definition."

This warning makes the user aware that the number of words or lines found by the segmentation algorithms does not match the values that were specified by the Zone Syntax for this zone. Thus, the Zone Syntax cannot be applied to the zone in question.

6263 "ERROR: No value specified with IMAGE_RESOLUTION_DPI keyword."

The **IMAGE_RESOLUTION_DPI** keyword was specified in a ZDF file with no additional parameter.

6264 "ERROR: Invalid value used with IMAGE_RESOLUTION_DPI keyword."

The **IMAGE_RESOLUTION_DPI** keyword was specified in a ZDF file with an invalid parameter.

6265 "ERROR: No value specified with N_CHARS_PER_INCH keyword."

The **N_CHARS_PER_INCH** keyword was specified in a ZDF file with no additional parameter.

6266 "ERROR: Invalid value used with N_CHARS_PER_INCH keyword."

The **N_CHARS_PER_INCH** keyword was specified in a ZDF file with an invalid parameter.

6267 "WARNING: Multiple Segmentation options specified."

More than one of the Segmentation options was specified in a zone definition in the ZDF file.

- 6268** "ERROR: No value specified with **FIND_PUNCTUATION** keyword."
The **FIND_PUNCTUATION** keyword was specified in a ZDF file with no additional parameter. This keyword expects the parameter to be **ON** or **OFF**.
- 6269** "ERROR: Invalid value used with **FIND_PUNCTUATION** keyword."
The **FIND_PUNCTUATION** keyword was specified in a ZDF file with an invalid parameter. This keyword expects the parameter to be **ON** or **OFF**.
- 6270** "ERROR: Could not open ZRF file"
NestorReader OT API could not open the specified ZRF file. This may happen if the disk is read-only (such as a CD-ROM), or if the disk is full, or if the directory specified does not exist.
- 6273** "ERROR: Invalid zone label"
This message indicates that the **LABEL** keyword was entered with no additional parameter or that the label had more than 127 characters.
- 6274** "ERROR: Invalid results format in zone results file"
The ZRF file may have been created by some application other than NestorReader OT API, or it may have been corrupted.
- 6279** "ERROR: Invalid value used with **RESULTS_FORMAT** keyword"
Refer to ZDF and **ZRF Files** for proper value for **RESULTS_FORMAT**
- 6280** "ERROR: No filename following **OUTPUT_IMAGE_NAME** keyword"
Either **NONE** or a valid path or pathname should follow the **OUTPUT_IMAGE_NAME** keyword.
- 6281** "ERROR: No filename following **ZDF_FILENAME** keyword"
This may be an edited ZRF file which has had the information deleted.
- 6282** "ERROR: No memory loaded for the constraint specified."
There may be a problem with the **CONSTRAINT** definitions in the ZDF file.

- 6283** "ERROR: No memory named for DEFINE_CONSTRAINT keyword."
The ZDF file has an incorrect format for the **DEFINE_CONSTRAINT** keyword.
- 6284** "ERROR: No setting following DROP_OUT_INK keyword."
- 6285** "ERROR: Invalid setting following DROP_OUT_INK keyword."
A hand-edited ZDF file has an incorrect grammar for this keyword.
- 6286** "ERROR: No image name following FORM_REMOVAL_TEMPLATE keyword."
This keyword must be followed either by **NONE** or by an image name.
- 6287** "ERROR: Undefined logical constraint followed CONSTRAINT keyword."
No logical constraint was defined for the character used with the **CONSTRAINT** keyword in a Data zone.
- 6288** "ERROR: Invalid value followed PRINT_TYPE keyword."
- 6289** "ERROR: No value followed PRINT_TYPE keyword."
This may be the results of an edited ZRF file using a program other than NestorReader OT API.
- 6290** "ERROR: Bad format encountered in OMR_GROUP."
This is usually the result of hand-editing of a ZDF or ZRF file, or of such a file created by another application in the wrong format.
- 6291** "ERROR: Mismatch between OMR_NUM_BOXES and actual number of boxes."
This is usually the result of hand-editing of a ZDF or ZRF file, or of such a file created by another application in the wrong format. Ensure that the number of OMR box definitions matches the number defined by **OMR_NUM_BOXES**
- 6292** "ERROR: No value followed OMR_NUM_BOXES keyword."
- 6293** "ERROR: No value followed OMR_TYPE keyword."
- 6294** "ERROR: No OMR_TYPE keyword was specified for an OMR zone."

	The OMR zone description does not follow the correct grammar.
6295	<p>"ERROR: Fewer strings than OMR boxes defined."</p> <p>Not enough strings were defined to associate with the OMR boxes. The number of strings, if there are any at all, should match the number of OMR boxes.</p>
6296	"ERROR: Invalid OMR type specified."
6297	"WARNING: No action specified with keyword."
6298	"WARNING: Invalid setting used with keyword."
6299	<p>"RECOVERABLE_ERROR: No value followed ORIGINAL_IMAGE_SIZE keyword."</p> <p>The zone description does not follow the correct grammar.</p>
6300	<p>"ERROR: could not open TXT file"</p> <p>NestorReader OT API could not open a file for output in the TXT format. Check the output file name and results directory setting.</p>
7003	<p>"ERROR: determining registration data."</p> <p>This error indicates that NestorReader OT API could not register the image using the specified registration mark(s). The image may not be similar to the one used to initialize the registration information, or it may be too severely skewed or shifted for the registration mark to be found.</p>
7004	<p>"ERROR: Writing zone location file."</p> <p>NestorReader OT API could not open the specified .ZRF file. This may happen if the disk is read-only (such as a CD-ROM), or if the disk is full.</p>
7006	<p>"ERROR: could not find registration mark."</p> <p>See error #7003.</p>
7007	<p>"ERROR: reading .spc file"</p> <p>See error #6023.</p>
7011	"ERROR: Invalid tiff byte order, file may not be valid tiff format."
7012	"ERROR: Invalid tiff version number, file may not be valid tiff format."
7013	"ERROR: Reading tiff file header, file may not be valid tiff format."

7014	"ERROR: Decompressing tiff file."
7015	"ERROR: Reading tiff file tag information."
7016	"ERROR: Reading tiff file, multiple strip files not supported."
7017	"ERROR: Reading tiff file, samples per pixel must be 1."
7018	"ERROR: Reading tiff file, only orientation type 1 supported."
7019	"ERROR: Reading tiff file, gray scale not supported." These error codes indicate that this is not a TIFF image format which is supported by NestorReader or that your TIFF image has been corrupted.
7020	"Word is too complex for context and/or dictionary." See error #6101.
7021	"ERROR: Tiff low res fax images not supported for specified compression."
7022	"ERROR: unknown compression type in tiff file."
7023	"ERROR: LZW compression not supported in tiff files."
7024	"ERROR: PACK BITS compression not supported in tiff files."
7025	"ERROR: Tiff group 3 and 4 uncompressed formats not supported."
7026	"ERROR: Unknown group 3 compression option in tiff file."
7027	"ERROR: Unknown group 4 compression option in tiff file."
7028	"ERROR: Reading or decompressing image file."
7029	"ERROR: Reading tiff file, bits per pixel must be 1."
7030	"ERROR: Reading tiff file, bits per sample must be 1." These error messages indicate that this is not a TIFF image format which is supported by NestorReader or that your TIFF image has been corrupted.
7031	"ERROR: DotsPerInch not specified for CharsPerInch segmentation." The IMAGE_RESOLUTION_DPI parameter must be entered if the N_CHARS_PER_INCH keyword is used. Ensure that the ZDF file declares the resolution of the image in dots per inch before specifying a zone which uses CPI segmentation.

7033	<p>"ERROR: Image too large to process."</p> <p>The image file is too large, and cannot be processed. If feasible, the file may be scanned piecewise or separated into parts by a graphics program.</p>
7035	<p>"ERROR: Unknown image type."</p> <p>NestorReader OT API determines an image's type based on its suffix (TIF, PCX, etc.). This error indicates that the suffix is not one that is supported by NCS. See the documentation for a list of supported image types and their suffixes.</p>
7037	<p>"ERROR: Too Many Registration/ID zones."</p> <p>More than a total of 15 Registration and Identification zones were defined. Delete the excess zones.</p>
7038	<p>"ERROR: Failed to Identify page."</p> <p>The form identification logic was unable to determine the form type for the current image. The application logic should take appropriate action when it detects this error.</p>
7039	<p>"ERROR: Only one Identification zone is allowed."</p> <p>More than one Identification zone was defined in the ZDFile or in the ZDFdata structure. Only one is allowed.</p>
7040	<p>"ERROR: Single Registration zones may not be of label type."</p> <p>If there is only one Registration zone, it must not contain a string, else the registration logic will not be able to determine a skew angle accurately. Either define an additional Registration zone, or select a different Registration zone.</p>
7041	<p>"ERROR: Missing Identification zone in ZDF."</p> <p>Form ID attempted with no ID Zones defined. Each ZDF data structure used in such cases must have such a zone.</p>
7042	<p>"ERROR: Could not locate OMR mark."</p> <p>No OMRmark was found in the defined OMR field.</p>
7044	<p>"ERROR: All registration zones must be strings to use NR_IdAndProcess."</p> <p>An Identification zone was found to have no string in its szRegistrationIDStringfield.</p>
7045	<p>"ERROR: Incorrect memory file format or floating point library mismatch."</p> <p>A recognition memory may be corrupted, or (if using the WATCOM compiler) there may be a mismatch in the floating</p>

	point options used in compiling the application versus the library being used.
7046	<p>"ERROR: Unable to load memory; please remove one or more to make room."</p> <p>The recognizer is running low on available RAM</p>
7047	<p>"WARNING: Can't look for registration strings. No machine print memory loaded."</p> <p>A machine print memory needs to be defined in index 0 of the logical constraints array prior to attempting to perform registration or location of registration marks.</p>
7049	<p>"WARNING: Excessive image distortion found during registration."</p> <p>A nonlinear image distortion was detected, and registration may not be accurate. the image should be rescanned if possible.</p>
7050	<p>"ERROR: could not find registration strings."</p> <p>The image may be missing the strings specified, or its quality may be such that the strings are unrecognizable.</p>
7051	<p>"WARNING: Path for OUTPUT_IMAGE_NAME cannot be the same as input image path"</p> <p>The output image may overwrite the original image if it is written to the same directory as the original, so NestorReader OT API gives a warning when both the input image and the output image are in the same directory.</p>
7052	<p>"ERROR: Attempting recognition with no data zones defined."</p> <p>The ZDF file being used for recognition has no Data or OMR zones defined, so recognition will produce no recognition results. Define one or more Data or OMR zones before attempting recognition.</p>
7053	<p>"ERROR: Problem opening form removal template file."</p> <p>The image file specified for use during form removal could not be located in the directory specified, the current directory, or <NESTORPATH>\IMAGES Make sure that the file name is correct.</p>
7054	"ERROR: Recognition job was canceled by the user."
7055	"Bad file name; cannot open master image file. Form removal is not possible."

	<p>The file named to be used as a master image for form removal could not be found. Make sure that the value for the FORM_REMOVAL_TEMPLATE keyword is a valid pathname for an image file.</p>
7056	<p>"WARNING: Region too large for character repair (dilation)."</p> <p>Available RAM prohibits using character repair on one or more zones due to their size. Reduce the number of characters in each zone, or turn off character repair.</p>
7057	<p>"ERROR: Initializing work buffer, may be problem with nestor.ini."</p> <p>Either there was a problem in reading the WORK_BUFFER_SIZE parameter from NESTOR.INI or the value specified there was less than 500000.</p>
7058	<p>"ERROR: Opening startup file nestor.ini."</p> <p>An error was encountered in trying to open NESTOR.INI. Make sure that it exists in a subdirectory named BIN underneath the directory specified by the NESTORPATH environment variable.</p>
7059	<p>"ERROR: missing value in [zdf defaults] section of nestor.ini."</p> <p>An error occurred while trying to read a value from the [zdf defaults] area of NESTOR.INI. That section should be examined to ensure that it contains valid definitions for all keywords present in a fresh installation of NESTOR.INI.</p>
7060	<p>"TXT file for dictionary creation was empty."</p> <p>The text (.TXT) file supplied for creating a dictionary was empty. Since an empty dictionary is useless, no dictionary was created.</p>
10001	"OmniTools Internal Error"
10002	<p>"OmniTools Internal Error"</p> <p>An internal error occurred in NestorReader OT API which may cause unpredictable behavior.</p>
10003	<p>"GlobalAlloc failed, memory low"</p> <p>Windows global memory is low. You should close some other application to free Windows memory or exit the application using NestorReader OT API.</p>
10004	"GlobalLock failed"

	NestorReader OT API was unable to access allocated memory. This might be corrected by closing some other application.
10005	"Job was corrupted" NestorReader OT API's internal data associated with a job was corrupted. The recognition will need to be redone.
10006	"All available sessions are in use" NestorReader OT API's internal limit on the number of open sessions has been met. One or more open sessions should be deleted using OT_DeleteSession() or another application which is using NestorReader OT API should be closed.
10007	"RegisterClass failed" NestorReader OT API was unable to register a Windows class with the operating system. Windows's system resources may be exhaustion; another application should be closed to free resources.
10008	"GetCurrentTask failed" NestorReader OT API was unable to determine the task identifier for the calling application. The Windows operating system may be confused or corrupted.
10009	"CreateWindow for the invisible window failed" NestorReader OT API was unable to create an invisible window used to process Windows messages. Windows's system resources may be exhaustion; another application should be closed to free resources.
10010	"Cannot find NRWTOOLS.DLL" NestorReader OTAPI cannot locate an essential DLL that is needed to perform recognition. Check to make sure that the NESTORPATH environmental variable is set to the directory where NestorReader OT API was installed, and that NRWTOOLS.DLL is in the LIB subdirectory under that path.
10011	"Missing function in NRWTOOLS.DLL" NestorReader OT API was unable to locate a function in NRWTOOLS.DLL . Check to make sure that the versions of NRWTOOLS.DLL in the <NESTORPATH>\LIB directory and the OMNI.DLL/OMNI32.DLL in your Windows directory are compatible.
10012	"DestroyWindow for the invisible window failed"

- NestorReader OT API was unable to destroy a window it created. The Windows operating system may be confused or corrupted.
- 10013** "UnregisterClass failed"
- NestorReader OT API was unable to unregister a class it created. The Windows operating system may be confused or corrupted.
- 10014** "Last available session has been used"
- The session that was just created is the last one that can be created for use with NestorReader OT API. This is for informational purposes only.
- 10015** "Cannot initialize Recognition Server"
- NestorReader OT API was unable to start the Recognition Server. No recognition may be performed. It is possible that there is not enough RAM available to start the Recognition Server; either close some other application or increase the size of your Windows Virtual Memory.
- 10016** "OmniTools can run only in Optanced Mode"
- An attempt was made to start NestorReader OT API while running Windows 3.x in Standard mode. NestorReader OT API requires that Windows 3.x be run in Optanced mode.
- 10017** "File not found"
- NestorReader OT API was unable to find an image or ZDF file. Check to make sure that the pathnames used in the NestorReader OT API function call exist.
- 10018** "Filenames in ZDFList must have ZDF or LST extensions"
- NestorReader OT API detected a file that did not have a ZDF or **.LST** extension in the **ZDFList** parameter in a call to **OT_RecognizeFile()** or **OT_RecognizeMem()**. Only ZDF files or **.LST** files may be supplied for this parameter.
- 10019** "The logical constraint for the DEFAULT data zone is now undefined"
- This is a warning that the ZDF file which was just read does not define the logical constraint which is being used in the **DEFAULT** data zone. Either define the logical constraint being used, or set the **DEFAULT** zone's logical constraint to the value of a defined logical constraint.

10020	<p>"Reached maximum permitted number of ZDF files for this job"</p> <p>This is a warning that the number of ZDF files specified for form identification (in the ZDFList parameter in a call to OT_RecognizeFile() or OT_RecognizeMem()) has reached the maximum permitted in NestorReader OT API. The number of forms being identified should be reduced.</p>
10021	<p>"No ZDF defined for recognition job"</p> <p>No ZDF file name was specified in the ZDFList parameter in a call to OT_RecognizeFile() or OT_RecognizeMem(). A valid ZDF or LST file name must be supplied.</p>
10022	<p>"Error occurred reading file"</p> <p>Some but perhaps not all ZDF file names were read from a LST file. There may be a problem with the contents of the LST file.</p>
10023	<p>"NRWTOOLS.DLL was unable to perform requested function"</p> <p>An internal error has occurred in NestorReader OT API. Future calls to NestorReader OT API may also have problems. The application should close the NestorReader OT API session and open another NestorReader OT API session.</p>
10024	<p>"Unable to find specified job"</p> <p>The job number supplied to the NestorReader OT API function cannot be found. Either it is an invalid job number, or the results for that job have already been deleted.</p>
10025	<p>"The Recognition Server is not running"</p> <p>NestorReader OT API has detected that the Recognition Server has terminated abnormally. the application should exit NestorReader OT API and restart.</p>
10027	<p>"The specified job is not yet completed"</p> <p>The application called OT_WriteResults() for a job number which has not yet completed recognition. The application should wait until the job's status indicates that recognition is complete before trying to write the results to disk.</p>
10028	<p>"No file name specified"</p> <p>The string which was expected to contain a file name was empty. Make sure that a valid file name is supplied to the NestorReader OT API function.</p>

- 10029** "No results are available for this job"
- An attempt was made to write or modify results when there were no results present for the job specified. Make sure that the job has results available before trying to write or access them.
- 10031** "The ZRF file cannot be modified"
- The settings for a **ZRF** file which has been read into the buffer may not be modified; only the results may be modified.
- 10032** "GLOBAL and DEFAULT are reserved zone names"
- The zone names **GLOBAL** and **DEFAULT** are reserved names; Data, OMR, Registration, and ID zones may not be created with these names. Select another name for the zone.
- 10033** "The name supplied is too long"
- A zone name was specified which was longer than the a maximum of 128 characters in length; define a shorter zone name.
- 10034** "A name is required"
- Either the name of the zone or the name of a parameter was missing in the NestorReader OT API call which generated the error. Check the strings being supplied to the call.
- 10037** "No zone found for specified name"
- There is no zone in the current ZDF file with the name specified. **OT_GetZoneName()** should be used to get the valid zone names.
- 10038** "Duplicate zone names are not allowed"
- An attempt was made to define a zone with the same name as an existing zone. This is not allowed in NestorReader OT API.
- 10039** "Not a valid ZDF parameter"
- An unrecognized parameter name was specified in an NestorReader OT API call. Please refer to **Chapter 3: ZDF and .ZRF Files** for the valid parameter names.
- 10040** "Specified ZDF parameter setting is not valid"
- An invalid format was found for the parameter specified. An example of such a problem is if a string which does not evaluate to a numeric value was supplied for a parameter

	which requires a numeric value. Please refer to Chapter 3: ZDF and .ZRF Files for the valid parameter settings.
10041	<p>"Internal ZDF value is not known"</p> <p>A setting for a parameter was specified which was not a valid value for that parameter. Please refer to Chapter 3: ZDF and .ZRF Files for the valid parameter settings.</p>
10042	<p>"Supplied confidence cut-off value is invalid"</p> <p>Confidence cutoff values must be in the range -1 to 100; a value outside that range was specified.</p>
10043	<p>"Supplied grouping code is invalid"</p> <p>Please refer to OT_SetResultsFormat() for the valid grouping settings.</p>
10044	<p>"Supplied job number is invalid"</p> <p>No job with the number specified exists. Check the code calling NestorReader OT API for its handling of job numbers.</p>
10045	<p>"Cannot rename the default zone"</p> <p>An attempt was made to reset the LABEL parameter for the DEFAULT zone. This is not allowed.</p>
10046	<p>"No defined zones; cannot write the file"</p> <p>A ZDF file cannot be created with no Data, OMR, Registration, or ID zones in it. Add one or more zones to the ZDF definition before attempting to write it to disk.</p>
10047	<p>"No image supplied"</p> <p>A valid image was not supplied in RAM for OT_RecognizeMem(). Verify that an image is present in RAM before attempting recognition.</p>
10048	<p>"Length supplied is not valid"</p> <p>A negative value was supplied as the length of either the image or its header in the call to OT_RecognizeMem(). This is meaningless, and indicates some problem in the calling code.</p>
10049	<p>"An image header was expected but not supplied"</p> <p>The length of the header was specified, but the MEMPTR for the header was NULLPTR in a call to OT_RecognizeMem().</p>
10050	<p>"Unrecognized imagetype"</p>

Either the image type specified was unfamiliar, or NestorReader OT API was unable to determine the type of the image. If the image was being taken from a file, check its extension to ensure that it is **.DCX**, **.PCX**, **.PDA**, or **.TIF** as appropriate. Otherwise, check the type being specified in the NestorReader OT API call.

- 10052** "Zone name contains invalid character"
- Zone names may contain letters, numbers, and underscores (_) as valid characters. Spaces and other characters are not allowed.
- 10053** "ZDF values can be set only in ZDF file read into edit area"
- Only the ZDF definition in the edit area (job number -1) can have its settings changed.
- 10056** "Illegal zone syntax expression"
- An invalid expression was used in defining zone syntax. Please refer to **Appendix B: Defining Syntax for a Zone** for the valid expression formats.
- 10058** "Logical constraint has not been defined"
- An undefined logical constraint was specified while defining syntax information. Please ensure that the logical constraint is defined prior to using it in a syntax definition.
- 10059** "Zone syntax wildcards are limited to one per expression"
- Only one wild card is allowed per word, line, or zone syntax definition. Please refer to **Appendix B: Defining Syntax for a Zone**.
- 10060** "Expression contains undefined zone syntax symbol"
- An unfamiliar symbol was found in a zone syntax definition. Please refer to **Appendix B: Defining Syntax for a Zone** for the valid expression formats.
- 10061** "Zone syntax expression is empty"
- The string supplied did not contain the name of a **WORD_TYPE** or **LINE_TYPE**. Please refer to **Appendix B: Defining Syntax for a Zone** for the valid expression formats.
- 10062** "Cannot delete symbol, because it is contained in higher level symbols"
- An attempt was made to delete a **WORD_TYPE** or **LINE_TYPE** definition by supplying an empty string for its value. However, a higher-level syntax expression uses that

definition (e.g., a **ZONE_SYNTAX** may use the **LINE_TYPE** being cleared, or a **LINE_TYPE** might use the **WORD_TYPE** being cleared). Syntax definition should be cleared starting at the topmost level: **ZONE_SYNTAX**, then **LINE_TYPE**, then **WORD_TYPE**

10063 "Zone syntax symbol was not found"

A **WORD_TYPE** or **LINE_TYPE** was not found for use in defining a higher-level syntax. Each **WORD_TYPE** must be defined before being used in a **LINE_TYPE** definition, which in turn must be defined before being used in a **ZONE_SYNTAX** definition.

10064 "Missing dictionary name"

The call to the NestorReader OT API dictionary maintenance function was not supplied with a dictionary file name.

10065 "Unknown dictionary type"

The dictionary type specified was not **NORMAL** (0) or **SPECIAL** (1).

10066 "Unable to create temporary working file"

NestorReader OT API was unable to create a temporary file on disk. The disk might be full, or the **FILES** setting in your **CONFIG.SYS** may need to be increased.

10067 "Missing word to delete"

No word was supplied to be deleted in the call to **OT_DeleteWord()**

10068 "Unable to copy file"

NestorReader OT API was unable to perform an operation on a temporary file being used during dictionary maintenance. The disk might be full.

10069 "Unable to read dictionary file"

NestorReader OT API was unable to open the dictionary file specified. Check to ensure that the file exists.

10077 "Logical constraint labels must be one character (0-9, A-Z, or a-z)"

Please refer to **Chapter 3: ZDF and .ZRF Files** for the correct logical constraint settings.

10078 "Defined memory name too long to return in OmniTools string"

	The combination of the memory name and the logical constraint settings is too long to return in the standard length NestorReader OT API string.
10080	<p>"Unknown print type"</p> <p>The print type specified for a logical constraint must be one of MP, HP, or MH.</p>
10081	<p>"Unable to start OMNIVIEW.EXE"</p> <p>NestorReader OT API was unable to start the NR Form Editor utility. The problem may be a lack of Windows resources, or the file NREDITOR.EXE may not be in its expected place in <NESTORPATH>\BIN</p>
10082	<p>"Unable to start VIDEO.EXE"</p> <p>NestorReader OT API was unable to start the NR Verifier utility. The problem may be a lack of Windows resources, or the file NRVERIFY.EXE may not be in its expected place in <NESTORPATH>\BIN</p>
10083	<p>"Resulting command line greater than 128 characters"</p> <p>The command line needed to start the utility exceeded the maximum allowed. If calling ODT_CorrectResults() try putting the names of ZRF files in a LST file instead of supplying them as parameters. If calling ODT_DisplayImage() try putting the names of allowed zones into a LST file. If neither of these apply, try to shorten the pathnames of the image and/or ZDFZRF files.</p>
10084	<p>"The parameter is read-only, and cannot be set by the user"</p> <p>The parameter specified in the .ZRF file may only be examined, not modified, after recognition.</p>
10085	<p>"Unknown file format specified"</p> <p>The results file format specified is not supported by NestorReader OT API at this time. Please refer to OT_WriteResults() for the valid file formats.</p>
10086	<p>"Results file name required when OT_RecognizeMem is used"</p> <p>OT_WriteResults() requires a filename for the results when OT_RecognizeMem() was used to perform recognition, because there is no image name to use as the basis for the results file name.</p>
10087	"The Recognition Server is too busy to respond"

	Too many requests for recognition have been submitted, and the Recognition Server is unable to process more requests at this time. The recognition request should be made again at a later time.
10088	<p>"The string length parameter is too short"</p> <p>The minimum length for a string is 10; call OT_SetParameterHandling() with a value at least that large.</p>
10089	<p>"Unknown string style"</p> <p>Please refer to the documentation for OT_SetParameterHandling() for the allowed values for the string style.</p>
10090	<p>"Unknown MEMPTR style"</p> <p>Please refer to the documentation for OT_SetParameterHandling() for the allowed values for the MEMPTR style.</p>
10091	<p>"Unknown environment type"</p> <p>Please refer to the documentation for OT_SetEnvironment() for the allowed values for the environment type.</p>
10092	<p>"Unknown zone type"</p> <p>Please refer to the documentation for OT_AddZDFZone() for the allowed values for the zone type.</p>
10093	"No more than 4 registration zones and 1 ID zone are allowed"
10094	<p>"Image snippets are not supported for OMR and identification zones"</p> <p>A complete image must be supplied when calculating OMR and identification zones due to their reliance upon the location of the zone on the page.</p>
10095	<p>"Call OT_UseImageFile to define an image for registration or ID zones"</p> <p>A reference image file must be defined for use in defining Registration or Identification zones.</p>
10096	<p>"The index is not the index of a new or existing object"</p> <p>Use an index of 0 or N + 1 (where N is the current number of objects) to indicate a new object, or use a value in the range 1</p>

through N to indicate an existing object, when accessing an indexed parameter.

10097 "The OMR box did not exist"

The application has supplied 0 for the coordinates or for the width and height of an OMR box which is being newly defined. The new box will not be created.

10098 "Use OT_DeleteZDFZone to delete all or the last OMR box in a zone"

Use a call to **OT_DeleteZDFZone()** rather than setting the coordinates of the last OMR box to 0 in order to delete the last OMR box.

10099 "The attribute is not valid for the type of zone"

An attempt was made to access or set a parameter which does not exist for the type of zone being accessed. Check the name of the zone and its type to ensure that the correct zone is being accessed in the correct manner.

10100 "The index is not the index of an existing object"

An attempt was made to access an indexed parameter for an index which does not exist. Check the code to see that it knows how many indices are valid for the indexed object.

10101 "Cannot set the registration or identification string to empty"

The **REGISTRATION_STRING** and **IDENTIFICATION_STRING** values must be non-empty strings. Additionally, the application code should seldom modify these strings, as they are set by NestorReader OT API.

10102 "This registration zone uses a graphic mark, not a string"

An attempt was made to access the **REGISTRATION_STRING** for a zone which uses a graphic mark rather than a string.

10103 "Word grouping does not apply to OMR zones, using line grouping instead"

10104 "The logical constraint table cannot hold more constraints"

An old-style ZDF file not created by NestorReader OT API was read whose zone syntax definitions are too complex for NestorReader OT API. Create a new ZDF file using NestorReader OT API.

10105 "Only existing OMR strings can be set to a new value"

	An attempt was made to define new OMR strings in ZRF file. The number of OMR strings and boxes cannot be changed in a results file.
10106	"Logical constraints 0 to 9 are reserved for the system and cannot be modified"
10107	<p>"A system reserved logical constraint (0-9) does not agree with OMNI.INI"</p> <p>A ZDF or ZRF file was read which has a different definition for one of the logical constraints in the range 0-9 from the values in OMNI . INI</p>
10108	<p>"Unknown format for DEFINE_CONSTRAINT definition"</p> <p>The parameter setting associated with the DEFINE_CONSTRAINT string was not in the proper format. Please refer to Chapter 3: ZDF and .ZRF Files for the proper format.</p>
10109	<p>"Logical constraint 0 must be defined to determine registration/ID values"</p> <p>NestorReader OT API uses logical constraint 0 to locate strings for Registration and Identification zones. Logical constraint 0 must be defined prior to the definition of these types of zones, and it should be associated with an alphanumeric machine print memory.</p>
10110	<p>"The OMNI.INI file was not found"</p> <p>NestorReader OT API reads some default settings from the file <NESTORPATH>\BIN\OMNI . INI This file should be present as a result of installation of NestorReader OT API; if not, please reinstall NestorReader OT API.</p>
10111	<p>"The attribute is valid for ZDF but not ZRF files"</p> <p>An attempt was made to set a parameter in a results ZRF file which is only applicable in a ZDF file.</p>
10112	<p>"The ZDF has too many data zones"</p> <p>The ZDF file has too many data zones to fit into RAM. Multiple ZDF files may have to be used to process the image.</p>
10113	<p>"The decompression engine is busy"</p> <p>The decompression engine is in use by another NestorReader OT API session. The decompression attempt should be made again later.</p>
10114	"Image is being used by a prior call to OmniTools"

	An attempt has been made to replace the image in the image buffer while the image was in use for recognition, or an attempt was made to change scanner settings while an image was being scanned. The attempt should be made again later.
10115	<p>"Scanner is being used by another OmniTools session"</p> <p>An attempt was made to scan an image while another session was already scanning an image. The attempt should be made again later.</p>
10116	<p>"Image file already exists and has not been overwritten"</p> <p>The file named in an OT_WriteImage() call already existed; NestorReader OT API will not overwrite an image file.</p>
10117	<p>"Unknown file compression format"</p> <p>The compression format specified in an OT_WriteImage() call is not one that is recognized by NestorReader OT API. Check the value specified for the compression format.</p>
10118	<p>"Must first read or scan image"</p> <p>OT_WriteImage() was called while there was no image in the image buffer.</p>
10119	<p>"The control string is not valid"</p> <p>The control string passed to OT_AddWork() was either empty or had one or more invalid characters in it.</p>
10120	<p>"Unknown return status supplied"</p> <p>The status supplied when calling OT_ReturnWork() was not a valid value.</p>
10121	<p>"A control or control handle was not supplied as a parameter"</p> <p>The hControl parameter passed to OT_CreatePicture() was not a valid control handle for Visual Basic.</p>
10122	<p>"The property specified was not found in the control"</p> <p>The control associated with the hControl parameter passed to OT_CreatePicture() did not have a PICTURE property.</p>
10123	"Problem setting the specified picture property"
10124	<p>"Error calling CreateBitmapIndirect"</p> <p>An error occurred while trying to set the PICTURE property of the hControl parameter passed to OT_CreatePicture(). The system may be low on available RAM.</p>
10125	"FlushBuffer should be either 0 or 1"

	The IFlushBuffer parameter passed to OT_CreateBitmap() or OT_CreatePicture() had a value other than 0 or 1, which is invalid.
10126	"The clip coordinates are not reasonable" Either IBottomRightX was less than ITopLeftX , or IBottomRightY was less than ITopLeftY when specifying the coordinates for a clip out of an image.
10127	"The clip coordinates are off the image" The coordinates for a clip out of an image did not fall within the image's boundaries.
10128	"Size of clip was reduced because it was partially off the image"
10129	"Scaling factor must be 0, 2, 4, 8, 16, or 32"
10130	"There is a ZRF file in the edit buffer. Need a ZDF file." Recognition was attempted that specified the use of the ZDF definition in the edit buffer, but there was a ZRF file (with results in it) in the edit buffer instead.
10131	"GlobalAllocPtr failed, memory low!" Available RAM may be almost exhausted. Try closing some applications to make more RAM available.
10132	"CreateBitmap failed"
10133	"CreateIC failed"
10134	"CreateCompatibleDC failed"
10135	"CreateCompatibleBitmap failed"
10136	"SetBkColor failed"
10137	"SetTextColor failed"
10138	"BitBlt failed" An error occurred while trying to set the PICTURE property of the hControl parameter passed to OT_CreatePicture() . The system may be low on available RAM.
10139	"ForceBlackAndWhite must be either 0 or 1"
10140	"Cannot find OT_FORM.DLL" NestorReader OT API was unable to locate OT_FORM.DLL , which should be in the <NESTORPATH>\LIB directory. Make sure that the file is present; if it is not, it may be

- necessary to reinstall NestorReader OT API or to reset the **NESTORPATH** environment variable.
- 10141** "Missing function in OT_FORM.DLL"
- NestorReader OT API was unable to find an expected function in **OT_FORM.DLL**. There may be an NestorReader OT API version mismatch, or **OT_FORM.DLL** may have been overwritten by a file with the same name from a different program. Reinstall NestorReader OT API with the latest version available.
- 10142** "Cannot find OT_ZONE.DLL"
- NestorReader OT API was unable to locate **OT_ZONE.DLL**, which should be in the **<NESTORPATH>\LIB** directory. Make sure that the file is present; if it is not, it may be necessary to reinstall NestorReader OT API or to reset the **NESTORPATH** environment variable.
- 10143** "Missing function in OT_ZONE.DLL"
- NestorReader OT API was unable to find an expected function in **OT_ZONE.DLL**. There may be an NestorReader OT API version mismatch, or **OT_ZONE.DLL** may have been overwritten by a file with the same name from a different program. Reinstall NestorReader OT API with the latest version available.
- 10144** "Pixel Translations Error"
- This is the error number, and error prolog whenever we report an error message received from the Pixel code.
- 10145** "The NRWTOOLS.DLL and RSRVRX87.EXE are not rev compatible with OMNI.DLL"
- This error can only occur in the 16 bit product. It indicates a mismatch between components and will prevent a session from opening.
- 10146** "OT_RecognizeMem ignores the ZDF attribute OUTPUT_IMAGE_NAME"
- The ZDF attribute **OUTPUT_IMAGE_NAME** indicates the file directory where the image file (after preprocessing) is to be stored. The implementation of this functionality assumes that the image was from file. In the case of API function **OT_RecognizeMem**, the of the image file that was read (if there was one) is not available, and so this warning is issued, and this attribute is turned off.

10147	<p>"The ZDF parameter is undefined"</p> <p>This error occurs when the call to <code>OT_GetZDFValue</code> asks for value of an undefined attribute.</p>
10149	<p>"The Win32 version can support no more than 128 client sessions"</p> <p>The 32 bit version of the product supports 128 sessions per application. When this error occurs, no more sessions can be opened by the application.</p>
10151	<p>"The Nestor path is not defined in the environment or in OMNITOOL.INI"</p> <p>This warning is issued when creating a session if the <code>NESTORPATH</code> has not been defined. When the <code>NESTORPATH</code> is not defined, the operation of <code>NestorReader</code> OT API is unpredictable.</p>
10152	<p>"The dictionary is empty"</p> <p>This warning occurs when the call to <code>OT_ConvertDictionaryToText</code> operates on a dictionary that is empty.</p>
10153	<p>"The handle to the device context is not valid"</p> <p>This error occurs when passing in the <code>OT_ViewImage</code> call a null or invalid handle to a device context.</p>
10154	<p>"OT_CreatePicture is not supported: use OT_ViewImage"</p> <p>This error occurs when calling <code>OT_CreatePicture</code> in the 32 bit product.</p>

Appendix B : File Formats

Notice

The contents of this appendix are reprinted from "Appendix A: File Formats" in the *NestorReader 3.0 Developer's Guide*, © Copyright Nestor Incorporated, 1995, with modifications as necessary to reflect the interface available through the ImageBASIC Nestor ActiveX control.

Zone Definition and Zone Recognition File Formats

NestorReader uses several file formats for storage of form definitions, results, and other data. Some of these files are in ASCII text format, and are usually accessed by NestorReader functions. However, at some times it may be preferable to the developer to create or interpret these files directly. This appendix will describe the formats of those files which developers may be to edit.

The Zone Definition File (ZDF) and the Zone Recognition File (.ZRF) are the main form of file-based communication between the NestorReader recognition server and a user's application.

- The ZDF file instructs the recognition server on how to perform image processing for an image
- The ZRF file provides the results of the recognition (region processing with the option of ICR results) back to the application

Each file consists of a set of *Global Definitions* and one or more *zones* of several types. In addition, the ZRF file may contain the results of recognition.

- The ZDF file is generated by the SaveZDFFile method, or it may be created using a text editor or other means
- The ZRF file is generated as a result of ICR and is available as one of the OutputFormat options

Both types of files are written in ASCII format for easy viewing and both files may be manually edited with a standard text editor. Form- and zone-level settings are defined by functional keywords and associated values. All keywords which are stored in a ZDF file will be duplicated in the ZRF file which results from recognition using that ZDF file. The ZRF file differs from the ZDF file in that it has a block of results added to each zone definition using its own procedural keywords.

These files are organized into the following types of data:

- Global Definitions
- Data Zone Definitions
- OMR Zone Definitions
- Registration Zone Definitions
- Identification Zone Definitions

ZDF Global Definitions

The *Global Definitions* define settings which are applicable for the duration of the use of the ZDF file, or which were in effect during the recognition which produced the **ZRF** file. Settings defined in the Global Definitions cannot be changed in subsequent zone definitions.

The Global Definitions occur at the beginning of the ZDF or ZRF file. Any Global Definition which appears after the first occurrence of any type of zone is an error and is ignored. Most of the keywords are available by using the dialogs within NR Form Editor. Ones that are not available through NR Form Editor are noted as such and can be accessed by opening the .ZDF file in any text editor.

The allowed Global keywords are as follows:

- ALLOW_ID_USING_BLOBS
- BLOB_MEMORY
- DEFAULT_DOCUMENT_TYPE
- DEFINE_CONSTRAINT
- DESKEWING
- DROP_OUT_INK
- FLAG_UNCERTAIN_CHAR_MAX_HEIGHT
- FLAG_UNCERTAIN_CHAR_MIN_OVERLAP
- FLAG_UNCERTAIN_CHAR_REPLACEMENT
- FLAG_UNCERTAIN_CHAR_SPLIT_OVERLAP
- FORM_REMOVAL_TEMPLATE
- GRAPHICS_REMOVAL
- IMAGE_RESOLUTION_DPI
- LINE_TYPE
- LONG_LINE_LENGTH

MAX_ID_ERROR
MAX_LINE_GAP
MAX_NONLINEAR_REG_ERROR
MAX_REG_ERROR
MAX_REG_ID_ZONES
MAX_REG_OFFSET
MIN_LINE_SPLIT_HEIGHT
MIN_MATCHED_ID_MARKS
MIN_MATCHED_REG_MARKS
MIN_OMR_FILL_STD
MIN_OMR_FILL_WT
NOISE_AREA
NOISE_HEIGHT
NOISE_WIDTH
OMR_BORDER_SIZE
ORIGINAL_IMAGE_NAME
ORIGINAL_IMAGE_SIZE
OUTPUT_IMAGE_NAME
PUNCTUATION_MEMORY
REGISTRATION
REMOVE_LINES
RESULTS_FORMAT
SMART_NOISE_CLASSIFICATION
VERIFY_RESULTS_FORMAT
WEIGHTED_CENTER
WORD_TYPE
ZDF_FILENAME

ALLOW_ID_USING_BLOBS

Syntax: ALLOW_ID_USING_BLOBS < TRUE | FALSE >
Possible Values: TRUE
FALSE

Valid In: ZDF, ZRF

Description: This setting turns on or off whether blobs can be used for form identification. When turned off, only strings are allowed. The default value is **TRUE**.
This keyword is not available through NR Form Editor.

Example: ALLOW_ID_USING_BLOBS TRUE

Related Property: None (Can be set through **SetZoneParam** method)

BLOB_MEMORY

Syntax: BLOB_MEMORY < ON | OFF >

Possible Values: ON
OFF

Valid In: ZDF, ZRF

Description: This setting turns on or off the use of a special neural network to determine what is noise and what is text in an image. When turned off, only rules are used for this determination. The default value is ON.

Example: BLOB_MEMORY ON

Related Property: OptSmartNoiseDetect

DEFAULT_DOCUMENT_TYPE

Syntax: DEFAULT_DOCUMENT_TYPE < TRUE | FALSE >

Possible Values: TRUE
FALSE

Valid In: ZDF, ZRF

Description: When form identification fails and this parameter is True, the first .ZDF file in the list of .ZDFs supplied to the Recognition Sever is taken as the default document type and the remainder of the image processing will be performed using this .ZDF. When this option is True, the first .ZDF in the .ZDF list if not required to contain any identification zones. The only place this keyword is looked for is in the first .ZDF.
This keyword is not available through NR Form Editor.

Example: DEFAULT_DOCUMENT_TYPE TRUE

Related Property: None (Can be set through **SetZoneParam** method)

DEFINE_CONSTRAINT

Syntax: DEFINE_CONSTRAINT <label> <mem> <print> <class>

Valid In:	ZDF, ZRF
Description:	<p>This keyword defines a logical constraint for later use in zone syntax definitions or in specifying the constraint for a Data Zone. This keyword is followed at a minimum by a single character which is to be used as the label for the constraint and the name of the recognition memory to be used for this logical constraint. Optionally, the print type (machine print (MP), hand print (HP), or automatic selection of machine print and hand print (MH)) and a list of the allowed classes for this logical constraint may be defined.</p> <p><label> is a single character in the range 0-9, A-Z, or a-z. The <label> is case-sensitive.</p> <p><mem> is the name for the recognition memory that is to be used for this logical constraint. NestorReader OT API will look for the recognition memory in the <NESTORPATH>\MEMORIES directory.</p> <p><print> may be MP (machine print), HP (hand print), or MH (unknown/mixed). The absence of the <print type> results in the use of the broadest range of print types available in the named <memory>.</p> <p><class> may be the word ALL, or a list of allowed classes. The absence of the <class list> is equivalent to ALL. The list of allowed classes may include ranges for contiguous character values such as aA-F or 0-9. The only default contiguous ranges are 0-9, A-Z, and a-z. Other characters, such as punctuation, hyphens, ampersands, etc. must be listed explicitly. The hyphen (or dash) will be interpreted as a character class when it is seen outside one of these valid range declarations. For instance, a-9 specifies three characters: a, -, and 9.</p> <p>Note: NestorReader uses an initialization (.ini) file, NESTOR.INI. This file is used by Nestor so that it has the correct character memories loaded for registration and identification string creation. This file uses logical constraint labels 0 - 9 as system logical constraints. These logical constraint labels should not be redefined by your application.</p>
Example:	<pre> DEFINE_CONSTRAINT A ANHM0079.ME2 DEFINE_CONSTRAINT B ANHM0079.ME2 MP DEFINE_CONSTRAINT C ANHM0079.ME2 MP 0-9A-Z,.- DEFINE_CONSTRAINT D ANHM0079.ME2 MP 0123456ABCD </pre>
Related Property:	None (Can be set through SetZoneParam method)

DESKEWING Keyword

Syntax: DESKEWING< ON | OFF >
Possible Values: ON
OFF
Valid In: ZDF, ZRF
Description: If deskewing is off, only translation and scaling factors are taken into account during registration. This option only has effect if registration is turned on. It is intended to allow registration of images with only a single registration mark which have been deskewed by some other process. The default value is ON.
This keyword is not available through NR Form Editor.
Example: DESKEWING ON
Related Property: OptDeskewActivate

DROP_OUT_INK

Syntax: DROP_OUT_INK <TRUE | FALSE>
Possible Values: TRUE
FALSE
Valid In: ZDF, ZRF
Description: This keyword specifies whether the printed forms are printed in drop-out("blind") ink. If this setting is **TRUE**, then it will be assumed that any blank OMR boxes that were detected while defining the OMR zone will be invisible to the scanner. If this setting is **FALSE**, the data gathered during the definition of the OMR zone will be used to subtract the printed information from the actual marks.
Example: DROP_OUT_INK TRUE
Related Property: OptDropOutInkUsed

FLAG_UNCERTAIN_CHAR_MAX_HEIGHT

Syntax: FLAG_UNCERTAIN_CHAR_MAX_HEIGHT <PERCENTAGE>
Possible Values: Integers, 0 to 1000
Valid In: ZDF, ZRF
Description: This setting is the maximum percentage of the height of the zone that a character can be in order to be considered for uncertainty flagging. For example, 350 means that the character can be up to 3.5 times as high as the zone. The default value is 350.
This keyword is not available through NR Form Editor.

Example: FLAG_UNCERTAIN_CHAR_MAX_HEIGHT 350

Related Property: UncertainCharMaxHeight

FLAG_UNCERTAIN_CHAR Note:

The next 4 settings deal with the situation where the segmentation process sometimes overlooks characters. There are 2 typical examples of this behavior. The first is when a character from one line drops down into another line-such as a lower case “g” in the first line touching the top of an “l” in the second line. Without these settings, either the “g” and the ‘l’ would not appear in the recognition results, or they would be combined and represented by the wrong character (usually with a high confidence value) in either the first or the second line. The other typical example of overlooked characters is best explained by a “y” that is written with a tiny top and a long tail that extends beyond the current line or zone. In this case, the “y” would not appear in the recognition results. In both examples, nothing is presented to the verification operator to correct.

For a single word or single line zone, the height percentage calculated below is relative to the zone. For a multiple line zone, the height percentage is relative to the height of each line found by the segmentation code.

The noise width and noise height settings also interact with these settings. If something matches the noise criteria, there will be no attempt to flag it as a uncertainty character.

With the following settings, even through the recognition process does not know what the correct characters are, they are at least identified with an uncertainty character on a particular line within a zone and with a low confidence. This guarantees that they will be flagged as uncertain and the verification operator will be given a chance to correct them.

These settings are intended for advanced users and only need to be modified for very special cases.

FLAG_UNCERTAIN_CHAR_MIN_OVERLAP

Syntax: FLAG_UNCERTAIN_CHAR_MIN_OVERLAP PERCENTAGE >

Possible Values: Integers, 0 to 100

Valid In: ZDF, ZRF

Description: This setting is the percentage of the height of the zone or line (for multiple line zones) that a character must be before it can be considered for uncertainty flagging. The default value is 30. This keyword is not available through NR Form Editor. See Note in FLAG_UNCERTAIN_CHAR_MAX_HEIGHT.

Example: FLAG_UNCERTAIN_CHAR_MIN_OVERLAP 30

Related Property: UncertainCharMinOverlap

FLAG_UNCERTAIN_CHAR_REPLACEMENT

Syntax: FLAG_UNCERTAIN_CHAR_REPLACEMENT < CHARACTER >

Possible Values: Any Printing Character

Valid In: ZDF, ZRF

Description: This is what is placed in the results if a character is flagged as uncertain. The verification operator then replaces this with the correct character. The default is ? . This keyword is not available through NR Form Editor. See Note in FLAG_UNCERTAIN_CHAR_MAX_HEIGHT.

Example: FLAG_UNCERTAIN_CHAR_REPLACEMENT ?

Related Property: UncertainCharReplacement

FLAG_UNCERTAIN_CHAR_SPLIT_OVERLAP

Syntax: FLAG_UNCERTAIN_CHAR_SPLIT_OVERLAP < *percentage* >

Possible Values: Integers, 0 to 100

Valid In: ZDF, ZRF

Description: This setting is the percentage into a given line or zone a character must be before it will be considered for splitting into two characters. The default value is 30. This keyword is not available through NR Form Editor. See Note in FLAG_UNCERTAIN_CHAR_MAX_HEIGHT.

Example: FLAG_UNCERTAIN_CHAR_SPLIT_OVERLAP 40

Related Property: UncertainCharSplitOverlap

FORM_REMOVAL_TEMPLATE

Syntax: FORM_REMOVAL_TEMPLATE <NONE | pathname>

Possible Values: NONE
The path and file name of an image file

Valid In: ZDF, ZRF

Description: This keyword specifies the name of the image file being used for form removal. If the pathname specified does not exist, NestorReader OT API will search the current directory and the **<NESTORPATH>\IMAGES** directory for the image file. If a file name is specified, form removal will be activated. It is strongly recommended that registration be used if form removal is activated, and that at least three registration marks be used.

Example: FORM_REMOVAL_TEMPLATE BLANK.PCX

Related Property: OptFormRemovalFile

GRAPHICS_REMOVAL

Syntax: GRAPHICS_REMOVAL < ON | OFF >

Possible Values: ON
OFF

Valid In: ZDF, ZRF

Description: If graphics removal is on, large blobs which are considered graphics are removed. If it is off, they are converted to text blobs. The default value is ON.
This keyword is not available through NR Form Editor.

Example: GRAPHICS_REMOVAL ON

Related Property: OptRemoveGraphics

IMAGE_RESOLUTION_DPI

Syntax: IMAGE_RESOLUTION_DPI <resolution>

Possible Values: 0 (zero)
The image resolution in DPI

Valid In: ZDF, ZRF

Description: This keyword specifies the resolution at which the image was scanned. This is required if any of the Data Zones specify a non-zero value for the **N_CHARS_PER_INCH** setting.

Example: IMAGE_RESOLUTION_DPI 300

Related Property: ICRImageResolution

LINE_TYPE

Syntax: LINE_TYPE < label > < list_of_word_labels >

Possible Values: See Appendix C : Zone Syntax Definition

Valid In: ZDF, ZRF

Description: This is an identifier tag that will be used to refer to the syntax for a line of text and the line syntax definition. Please refer to **Appendix B: Defining Syntax for a Zone** for more information.

Example: LINE_TYPE ADDRESS_LAST (CITY) STATE ZIP_5|ZIP_9

Related Property: None (Can be set through **SetZoneParam** method)

LONG_LINE_LENGTH

Syntax: LONG_LINE_LENGTH <length>

Possible Values: Integers >= 0

Valid In: ZDF, ZRF

Description: This specifies the maximum length (in pixels) a line on the image can attain before being classified as a "long" line when **REMOVE_LINES** is ON. The default value is 75 pixels if this keyword does not appear in the **ZDF** file.
A valid value is an integer indicating the maximum length of a line to be recognized. This should be greater than the maximum expected size (in pixels) of the largest character in the image.

Example: LONG_LINE_LENGTH 75

Related Property: OptLineMinLength

MAX_ID_ERROR

Syntax: MAX_ID_ERROR< DISTANCE >

Possible Values: 0.0 to 15.0

Valid In: ZDF, ZRF

Description: This setting is the amount of mismatch between the features of a form identification mark and the closest matching blob in the expected location of the form identification mark. This setting stops the form identification code from simply guessing at form identification marks when there is nothing in the region which actually looks like the expected mark. The default value is
This keyword is not available through NR Form Editor.

Example: MAX_ID_ERROR 1

Related Property: None (Can be set through **SetZoneParam** method)

MAX_LINE_GAP

Syntax: MAX_LINE_GAP<distance>

Possible Values: Integer >= 0

Valid In: ZDF, ZRF

Description: This specifies the maximum size (in pixels) of a gap that can be spanned when removing long lines. Lines that are within this distance of "long" lines (as defined by **LONG_LINE_LENGTH**) will be removed when **REMOVE_LINES** is ON. This allows the detection of long lines with small breaks in them. The default value is 3.

Example: MAX_LINE_GAP 2

Related Property: OptMaxLineGap

MAX_NONLINEAR_REG_ERROR

Syntax: MAX_NONLINEAR_REG_ERROR< PIXELS >

Possible Values: Integer, 0 to 50

Valid In: ZDF, ZRF

Description: This setting specifies the maximum error which can be tolerated when registration is to occur. The registration error is the amount of difference (either x or y) between the expected location of a registration mark on the template and the location where the mark is found on the input document after all of the deskewing, scaling, and translation information has been applied to the input document. This error is caused by non-linear distortions in both the input image and the template. The default value is 20.

This keyword is not available through NR Form Editor.

Example: MAX_NONLINEAR_REG_ERROR 20

Related Property: None (Can be set through **SetZoneParam** method)

MAX_REG_ERROR

Syntax: MAX_REG_ERROR< DISTANCE >

Possible Values: 0.0 to 15.0

Valid In: ZDF, ZRF

Description: This setting is the amount of mismatch between the features of a registration mark and the closest matching blob in the expected location of the registration mark. This setting stops the registration code from simply guessing at registration marks when there is nothing in the region which actually looks like the expected mark. The default value is 5.

This keyword is not available through NR Form Editor.

Example: MAX_REG_ERROR 5

Related Property: None (Can be set through **SetZoneParam** method)

MAX_REG_ID_ZONES

Syntax: MAX_REG_ID_ZONES< INTEGER >
Possible Values: Integer, 0 to 15
Valid In: ZDF, ZRF
Description: This setting is the maximum total number of registration and form identification zones. The default value is 5.
This keyword is not available through NR Form Editor.
Example: MAX_REG_ID_ZONES 15
Related Property: None (Can be set through **SetZoneParam** method)

MAX_REG_OFFSET

Syntax: MAX_REG_OFFSET< PIXELS >
Possible Values: Integer, 0 to 500
Valid In: ZDF, ZRF
Description: This setting is the size of the region around the expected locations of registration or form identification marks to search for the marks. This is also the maximum number of pixels that an image can be moved during the registration process. The default value is 100.
This keyword is not available through NR Form Editor.
Example: MAX_REG_OFFSET 100
Related Property: None (Can be set through **SetZoneParam** method)

MIN_LINE_SPLIT_HEIGHT

Syntax: MIN_LINE_SPLIT_HEIGHT< PIXELS >
Possible Values: Integer, 0 to 300
Valid In: ZDF, ZRF
Description: This setting is the minimum height of a line which will have line splitting attempted. Line splitting works in the multiple line segmentation mode and attempts to make a horizontal split between 2 or more lines which have been incorrectly connected due to the excursion of one or more characters from one line to the other. For example, the bottom of a “y” dropping into some characters in the line below. The default value is 5.
This keyword is not available through NR Form Editor.
Example: MIN_LINE_SPLIT_HEIGHT 75
Related Property: ICRMinLineSplit

MIN_MATCHED_ID_MARKS

- Syntax:** MIN_MATCHED_ID_MARKS< INTEGER >
- Possible Values:** Integer, 0 to 15
- Valid In:** ZDF, ZRF
- Description:** This setting is the minimum number of form identification marks which need to be found and still allow processing to continue. For example, if 2 form identification marks are defined in the .ZDF file, the user may specify that only 1 needs to be located to allow processing of the image to continue. If this setting to 0, form identification will be attempted but fail, and processing of the image will continue. Setting the DEFAULT_DOCUMENT_TYPE TRUE will override this setting. The default value is 15.
This keyword is not available through NR Form Editor.
- Example:** MIN_MATCHED_ID_MARKS 15
- Related Property:** None (Can be set through **SetZoneParam** method)

MIN_MATCHED_REG_MARKS

- Syntax:** MIN_MATCHED_REG_MARKS< INTEGER >
- Possible Values:** Integer, 0 to 15
- Valid In:** ZDF, ZRF
- Description:** This setting is the minimum number of registration marks which need to be found and still allow registration to continue. For example, if 6 registration marks are defined in the .ZDF file, the user may specify that only 3 need to be located to allow processing of the image to continue. If this setting to 0, registration will be attempted but fail, and processing of the image will continue. The default value is 5.
This keyword is not available through NR Form Editor.
- Example:** MIN_MATCHED_ID_MARKS 15
- Related Property:** None (Can be set through **SetZoneParam** method)

MIN_OMR_FILL_STD

- Syntax:** MIN_OMR_FILL_STD<pixels>
- Possible Values:** Integer >= 0, typically less than 10
- Valid In:** ZDF, ZRF
- Description:** This is the percentage of the OMR box that must be filled before it will be seen as “on” or “checked.” WEIGHTED_CENTER must be FALSE for this setting to be used. The default value is 5.

This keyword is not available through NR Form Editor.

Example: MIN_OMR_FILL_STD 5

Related Property: None (Can be set through **SetZoneParam** method)

MIN_OMR_FILL_WT

Syntax: MIN_OMR_FILL_WT<pixels>

Possible Values: Integer ≥ 0 , typically less than 10

Valid In: ZDF, ZRF

Description: This is the center weighted percentage of the OMR box that must be filled before it will be seen as “on” or “checked.” This percentage is in addition to the center weighted percentage that was generated when the OMR box was created. WEIGHTED_CENTER must be TRUE for this setting to be used. The default value is 4.

This keyword is not available through NR Form Editor.

Example: MIN_OMR_FILL_WT 4

Related Property: OMRFillForChecked

NOISE_AREA

Syntax: NOISE_AREA<pixels>

Possible Values: Integer ≥ 0 , typically less than 20

Valid In: ZDF, ZRF

Description: Any speck or character in the image with an area in black pixels less than the given value will be interpreted as noise. The default value is 20.

Example: NOISE_AREA 30

Related Property: OptSpeckMaxArea

NOISE_HEIGHT

Syntax: NOISE_HEIGHT<pixels>

Possible Values: Integer ≥ 0 , typically less than 10

Valid In: ZDF, ZRF

Description: Any speck or character in the image with a height in pixels less than the given value will be interpreted as noise. The default value is 4.

Example: NOISE_HEIGHT 3

Related Property: OptSpeckHeight

NOISE_WIDTH

Syntax: NOISE_WIDTH<pixels>
Possible Values: Integer ≥ 0 , typically less than 10
Valid In: ZDF, ZRF
Description: Any speck or character in the image with a width in pixels less than the given value will be interpreted as noise. The default value is 4.
Example: NOISE_WIDTH 3
Related Property: OptSpeckWidth

OMR_BORDER_SIZE

Syntax: OMR_BORDER_SIZE<pixels>
Possible Values: Integer ≥ 0 , typically less than 10
Valid In: ZDF
Description: This is the size (in pixels) of the white space that is added on all sides of the blob found when detecting OMR boxes (i.e. when they are defined). This border size is added to the boundaries of the shrink wrapped coordinates of the boxes. It has no effect during recognition. WEIGHTED_CENTER must be TRUE for this setting to be used. The default value is 5.
Note: If OMR_BORDER_SIZE = -1, then the original BorderWidth calculation is used: $\text{BorderWidth} = 0.15 * \max(\text{OmrWidth}, \text{OmrHeight})$; if $\text{BorderWidth} > 15$ then $\text{BorderWidth} = 15$; if $\text{BorderWidth} < 3$ then $\text{BorderWidth} = 3$. This mode is provided for backward compatibility only, and would rarely be used.
This keyword is not available through NR Form Editor.
Example: OMR_BORDER_SIZE 5
Related Property: None (Can be set through **SetZoneParam** method)

ORIGINAL_IMAGE_NAME

Syntax: ORIGINAL_IMAGE_NAME<NONE | pathname>
Possible Values: Path and file name of the source image file
Valid In: ZRF
Description: This keyword specifies the name of the image which was processed to produce the **ZRF** file. The value will reflect the image file name; if the image was supplied in RAM and thus had no valid file name, the value will be **NONE**. This is for informational purposes only.
Example: ORIGINAL_IMAGE_NAME C:\IMAGES\FORM0001.TIF

Related Property: None

ORIGINAL_IMAGE_SIZE

Syntax: ORIGINAL_IMAGE_SIZE<width> <height>

Possible Values: Two integers >= 0

Related Property: None (Can be set through **SetZoneParam** method)

Valid In: ZDF, ZRF

Description: This specifies the coordinate system for this **.ZDF** or **.ZRF** file by giving the width and height of the form in arbitrary units.

Example: ORIGINAL_IMAGE_SIZE 850 1100

Related Property: None (Can be set through **SetZoneParam** method)

OUTPUT_IMAGE_NAME

Syntax: OUTPUT_IMAGE_NAME<NONE | pathname>

Possible Values: NONE

A path name

A path and filename

Valid In: ZDF, ZRF

Description: This keyword specifies either the path where the output image will be stored (in the **ZDF** file), or the full pathname where the output image was stored (in the **ZRF** file). The output file will be in the TIFF (**TIF**) format. An error will be produced if the pathname is the same as that of the original image. This prevents overwriting the original image. A value of **NONE** indicates that no output file is to be generated.

Example (ZDF): OUTPUT_IMAGE_NAME C:\IMAGES\SAVE

Example (ZRF): OUTPUT_IMAGE_NAME C:\IMAGES\SAVE\FORM0001.TIF

Related Property: None

PUNCTUATION_MEMORY

Syntax: PUNCTUATION_MEMORY< ON | OFF >

Possible Values: ON

OFF

Valid In: ZDF, ZRF

Description: This turns on or off the use of a special neural network to identify punctuation marks. If turned off, it is likely that no punctuation will be recognized as such in hand print fields, though punctuation may be recognized in machine print fields. The default is **ON**.

Example: PUNCTUATION_MEMORY ON

Related Property:

REGISTRATION

Syntax: REGISTRATION<ON | OFF>

Possible Values: ON
OFF

Valid In: ZDF, ZRF

Description: This keyword specifies whether the software should perform image registration and deskew during processing. This should always be **ON** when performing form removal

Example: REGISTRATION OFF

Related Property:

REMOVE_LINES

Syntax: REMOVE_LINES<ON | OFF>

Possible Values: ON
OFF

Valid In: ZDF, ZRF

Description: This keyword toggles on and off the long-line removal processing.

Example: REMOVE_LINES ON

Related Property:

RESULTS_FORMAT

Syntax: RESULTS_FORMAT<CSV | ZRF | TXT [delimiter]>

Possible Values: CSV
ZRF
TXT

Valid In: ZDF, ZRF

Description: TXT allows an additional value in quotes to determine the delimiter character to use between words; the default delimiter is a space.

CSV (Comma Separated Value) is a file format which contains the names of the data zones on the first line, with the next line or lines containing the complete recognition results for those zones from one or more images. This is a file format commonly used for import into databases.

Example: RESULTS_FORMAT ZRF

RESULTS_FORMAT TXT " , "
RESULTS_FORMAT TXT

Related Property: OutputFormat

SMART_NOISE_CLASSIFICATION

Syntax: SMART_NOISE_CLASSIFICATION< ON | OFF >

Possible Values: ON
OFF

Valid In: ZDF, ZRF

Description: When turned on, this parameter allows things that were initially seen as noise to be interpreted as possibly being part of a fragmented or broken character. This is useful if the image has many fragmented characters. The default is **ON**.

Example: SMART_NOISE_CLASSIFICATION ON

Related Property: ICRSmartNoiseClassification

VERIFY_RESULTS_FORMAT

Syntax: VERIFY_RESULTS_FORMAT<format>

Possible Values: TXT
ZRF
CSV

Valid In: ZDF, ZRF

Description: This specifies the format in which NR Verifier is to write out the results of verification. Since NR Verifier only accepts **ZRF** files as input, this allows a different file format to be specified as NR Verifier's output for further processing. The default is **ZRF**.

Example: VERIFY_RESULTS_FORMAT CSV

Related Property:

WEIGHTED_CENTER

Syntax: WEIGHTED_CENTER<TRUE | FALSE>

Possible Values: TRUE
FALSE

Valid In: ZDF, ZRF

Description: This setting should be true if the amount that is filled is more towards the center of the box ("weighted" towards the box's center), such as when a "X" is used. When this setting is true, then the global values for MIN_OMR_FILL_WT and

OMR_BORDER_SIZE will be used. If this setting is false, then the global value MIN_OMR_FILL_STD will be used.

This keyword is not available through NR Form Editor.

Example: WEIGHTED_CENTER TRUE

Related Property: OMRUseWeightedCenter

WORD_TYPE

Syntax: WORD_TYPE< list of constraints >

Possible Values: See Defining Zone Syntax

Valid In: ZDF, ZRF

Description: This is an identifier tag that will be used to refer to the syntax for a line of text and the word syntax definition. Please refer to "Document Definition" on page 16 for more information.

ZDF_FILENAME

Syntax:

Possible Values: NONE

Fully qualified path and file name

Valid In: ZDF, ZRF

Description: This is used to identify the **ZDF** file which was used during recognition. If the **ZDF** structure was presented in RAM instead of being read from a file, this field contains the value **NONE**. This is most useful when form identification is being used, as it helps indicate the form which was recognized.

Example: ZDF_FILENAME C:\NR\IRS1040.ZDF

Related Property:

ZDF Data Zone Definitions

A Data Zone defines an area on the image where recognition is to be performed on text. In the case of a ZRF file Data Zone, information related to the results is presented in addition to the other settings for the Data Zone.

The start of a Data Zone is indicated by the keyword

START_DATA_ZONE

The end of a Data Zone is indicated by the keyword

END_DATA_ZONE

In a typical ZDF or ZRF file, multiple Data Zones will be defined using these keywords. Between the two end markers, the following keywords may be found

to define each zone. Unless otherwise noted in the description, all keywords are valid in both ZDF and ZRF files.

Most of these options may be set in either the Form Editor utility or through properties as described in 'Zone Definition' on page 20. Those keywords that indicate that they do not have an equivalent property and are not accessible through NR Form Editor can still be accessed. The **GetZoneParam** and **SetZoneParam** methods may be used, or the ZDF file may be opened in any text editor.

The following keywords may be found in a Data Zone:

ALTERNATE_IDS
BOTTOM_RIGHT_X
BOTTOM_RIGHT_Y
CHARACTER_FATTENING_INCREMENT
CHARACTER_REPAIR_THRESHOLD
CONFIDENCE_THRESHOLD
CONSTRAINT
CONTEXT
DICTIONARY
END_RESULTS
ENTRY_REQUIRED
FIND_PUNCTUATION
LABEL
MIN_CHARS
N_CHARS_PER_INCH
PRINT_TYPE
REVIEW
SEGMENTATION
START_RESULTS
SYNTAX_MATCH_REQUIRED
TOP_LEFT_X
TOP_LEFT_Y
ZONE_SYNTAX_START
ZONE_SYNTAX_END

ALTERNATE_IDS

Syntax: ALTERNATE_IDS <0 | 1 | 2>
Possible Values: 0
1
2
Description: This keyword defines how many alternative identifications should be included in the recognition output; if used, it must be followed by a numeral in the range 0-2. This value is ignored if the **RESULTS_FORMAT** is **TEXT** or **CSV**, as that format only allows the output of the most likely character classifications.
Example: ALTERNATE_IDS 2
Related Property: AltChars

BOTTOM_RIGHT_X, BOTTOM_RIGHT_Y, TOP_LEFT_X, TOP_LEFT_Y (Data Zone)

Syntax: BOTTOM_RIGHT_X <pixel location>
BOTTOM_RIGHT_Y <pixel location>
TOP_LEFT_X <pixel location>
TOP_LEFT_Y <pixel location>
Possible Values: Integer >= 0
Description: These keywords are used to define the location (in the image coordinate system as defined by the settings for **ORIGINAL_IMAGE_SIZE** relative to the upper left corner of the image) of a zone.
Example: TOP_LEFT_X 100
TOP_LEFT_Y 50
BOTTOM_RIGHT_X 300
BOTTOM_RIGHT_Y 150
Related Property: BOTTOM_RIGHT_X → ZoneRight
BOTTOM_RIGHT_Y → ZoneBottom
TOP_LEFT_X → ZoneLeft
TOP_LEFT_Y → ZoneTop

CHARACTER_FATTENING_INCREMENT

Syntax: CHARACTER_FATTENING_INCREMENT <increment>
Possible Values: Integer 1 to 3
Description: This setting determines the amount in pixels by which characters will be dilated (or fattened) during character repair. This value is ignored unless character repair is being performed.

Example: CHARACTER_FATTENING_INCREMENT 2

Related Property: None

CHARACTER_REPAIR_THRESHOLD

Syntax: CHARACTER_REPAIR_THRESHOLD <threshold>

Possible Values: Integer 0 to 100

Description: This setting controls how often characters will be interpreted as needing repair. The threshold is compared to a calculated value that indicates how fragmented the characters in an image are, and repair will be done if the fragmentation metric exceeds the threshold. A threshold of 0 means that character repair will never be done, while a threshold of 100 means that character repair will always be done. The default value is 0 (off).

Example: CHARACTER_REPAIR_THRESHOLD 75

Related Property: None

CONFIDENCE_THRESHOLD

Syntax: CONFIDENCE_THRESHOLD < threshold >

Possible Values: Integer, 0 to 100

Description: Recognized characters with confidence values below this value may be replaced with a character to denote uncertainty of recognition. This value is used by NR Verifier to determine whether a zone requires human correction.

Example: CONFIDENCE_THRESHOLD 90

Related Property: None

CONSTRAINT

Syntax: CONSTRAINT < constraint label | NO_RECOGNITION >

Possible Values: Any previously-defined (using the **DEFINE_CONSTRAINT** keyword) logical constraint label
The keyword **NO_RECOGNITION** .

Description: This keyword defines the logical constraint of characters within a field. When a zone syntax is defined, it will override any setting that is given for **CONSTRAINT** unless its value is **NO_RECOGNITION**. This special keyword value, **NO_RECOGNITION** may be used to indicate that the zone is only defined for the purposes of including that zone's image contents in the output image file.

Example: CONSTRAINT8

CONSTRAINT NO_RECOGNITION

Related Property: None

CONTEXT

Syntax: CONTEXT <ON | OFF>

Possible Values: ON
OFF

Description: This keyword defines whether context checking is to be used for a zone. Context checking uses information about the probability of one letter following another in order to help guide recognition of characters. The English language is used as a basis for these probabilities unless **SPECIAL** dictionary is being used.

Example: CONTEXT ON

Related Property: UseContext

DICTIONARY

Syntax: DICTIONARY <OFF | ASSIST <path> | CLOSEST_MATCH <path>>

Possible Values: OFF
ASSIST
CLOSEST_MATCH

Description: This keyword specifies whether a dictionary should be used to aid recognition of words in a zone. This keyword accepts either 1 or 2 additional values. The keyword **DICTIONARY** must always be followed by either **OFF**, **ASSIST**, or **CLOSEST_MATCH**. If it is followed by **ASSIST** or **CLOSEST_MATCH** then that keyword must be followed by the pathname of the dictionary to be used. If the pathname does not exist, NestorReader OT API will search for the dictionary named in the current directory and in **<NESTORPATH>\LIB**. The **ASSIST** keyword specifies that the dictionary will assist recognition, but that the results will not be forced to match a word in the dictionary. The **CLOSEST_MATCH** keyword specifies that the output word must be in the dictionary.

Example: DICTIONARY ASSIST ENG001.DIC

Related Property: ZoneDictionary

END_RESULTS

Description: See **START_RESULTS**

ENTRY_REQUIRED

Syntax: ENTRY_REQUIRED<ON | OFF>

Possible Values: ON
OFF

Description: This keyword specifies whether an entry is required in this field, i.e., whether it is valid for the number of characters to be 0. This is included for use in verification of the output. If this value is set, NR Verifier will display the zone if there is no entry.

Example: ENTRY_REQUIRED ON

Related Property:

FIND_PUNCTUATION

Syntax: FIND_PUNCTUATION<TRUE | FALSE>

Possible Values: TRUE
FALSE

Description: This setting controls whether or not periods and commas are allowed in the results.

Example: FIND_PUNCTUATION TRUE

Related Property:

LABEL (Data Zone)

Syntax: LABEL <zone_name>

Possible Values: Any printable character string with no spaces

Description: This keyword specifies a name for the zone. This is required. No spaces are allowed in the label.

Example: LABEL First_Name

Related Property: ZoneName

MIN_CHARS

Syntax: MIN_CHARS<INTEGER>

Possible Values: Integer, 0 to 500

Description: This setting specifies the minimum number of expected characters in a zone. If this number of characters is not found, the Recognition Server first searches to the left of the zone and then to the right of the zone for the missing characters. The search stops in each direction when a space big enough to be considered a word break is found. It is an error to return less than MIN_CHARS characters for a zone. When MIN_CHARS

is used, the zone's segmentation type must be SINGLE_LINE or SINGLE_WORD.

This keyword is not available through NR Form Editor.

Example: MIN_CHARS 25

Related Property:

N_CHARS_PER_INCH

Syntax: N_CHARS_PER_INCH<CPI [COMB]>

Possible Values: Any non-negative floating point value

Description: This setting specifies the number of characters-per-inch (**CPI**) in the zone. If this value is non-zero, it will override the segmentation type and mode settings. Non-zero settings should be used only when there is certain information about the number of characters-per-inch in the image. The setting may optionally be followed by the word **COMB** if the left edge of the Data Zone is known to coincide with the beginning of the first character box.

Example: N_CHARS_PER_INCH 0.0
N_CHARS_PER_INCH 3.5
N_CHARS_PER_INCH 5.0 COMB

Related Property: ZoneCharsPerInch

PRINT_TYPE

Syntax: PRINT_TYPE<MP | HP | MH>

Possible Values: MP
HP
MH

Description: This is used to identify the type of print which was found in the zone during recognition. Its value will be either **MP** (machine print), **HP** (machine print), or **MH** (mixed machine print and hand print).

Example: PRINT_TYPE HP

Related Property:

REVIEW

Syntax: REVIEW <ALWAYS | AS_NEEDED | NEVER>

Possible Values: ALWAYS
AS_NEEDED
NEVER

Description: This keyword defines whether this field is sufficiently important that a human should always check it. This is mainly for use in data verification. A value of **ALWAYS** indicates that a human must always review the contents of the field. A value of **AS_NEEDED** indicates that a human only needs to look at the field if the recognition was not confident enough of one or more characters in the field. A value of **NEVER** indicates that a human never needs to review the recognition results for the field. The default is as defined in your **NESTOR.INI** file; initially, the default is **AS_NEEDED**.

Example: REVIEW AS_NEEDED

Related Property: None

SEGMENTATION

Syntax: SEGMENTATION <segment_type> <segment_mode>

Possible Values: See Description

Description: This keyword defines the segmentation type and mode to be used for a zone or zones. Both a segmentation type and a segmentation mode must be specified.

<segment_type> may one of the following values:

SINGLE_CHARACTER

only a single character is expected in the zone

SINGLE_WORD

all characters in the zone are expected to form a single word

SINGLE_LINE

potentially multiple words on a single line of text

MULTI_LINE

multiple words on multiple lines of text are expected

<segment_mode> may have one of the following values:

MIN_SEGMENTATION

STD_SEGMENTATION

MAX_SEGMENTATION

Each mode implies that more processing time will be devoted to separating characters than in the previous mode.

Example: SEGMENTATION SINGLE_LINE STD_SEGMENTATION

Related Property:

START_RESULTS, END_RESULTS

Syntax: START_RESULTS
END_RESULTS

Possible Values: None

Description: These keywords delimit the results block which contains the results of recognition for a Data Zone.

Example:

```
START_RESULTS
nLines 2  nChars 9
NCSINC
ONE
      N  76   47   36  31  34
      W   3
      D   0
      C  94   92   44  22  27
      8   5
      Q   0
      S  99  141   45  17  22
      C   0
      X   0
      I  80  367   38  43  26
      Z  19
      T   0
      N  73  413   47  21  20
      I  25
      R   0
      C  99  453   46  16  19
      6   0
      L   0

      O  99  169   99  35   32
      O   0
      C   0
      N  88  207  112  25  23
      R  10
      M   0
      E  87  251  107  21  25
```

```
O 10
6 1
```

END_RESULTS

On the line after the **START_RESULTS** keyword, the statistics for the results are reported. The statistics line has the format:

```
nLines X nChars Y
```

The value **X** is the number of lines found in the zone's recognition results, and **Y** is the total number of characters found in the zone's results. The value **Y** includes the blanks between words in the results, but does not include any breaks between lines or any blanks before or after words on a line.

Next in the results block is a blank line. Following the blank line are **X** lines of text. Each line of text is an ASCII string which is constructed by taking the most likely classification for each character in that line of text, with blanks inserted as appropriate.

Another blank line follows the lines of text. Following this blank line are the results for each character. The number of alternate identifications is whatever value is set for the current data zone using the **ALTERNATE_IDS** functional keyword; in our example, that value is 2, so there are two alternative identifications supplied for each character. The first (and only, if there are 0 alternative identifications requested) entry for each character consists of the character class, the confidence value for the identification, the coordinates of the bounding box of the character relative to the zone, and the width and height of the character. Consider the entry:

```
N 97 569 108 26 21
```

The character class is **N**, the confidence value is **97**, its bounding box top left corner is **569** pixels to the right of the zone's origin and **108** pixels down from it, and the character's bounding box is **26** pixels wide by **21** pixels high.

If there are any alternative identifications, they are each on a separate line, and they consist solely of the alternative class and its confidence value. In this case, the complete entry for the character is:

```
N 97 569 108 26 21
I 1
2 0
```

The first alternative has a class of **1**, and a confidence value of **1**; the second alternative has a class of **2**, and a confidence value of **0**.

Blanks within a line are indicated by a single blank line. The separation between lines is indicated by two successive blank lines.

SYNTAX_MATCH_REQUIRED

Syntax: SYNTAX_MATCH_REQUIRED <TRUE | FALSE>

Possible Values: TRUE
FALSE

Description: This keyword specifies whether the words and lines in a zone must exactly match the syntax. If this setting is **TRUE**, only words and lines which have an exact match to the syntax will be allowed. All others will be output as *****s.

This keyword is not available through NR Form Editor.

Example: SYNTAX_MATCH_REQUIRED FALSE

Related Property:

TOP_LEFT_X, TOP_LEFT_Y (Data Zone)

Description: See BOTTOM_RIGHT_X

ZONE_SYNTAX_START, ZONE_SYNTAX_END

Syntax: ZONE_SYNTAX_START
ZONE_SYNTAX_END

Possible Values: None

Description: The lines of text between these keywords define the syntax to be used in processing a data zone. All entries between the **ZONE_SYNTAX_START** keyword and the next **ZONE_SYNTAX_END** keyword will be interpreted as line syntax identifier tags (see the **LINE_TYPE** Global Definitions keyword above) which define the syntax for a zone.

Related Property: None

ZDF OMR Zone Definitions

An OMR Zone defines an area on the image where recognition is to be performed on check boxes. In the case of an OMR Zone in a ZRF file, the information present will also represent the results of recognition.

The start of an OMR Zone is indicated by the keyword

START_OMR_ZONE

The end of an OMR Zone is indicated by the keyword

END_OMR_ZONE

All OMR Zone keywords are valid in both **ZDF** and **ZRF** files.

The following keywords are allowed in an OMR zone:

BOTTOM_RIGHT_X

BOTTOM_RIGHT_Y

TOP_LEFT_X

TOP_LEFT_Y

LABEL

OMR_GROUP_START

OMR_GROUP_END

OMR_NUM_BOXES

OMR_STRINGS_START

OMR_STRINGS_END

OMR_TYPE

BOTTOM_RIGHT_X, BOTTOM_RIGHT_Y, TOP_LEFT_X, TOP_LEFT_Y (OMR Zone)

Syntax: BOTTOM_RIGHT_X <pixel location>
 BOTTOM_RIGHT_Y <pixel location>
 TOP_LEFT_X <pixel location>
 TOP_LEFT_Y <pixel location>

Possible Values: Integer image pixel coordinates >= 0

Description: These keywords are used to define the location (in the image coordinate system as defined by the settings for **ORIGINAL_IMAGE_SIZE** relative to the upper left corner of the image) of a zone.

Example: TOP_LEFT_X 100
 TOP_LEFT_Y 50
 BOTTOM_RIGHT_X 300
 BOTTOM_RIGHT_Y 150

Related Property: ZoneTop, ZoneBottom, ZoneLeft, ZoneRight

LABEL (OMR Zone)

Syntax: LABEL <zone_name>
Possible Values: A unique string with no embedded spaces
Description: This keyword specifies label (name) for the OMR zone.
Example: LABEL M_F_Check_Box
Related Property: ZoneName

OMR_GROUP_START, OMR_GROUP_END

Syntax: OMR_GROUP_START
...
<char> <conf> <x> <y> <w> <h>
...
OMR_GROUP_END

Description: This section is used to define the locations of the OMR boxes for this zone. The information returned from the recognition server is also stored here for use on the target form. There are one or more lines of text between the occurrences of these keywords, one for each OMR box in the zone.

Valid values:

<char> will have one of the following values 0 (not set), 1 (set), ? (uncertain), or * (error, such as multiple boxes marked in a **RADIO_GROUP** field). <char> will always have the value 0 in a **.ZDF** file.

<conf> is the percentage of the original box which was filled in the reference image. This should be ignored, as it is only meaningful to NestorReader OT API.

<x>, <y> are the coordinates of the top left corner of the OMR box.

<w>, <h> are the width and height of the OMR box.

Example: OMR_GROUP_START
0 24 81 16 16 20
0 26 97 16 18 20
0 19 66 50 13 20
OMR_GROUP_END

OMR_NUM_BOXES

Syntax: OMR_NUM_BOXES <number of OMR boxes>
Possible Values: Numeral corresponding to the number of defined OMR boxes.

Description: This value is set by the system after the OMR locations have been found. The value is stored in the **ZDF** file for each OMR group.

Example: OMR_NUM_BOXES 5

OMR_STRINGS_START, OMR_STRINGS_END

Syntax: OMR_STRINGS_START

...

<OMR label>

...

OMR_STRINGS_END

Description: These keywords indicate the beginning and end of the definition of the strings associated with the defined OMR boxes. The strings may have embedded spaces, and must be listed one per line in the same order as the OMR boxes with which they are associated in the **OMR_GROUP_START/END** section.

Example:

OMR_STRINGS_START

Maine

New York

Florida

OMR_STRINGS_END

OMR_TYPE

Syntax: OMR_TYPE <setting>

Possible Values: RADIO_GROUP
MULTIPLE_CHOICE

Description: This keyword specifies whether the OMR boxes in this zone should be treated as a multiple choice group or as radio buttons. A multiple choice group may have zero or more boxes checked. A group of radio buttons must have one and only one button checked.

Example: OMR_TYPE RADIO_GROUP

ZDF Registration Zone Definitions

Each registration zone begins with the keyword

START_REGISTRATION_ZONE

Each registration zone ends with the keyword

END_REGISTRATION_ZONE

The following keywords may be found in a Registration Zone definition.

BOTTOM_RIGHT_X
BOTTOM_RIGHT_Y
TOP_LEFT_X
TOP_LEFT_Y
FEATURES
LABEL
REGISTRATION_STRING

BOTTOM_RIGHT_X, BOTTOM_RIGHT_Y, TOP_LEFT_X, TOP_LEFT_Y (Registration Zone)

Syntax: BOTTOM_RIGHT_X <pixel location>
BOTTOM_RIGHT_Y <pixel location>
TOP_LEFT_X <pixel location>
TOP_LEFT_Y <pixel location>

Possible Values: Integer image pixel coordinates ≥ 0

Description: These keywords are used to define the location (in the image coordinate system as defined by the settings for **ORIGINAL_IMAGE_SIZE** relative to the upper left corner of the image) of a zone.

Example: TOP_LEFT_X 100
TOP_LEFT_Y 50
BOTTOM_RIGHT_X 300
BOTTOM_RIGHT_Y 150

Related Property: ZoneTop, ZoneBottom, ZoneLeft, ZoneRight

LABEL (Registration Zone)

Syntax: LABEL <zone_name>

Possible Values: A unique string with no embedded spaces

Description: This keyword specifies a label (name) for the OMR zone.

Example: LABEL M_F_Check_Box

Related Property: ZoneName

FEATURES

Syntax: FEATURES <f1> <f2> <f3> <f4> <f5> <f6> <f7> <f8> <f9>

Description: This keyword specifies the features for a registration mark
These values are generated automatically by NestorReader OT

API when OT_AddZDFZone() is called, or when a Registration zone is defined graphically in the NR Form Editor utility. The FEATURES keyword is followed by 9 floating point values. If this keyword is present, the REGISTRATION_STRING keyword is excluded. The FEATURES keyword may not be used in calls to OT_GetZDFValue() or OT_SetZDFValue().

Example:
FEATURES 0.32 1.34 3.21 5.42 0.43 0.9 0.0 0.1 1.0

REGISTRATION_STRING

Syntax: REGISTRATION_STRING <string>
Description: This keyword specifies a string which is to be used for registration. This string is generated by NestorReader OT API when the Registration zone is created, and is not supplied by the application. If this keyword is present, the keyword FEATURES is excluded.
Example: REGISTRATION_STRING 1040EZ

Index

A

- AboutBox Method 41
- Active Property 41
- Active Property, Using 12
- ALLOW_ID_USING_BLOBS 103
- alternate identifications 120, 127
- ALTERNATE_IDS** 127
- ALWAYS** 125
- anhm0096.me2 16
- Annotation Control 9
- ansm0087.me2 16
- AS_NEEDED** 125
- ASSIST 26, 123

B

- BLOB_MEMORY 104
- BOTTOM_RIGHT_X 120, 129, 132
- BOTTOM_RIGHT_Y 120, 121, 129, 132

C

- Cancel Parameter 56
- character coordinates 127
- character repair 121
- CHARACTER_FATTENING_INCREM
ENT 121
- CHARACTER_REPAIR_THRESHOLD
121
- CLOSEST_MATCH** 123
- COMB** 124
- Completion Event Parameters
 - Cancel 56
 - Percent 56
- confidence
 - threshold 5
 - value 127
 - value 5
- confidence threshold 122
- CONFIDENCE_THRESHOLD 121
- CONSTRAINT 28, 122
- CONTEXT 28, 122
- context checking 122
- Control Communication 8
- coordinate system 115
- CPI** 124

D

- Database, Creation of Licensing 11
- Database, Licensing 10
- DEFAULT_DOCUMENT_TYPE 104
- Define a Data Zone 20, 22
- Define a Registration Zone 15, 20
- Define an OMR Zone 20, 21
- DEFINE_CONSTRAINT 28, 30, 104, 122
- deskewing See registration
- DESKEWING Keyword 105
- DICTIONARY 28, 122
 - special 122
- Dictionary Types
 - Normal 33
 - Special 33
- DROP_OUT_INK 106
- drop-out ink 106

E

- edit buffer 13
- END_OMR_ZONE 129
- END_RESULTS 126
- ENTRY_REQUIRED 123
- Error Event 45
- Event Parameters
 - Cancel 56
 - Percent 56
- Events
 - Error 45
 - Progress 56

F

- fax 3
- FEATURES** 133
- File Output
 - OutputFileName Property 55
 - OutputFormat Property 55
 - OutputTo Property 56
- FIND_PUNCTUATION 123
- FLAG_UNCERTAIN_CHAR_MAX_H
EIGHT 106
- FLAG_UNCERTAIN_CHAR_MIN_OV
ERLAP 107
- FLAG_UNCERTAIN_CHAR_REPLAC
EMENT 108

FLAG_UNCERTAIN_CHAR_SPLIT_O
VERLAP 108

form

identification 4

form identification 119

Form Removal 15, 108, 116

FORM_REMOVAL_TEMPLATE 108

fragmented characters 117

G

global definitions 102

global keywords 102

Global Properties. *See* Image Properties

GRAPHICS_REMOVAL 109

Guidelines for Forms Design 7

H

HP 28, 105, 125

I

ICR Results

Result Property 57

image

preprocessing 4

image coordinate system 115, 121, 130,
132

image resolution 109

IMAGE_RESOLUTION_DPI 109

ImageDataSource 8, 9

ImageDataSource Property 46

L

LABEL 123, 130, 132

Lazy Initialization 12

LCM, Using 11

License Storage 10

License Verification 11

Licensing

Active Property 41

Line Syntax 109

LINE_TYPE 31, 129

Link ID 9

Link property 9

Linking

ImageDataSource Property 46

Linking Controls 8, 9

Load Time Improvement

Active Property 41

Logical Constraint 16, 105, 122. *See*

Character Constraint

Long Line Removal 110, 117

LONG_LINE_LENGTH 109, 110

M

MAX_ID_ERROR 110

MAX_LINE_GAP 110

MAX_NONLINEAR_REG_ERROR
111

MAX_REG_ERROR 111

MAX_REG_ID_ZONES 111

MAX_REG_OFFSET_ 112

MAX_SEGMENTATION 126

Method 47, 61

Methods

AboutBox 41

OCRPage 47

MH 28, 105, 125

MIN_CHARS 124

MIN_LINE_SPLIT_HEIGHT_ 112

MIN_MATCHED_ID_MARKS 112

MIN_MATCHED_REG_MARKS 113

MIN_OMR_FILL_STD 113

MIN_OMR_FILL_WT 113

MIN_SEGMENTATION 126

MP 28, 105, 125

MULTI_LINE 126

multiple choice 131

MULTIPLE_CHOICE 131

N

N_CHARS_PER_INCH 109, 124

NESTOR.INI 125

NEVER 125

NO_RECOGNITION 122

noise removal 4, 15, 114

NOISE_AREA 114

NOISE_HEIGHT 114

NOISE_WIDTH 114

Normal Dictionary 33

NR Verifier 118, 122, 123

O

OCRPage Method 47

OMR boxes 106, 130

OMR strings 131

OMR_BORDER_SIZE 114

OMR_GROUP_END 130

OMR_GROUP_START 130

OMR_NUM_BOXES 131

OMR_STRINGS_END 131

OMR_STRINGS_START 131
 OMR_TYPE 131
 ORIGINAL_IMAGE_NAME 115
 ORIGINAL_IMAGE_SIZE 115, 121,
 130, 132
 OT_AddZDFZone 133
 output formats 5
 CSV 5
 text 5
 output image 116
 OUTPUT_IMAGE_NAME 116
 OutputFileName Property 55
 OutputFormat Property 55
 OutputTo Property 56

P

Percent Parameter 56
 Print Type 16, 28, 105, 125
 PRINT_TYPE 124
 Progress Event 56
 Properties
 Active 41
 ImageDataSource 46
 OutputFileName 55
 OutputFormat 55
 OutputTo 56
 Result 57
 Property 44, 46, 48, 49, 50, 51, 52, 53,
 54, 55, 62
 punctuation 116, 123
 PUNCTUATION_MEMORY 116

R

radio buttons 131
 RADIO_GROUP 131
 Recognition
 OCRPage Method 47
 Recognition Memory 16, 28, 105
 Recognition Output 56
 Result Property 57
 recognition results 126
 Registration 4, 15, 16, 108, 116
 registration and deskew 116
 registration mark 133
 registration string 133
 REGISTRATION_STRING 133
 Releasing License Tokens 12
 REMOVE_LINES 110, 117
 Result Property 57
 RESULTS_FORMAT 117, 120

REVIEW 125
 Routing Data between Controls 8
 Runtime Linking 9

S

SEGMENTATION 125
 mode 125
 type 125
SINGLE_CHARACTER 125
SINGLE_LINE 125
SINGLE_WORD 125
SMART_NOISE_CLASSIFICATION
 15, 117
 Source of Image Data 8
 Special Dictionary 33
 START_OMR_ZONE 129
 START_RESULTS 126
STD_SEGMENTATION 126
 SYNTAX_MATCH_REQUIRED 128

T

Timing of License Locking/Unlocking
 11, 12
 Timing of Licensing Verification
 Active Property 41
 TMS Display Control 8
 TMS File Control 8
 TOP_LEFT_X 120, 121, 129, 132
 TOP_LEFT_Y 121, 129, 132

U

Using License Tokens 11

V

verification 5, 118, 123, 125
 VERIFY_RESULTS_FORMAT 118

W

WEIGHTED_CENTER 118
 wild card 30
 word syntax 118
 Word Type
 defining 30
 WORD_TYPE 118

Z

ZDF buffer *See* edit buffer
 ZDF file 13
 zone 4
 OMR 129
 properties 4

Zone Definition

Data Zone 20, 22

OMR Zone 20, 21

Registration Zone 15, 20

Zone Syntax 16, 109, 118, 122

output 32

ZONE_SYNTAX_END 32, 128

ZONE_SYNTAX_START 32, 128