

Morovia Barcode DLL 4.0

Reference Manual

Morovia Barcode DLL 4.0 Reference Manual

Copyright © 2006, 2009, 2021 Morovia Corporation. All rights reserved.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Morovia Corporation.

Morovia may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Morovia, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Morovia is a trademark of Morovia Corporation. Other product and company names mentioned herein may be the trademarks of their respective owners.

Publication date: July 16, 2021

Revision: 11544

Technical Support

Phone: (905) 752-0226

Fax: (905) 752-0355

Email: support@morovia.com

Web: <http://www.morovia.com>

For more information about Morovia products, visit <http://www.morovia.com>.

Table of Contents

1. Overview	1
2. System Requirements	3
3. Specification	5
3.1. Package Contents	5
3.2. Symbologies Supported	5
4. Licensing	7
5. Fundamentals	9
5.1. Design Mode	9
5.2. Zooming	9
5.3. Working Area	10
5.3.1. Bounding Borders	10
5.3.2. Symbol Margins	11
5.3.3. Symbol Area	11
5.4. Barcode Glossary	12
6. Working with Low Resolution Devices	13
6.1. Problem	14
6.2. Magic Numbers	14
6.3. Solution	15
6.4. Transferring Images	15
7. Programming Interface	17
7.1. General	17
7.2. Creating a Barcode Object	17
7.3. Modifying Properties	17
7.4. Loading/Saving Barcode Object	18
7.5. Exporting images	18
7.6. Destroying the object	18
7.7. Error Handling	18
7.8. Concurrency Issues	18
7.9. Data Type Issues	18
7.9.1. Boolean Type	19
7.9.2. String Type	19
7.10. Using Barcode DLL in a .Net Program	19
8. Barcode Object Properties and Methods Reference	21
8.1. General	21
8.1.1. Properties	21
8.1.2. Methods	23
8.1.3. Deprecated Properties	23
8.2. AutoLabelSize Property	25
8.3. AutoSize Property	26
8.4. BackColor, ForeColor Properties	27
8.5. BarHeight Property	28
8.6. BearerBars Property	29
8.7. BorderColor Property	30
8.8. BorderStyle Property	31
8.9. BorderWidth Property	32
8.10. Code25OptionalCheckDigit Property	33
8.11. Code39OptionalCheckDigit Property	34

8.12. Code39StartStopChars Property	35
8.13. Comment Property	36
8.14. CommentAlignment Property	37
8.15. CommentFont Property	38
8.16. CommentMarginTop, CommentMarginBottom, CommentMarginLeft, CommentMarginRight Properties	39
8.17. CommentOnTop Property	40
8.18. DataMatrixModuleSize Property	41
8.19. DataMatrixTargetSizeID Property	42
8.20. Font Property	44
8.21. I2of5OptionalCheckDigit Property	45
8.22. LabelWidth, LabelHeight Properties	46
8.23. MaxicodeClass Property	47
8.24. MaxicodeMode Property	48
8.25. MaxicodeCountryCode Property	49
8.26. MaxicodeZipCode Property	50
8.27. Measurement Property	51
8.28. Message Property	52
8.29. NarrowBarWidth Property	53
8.30. NarrowToWideRatio Property	54
8.31. PDFAspectRatio Property	55
8.32. PDFMaxCols Property	56
8.33. PDFMaxRows Property	57
8.34. PDFModuleHeight Property	58
8.35. PDFModuleWidth Property	59
8.36. PDFSecurityLevel Property	60
8.37. PDFTruncatedSymbol Property	61
8.38. Picture Property	62
8.39. QuietZones Property	63
8.40. RasterImageResolution Property	64
8.41. Rotation Property	65
8.42. ShowCheckDigit Property	66
8.43. ShowComment Property	67
8.44. ShowHRTText Property	68
8.45. Symbology Property	69
8.46. SymbolMarginTop, SymbolMarginBottom, SymbolMarginLeft, SymbolMarginRight Properties	71
8.47. TexAlignment Property	72
8.48. TextOnTop Property	74
8.49. UccEanOptionalCheckDigit Property	75
8.50. ZoomRatio Property	76
8.51. About Method	77
8.52. ExportImage Method	78
8.53. Load Method	80
8.54. Save Method	81
9. Error Handling	83
9.1. Error Codes	83
10. Barcode Technologies	87
10.1. Introduction	87

10.2. Code 39	88
10.3. Code 39 Full ASCII	89
10.4. Code 39 HIBC	89
10.5. Codabar	90
10.6. Code 93	90
10.7. MSI/Plessey, Code 25 and Code11	90
10.8. UPC-A,UPC-E and UPC Supplements	91
10.9. EAN-13, EAN-8 and EAN Supplements	92
10.10. ISBN/Bookland	93
10.11. Code 128	94
10.11.1. How Barcode DLL Implements the Code128	94
10.11.2. Tilde Codes	94
10.12. UCC/EAN-128	96
10.12.1. Introduction	96
10.12.2. How Barcode DLL Implements UCC/EAN-128	96
10.12.3. Auto Check Digit	99
10.12.4. Input Format	99
10.12.5. Validation	100
10.12.6. Non-standard Application	100
10.13. DataBar Symbology Family	102
10.13.1. What is GTIN?	103
10.13.2. Barcode Height	103
10.13.3. Human Readable Text	103
10.13.4. DataBar Expanded and DataBar Expanded Stacked	104
10.14. Interleaved 2 of 5 (ITF25)	105
10.15. POSTNET	106
10.16. PDF 417	107
10.16.1. Security Level	107
10.16.2. Size Control	107
10.16.3. Input Format	108
10.16.4. Truncated PDF	108
10.16.5. Global Label Identification (GLI)	108
10.16.6. Macro PDF417	109
10.17. Data Matrix	111
10.17.1. Enhanced Feature Support	111
10.17.2. Size Control	111
10.17.3. Module Size	112
10.17.4. Input Format	113
10.17.5. Macro 5 and 6	113
10.17.6. Extended Channel Interpretation (ECI)	113
10.17.7. Structural Append (SA)	114
10.18. MaxiCode	115
10.18.1. Barcode DLL implementation	115
10.18.2. Message Structure	115
10.18.3. Input Format	116
10.18.4. Extended Channel Interpretation (ECI)	117
10.18.5. Structural Append (SA)	117
11. Technical Support	119
A. Component Software License Agreement	121

Glossary	125
Index	129

List of Figures

5.1. Zooming Effect	10
5.2. Anatomy Of a Label	10
5.3. Symbol Margins	11
10.1. Example PDF417 Barcode	107
10.2. Truncated PDF417 Barcode	108
10.3. Example Data Matrix Barcode	111
10.4. Example MaxiCode Barcode	115

List of Tables

3.1. Barcode Formats Supported by Barcode DLL 4.0	5
6.1. Typical Low Resolution Devices	13
6.2. Pixel Size Under Typical Resolutions	13
6.3. Magic Numbers	14
8.1. List of Barcode Object Properties	21
8.2. List of Barcode Object Methods	23
8.3. List of Deprecated Properties	23
8.4. Border Styles	31
8.5. CommentAlignment Options	37
8.6. DataMatrixTargetSizeID options	42
8.7. Measurement Unit Options	51
8.8. Rotation Options	65
8.9. Symbology Options	69
8.10. TexAlignment Options	72
8.11. PersistFormat Options (Load method)	80
8.12. PersistFormat Options (Save method)	81
9.1. Error Codes (Operational)	83
9.2. Error Codes (Encoding)	84
10.1. Symbologies Supported by Barcode DLL	87
10.2. Examples of UPC-A, UPC-E and Supplement	91
10.3. Examples of EAN-13, EAN-8 and Supplement:	92
10.4. List of Known AIs	96
10.5. GS1 DataBar Family	102
10.6. Optional Fields in Macro PDF417	109
10.7. Data Matrix Symbol Sizes	111
10.8. Macro 5 and 6	113
10.9. MaxiCode Modes	115
10.10. Tilde Codes (MaxiCode)	116

Chapter 1. Overview

Welcome to Morovia Barcode DLL! Morovia Barcode DLL empowers developers to quickly build comprehensive Windows-based barcode applications. Using Morovia Barcode DLL, you can add barcode printing functionality to an existing application in just a couple of hours.

You can use Morovia Barcode DLL to build applications for both large corporations and small businesses. Typical applications include:

- Retail Packaging
- Shipping
- Labeling Software
- Order Tracking
- Banking
- Postal Applications
- Inventory Control
- Asset Tracking
- Tool Tracking
- Document Tracking

Morovia Barcode DLL supports most linear barcode formats, including *Code 39*, *UPC-A*, *UPC-E*, *EAN-13*, *EAN-8*, *Code 93*, *Code128*, *EAN-128*, *Codabar*, *POSTNET*, *Royal Mail*, *HIBC*, *Interleaved 2 of 5* *GS1 DataBar*, *GS1 DataBar Truncated*, *GS1 DataBar Limited*, *GS1 DataBar Stacked*, *GS1 DataBar Stacked Omnidirectional* *GS1 DataBar Expanded* and *GS1 DataBar Expanded Stacked*. Morovia Barcode DLL also supports three 2D barcode formats, such as *PDF417*, *Data matrix* and *MaxiCode*.

Unlike other products that create low quality bitmaps, Morovia Barcode DLL draws itself using high-resolution metafile graphics that are device-independent¹ and adapt to printing devices supported by Windows. The control also exports to a variety of graphics formats including *BMP*, *JPEG*, *GIF*, *PNG*, *TIF*, *WMF* and *EMF*.

The Barcode DLL provides methods to save itself into a file in binary or XML text format. With *XML*, data can be easily exchanged among computers. For example, you can store all the properties in a database, transmit them over Internet, and load at a later time.

Best of all, this component does not have dependency on any other third party DLLs when running on Windows 2000 or above. Only one file needs to be included with your installation package - *MrvBarDLL.dll*. There are no run time dependencies or additional files to be included.

Barcode DLL can be used in a variety of ways. In the software package we include examples for programming environments such as Visual C++ and .Net. Because the DLL API is exported using standard Windows DLL interface, we expect that there is no problems when working in other environments, such as Perl, PowerBuilder and Pascal.

Version 4 includes both 32-bit and 64-bit binaries. The interfaces remain the same as the previous version 3.

¹Special handling is required for working with low-resolution devices to produce high-quality barcodes.

Chapter 2. System Requirements

Windows 7 is the minimum required Windows operating system to run Morovia Barcode DLL. It does not run on Windows 98, ME and Windows NT. Morovia will not make changes to the program to accommodate those legacy systems.

Morovia Barcode DLL does not utilize any third party DLLs. Nor does it use any registry settings.

Chapter 3. Specification

Morovia Barcode DLL has been tested on many environments, such as Visual Basic and Visual C++. Standard header files for C/C++ compilers are included. Other environments, such as C# and Powerbuilder, you need to import/declare the function prototype first. Refer to language manual on how to call a DLL in your programming environment.

3.1. Package Contents

The Barcode DLL package contains the following contents:

- Barcode DLL - MrvBarDLL.dll.
- Import library MrvBarDLL.lib for C/C++ compilers
- Reference manual, which you are reading on
- Release Notes
- VC15 (Visual Studio 2015) sample
- Visual Studio 2015 .Net sample (C#)

3.2. Symbolologies Supported

Morovia Barcode DLL 4.0 supports the following barcode formats:

Table 3.1. Barcode Formats Supported by Barcode DLL 4.0

Code 11	UPC-A	Code 128
Code 25	UPC-E	GS1 128 ^a
Codabar	EAN-13	Telepen
Code 39	EAN-8	Telepen Numeric
HIBC	Bookland	Data Matrix
Code 39 Extended	Postnet	PDF417
Code 93	Planet	MaxiCode
MSI/Plessey	Royal Mail	GS1 DataBar Limited
Interleaved 2 of 5	GS1 DataBar Truncated	GS1 DataBar Stacked
GS1 DataBar		GS1 DataBar Stacked
		Omnidirectional
		GS1 DataBar Expanded
		GS1 DataBar Expanded Stacked

^aPreviously known as UCC/EAN-128

Chapter 4. Licensing

Our license terms are included at the end of this manual. For the latest pricing, visit <https://www.morovia.com/products.html>.

Barcode DLL is developed to assist software developers to add barcode printing functionality into their custom software. As a result, only developer licenses are available. Each developer license allows one developer to develop software incorporating the DLL and distribute the DLL bundled with custom software, up to 10,000 copies.

Each developer must have his/her own license to work with Barcode DLL. The full version and the trial version share the same binary file. The barcode object is created without watermark when valid `License To` and `Registration Code` are passed to the `CreateBarcodeObject` function.

Chapter 5. Fundamentals

The core component within Morovia Barcode DLL is a powerful, versatile control, also referred as *Barcode Object* throughout this manual. This control does not only create barcode image, but also human readable text and comment. It allows flexible layout of these components. This chapter gets you familiar with the basic concepts necessary to work with the control.

5.1. Design Mode

To simplify application design, this manual categorizes the control usage into two design modes: **barcode design mode** and **label design mode**. Barcode design mode is for those whose tasks emphasize on the barcode image rather than the “label”. Under barcode design mode, the size of the working area is not fixed and can expand or shrink necessarily to hold the complete barcode label, including *quiet zones*, comments and margins. On the contrary, under label design mode the working area is predecided and will not expand or shrink. The barcode portion can expand or shrink but it will not impact the overall label space.

The property **AutoLabelSize** determines the design mode. If **AutoLabelSize** is TRUE, you are working under the barcode design mode, where the label size is not fixed. If more data is encoded into the barcode, or you add more text into the comment, the overall size increases. Under barcode design mode, the properties *LabelWidth* and *LabelHeight* are read only and you can not change them; rather, you can increase or decrease the overall drawing area by changing the margin, barcode data, or comment size.

On the other side, when **AutoLabelSize** is FALSE, the working area is pre-determined by the properties *LabelWidth* and *LabelHeight*. You can modify the natural size by modifying these two properties. Any drawings outside the predefined area are clipped.

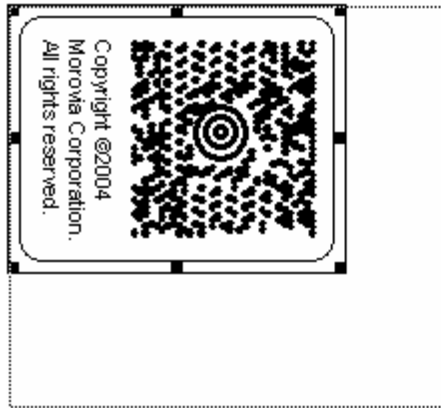
5.2. Zooming

In most cases you probably do not need the zooming feature. We highly recommend that you make sure that the **ZoomRatio** equals to 1.0 when you print the barcode. This zooming feature is provided for creating comprehensive labeling applications.

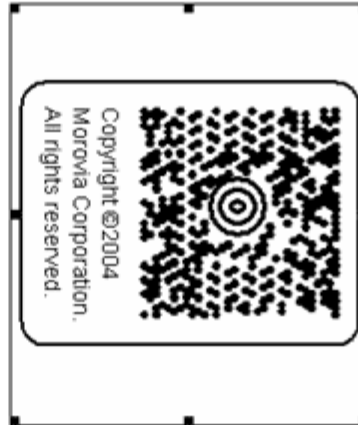
To avoid confusion, this manual defines two sizes here: **natural size** and **display size**. The natural size is the size of the control itself without scaling. The display size is the natural size multiplies the zoom ratio.

There are two types of zooming: *programmed* and *interactive*. In the first scenario, the control size is determined by the actual size and **ZoomRatio**. The program changes the control size by programmatically modifying the **ZoomRatio**. In the latter case, the user drags on the tracker box to the size desired, and the object responds by drawing itself to the maximum extent in the box specified. Although you can switch from one to the other zooming mode, you can not have both at the same time. The zoom mode is determined by property **AutoSize**: if this property is set to TRUE, the control determines the size and you can programmatically zoom the control by setting the **ZoomRatio** property to an appropriate value. If *AutoSize* is set to FALSE, the container takes the charge and the control redraws itself every time the container's size changes (with several extra lines of code you will be able to achieve the interactive dragging effect). Under the interactive mode you can not set *ZoomRatio* property; it becomes read-only.

Figure 5.1. Zooming Effect



When AutoSize is true, control draws itself based on the natural size and ZoomRatio



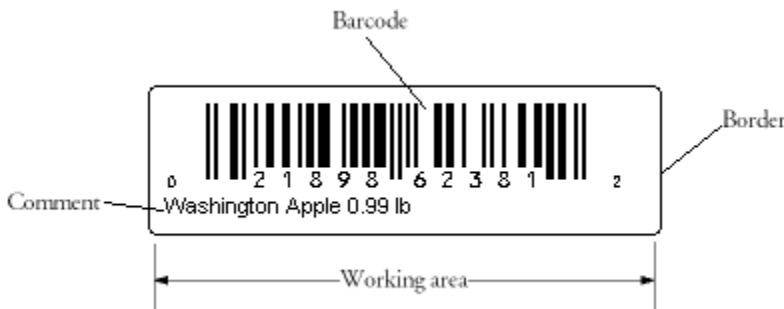
When AutoSize is set to false, control draws itself to the full extend that container prescribes

5.3. Working Area

There are a couple of elements appearing in the working area - barcode, human readable, margins and comment. Since the working area includes not only the barcode image, in this manual we use the term “label” or “the working area” to refer to the whole drawing.

The whole drawing (label) consists of three major components, as you can see from the illustration below: (1) the bounding borders surrounding the label; (2) symbol margins surrounding the symbol area; (3) the symbol area, which includes the barcode, human readable and the comment. Furthermore, the whole symbol area can be divided into two parts: (1) the barcode. A barcode can also have a human readable text. You can add extra white spaces surrounding the barcode, a.k.a. **Quiet Zones**. The human readable text can appear on the top of the barcode, or on the bottom, or does not appear at all. (2) the comment. A comment consists of one or more text paragraphs. Same as the human readable text, the comment can be placed on the top, on the bottom of the image, or does not appear at all.

Figure 5.2. Anatomy Of a Label



5.3.1. Bounding Borders

Three properties control the appearance of the borders: **BorderColor**, **BorderStyle** and **BorderWidth**. To turn off the border, set **BorderWidth** to 0 or **BorderStyle** to `mbxBorderStyleNone`. The **BorderWidth** property does not affect the overall size. The border aligns its outer edge to the boundary of the working area.

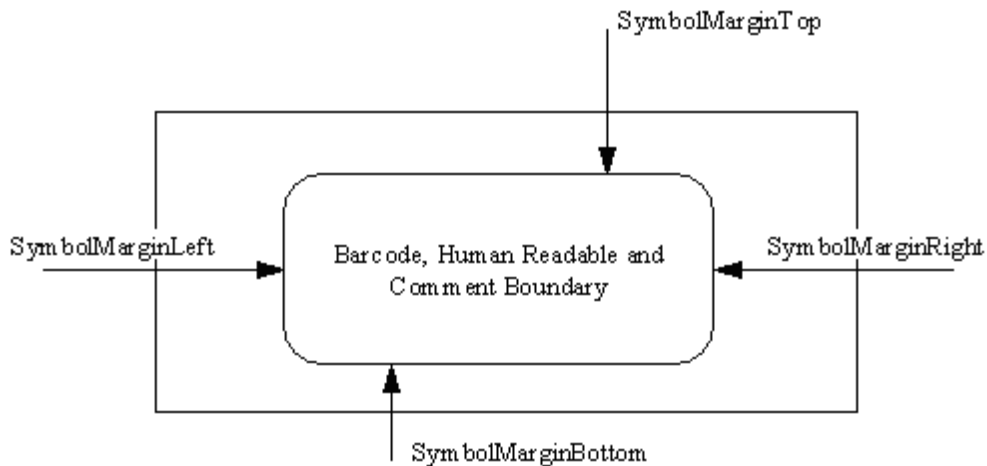
The borders are turned off by default in Barcode DLL 4.0.

5.3.2. Symbol Margins

The margin properties control the marginal space around the symbol area (barcode and comment). There are four symbol margin properties: **SymbolMarginLeft**, **SymbolMarginRight**, **SymbolMarginTop** and **SymbolMarginBottom**, which define the margin spaces in the four directions respectively.

By default, Barcode DLL adds 100 mils (2.54 mm) margins surrounding the symbol. To modify the symbol margins, set the four symbol margin properties to appropriate values.

Figure 5.3. Symbol Margins



5.3.3. Symbol Area

The symbol area consists of two components: *barcode* and *comment*. The barcode element comprises a barcode image, an optional human readable text and optional quiet zones. The human readable can be placed on the top or the bottom of the barcode element, or does not display at all.

The comment element contains one or more text paragraphs.

5.3.3.1. Barcode Element

The barcode part is always placed in the center of the working area (if there is no rotation at all).

The vertical position is determined by two comment margin properties: **CommentMarginTop** and **CommentMarginBottom**. If the comment is placed on top of the barcode area, the distance from the bottom edge of the comment to the top edge of the barcode is **CommentMarginBottom**; otherwise the top edge of the barcode is the top boundary of the symbol. Similar calculation applies to the case when comment is placed on the bottom.

5.3.3.2. Comment Element

The vertical position of the comment depends on the **CommentOnTop** property. When this property is set to **TRUE**, the distance from the top edge of the comment to the bottom of the symbol is expressed in **CommentMarginTop**. The distance from the bottom edge of the comment box to the top of the barcode is expressed in **CommentMarginBottom**.

The design mode affects the horizontal placement of the comment element. Under label design mode, where the label size (working area) is fixed regardless of the symbology and encoding data, the property **CommentMarginLeft** measures the distance from the left edge of the symbol to the left edge of the comment

box. The **CommentMarginRight** property measures the distance from the right edge of the symbol to the right edge of the comment box. The calculation is somehow different under barcode design mode: Under barcode design mode, the **CommentMarginLeft** measures the distance from the left edge of the comment text box to the left edge of the barcode, and **CommentMarginRight** measures the distance from the right edge of the barcode to the right edge of the comment box.

5.4. Barcode Glossary

A *barcode* consists of two elements. In linear symbologies, the dark element is called *Bar*, and the white element is called *Space*. In two dimensional symbologies, both elements are referred as *module* - the dark (black) module and the light (white) module.

Most linear symbologies allow 1 or 2 widths. In the first scenario, the width of an element could be 2, 3 or 4 times of the width of the narrowest element. In the second scenario, two widths are defined, one for the wide elements and one for the narrow elements. The width of the wide element divided by the width of the narrow element is called **NarrowToWideRatio**. In both cases, the overall barcode length achievable depends on how small the width of the narrowest element can go. The width of the narrowest element is also referred as X dimension (sometimes abbreviated as “X Dim”). In Barcode DLL you can set the *X dimension* through property **NarrowBarWidth**.

The length of the X dimension is usually very small. The industry measures the X dimension in 1/1000 inch units, called **mils**. A mil equals to 1/1000 inch. Nowadays millimeters are also used in some cases. Due to the legacy that most barcodes have so far been produced by specialized barcode printers and most of those barcode printers have a low resolution at 203-dpi, the X dimension is often integral times of the pixel width on a 203-dpi printer - a typical requirement is 15 mils (the width of three printer pixels on a 203 dpi printer).

The smallest width that a barcode reader can distinguish is called **scanner resolution**. Today, most commercial scanners have a resolution around 10 mils, meaning that they won't be able to read the barcodes with X dimension at 5 mils. High resolution scanners can go as low as 3~5 mils.

Barcode DLL supports two measurement units: mils and high metric (1/1000 cm). The user can select whichever convenient to use. On the other side, although you can set whatever values you desire, you may not be able to achieve the accuracy because of the limitations of the device. You can not print on 1.5 pixels. If you plan to create barcodes on low-resolution devices, such as fax transmission, screen or thermal printers, refer to Chapter 6, *Working with Low Resolution Devices*.

Chapter 6. Working with Low Resolution Devices

The top priority when making barcode images is to make sure that the barcode is readable. Although it seems quite obvious, it is nevertheless not a trivial task, especially when working with a low-resolution device. A low-resolution device is a printer or any target device that has a resolution lower than 300 dpi. A typical example is G3 fax transmission, which has a resolution only at 200 dpi. The computer display has a resolution of 96 dpi. Many thermal transfer printers also fall into this category, with resolution as low as 203 dpi.

Table 6.1. Typical Low Resolution Devices

Device	Resolution
Computer display	96 dpi
G3 Fax machine	200 dpi
Thermal Transfer Printer	203 dpi
Dot matrix printer	150 dpi, 240 dpi, 300 dpi

Why does device resolution become an issue? Modern computer graphics technology is based on a fundamental assumption that the program can address any logical units, being 1/100 inch per unit, or 1/1000 inch per unit. In this way, computer graphics simplifies the drawing process and makes the drawing data more portable. In practice, most devices are raster and the smallest unit they can address is a pixel. You can not print on 1.5 pixels. Even worse, the size of pixel does not match the English or metric unit system used in our daily lives. Let's take a look at the pixel size measured in mils and mm:

Table 6.2. Pixel Size Under Typical Resolutions

dpi (dots per inch)	size in mils	size in mm
96	10.42	0.265
150	6.67	0.169
203	4.93	0.125
240	4.17	0.106
300	3.33	0.085
600	1.67	0.042

As you can see, the size of pixel is variable from device to device. For a 600-dpi laser printer, there are 600 pixels in an inch. The same number of pixels occupy 6.25 inches on a computer screen, much longer than the one on the printer. To eliminate the tedious work converting the units back and force, computer graphic layer allows drawing commands to be specified with lengths in a logical unit. You could use pixel based drawing, but naturally most of software choose a mapping unit in mils (1/1000th inch) or 0.01 millimeters. This makes programming a lot easier, because the software does not need to address on the pixel level and the drawings are portable from device to device. Unfortunately, this approach works under a major assumption - the actual resolution does not impact the quality of the drawing. This assumption holds true for text renderings

and pictures, but not for barcode printing which requires great assurance on the constant width of each element. Converting from logical units to pixels usually results in rounding errors. When rounding errors are accumulated all the way, some elements will no longer keep the required constant widths, resulting in a low quality barcode.

6.1. Problem

To make high quality barcodes, several work-arounds have been adopted by barcoding software:

- Requiring the use of high-resolution printers
- Requiring use of Magic Numbers as NarrowBarWidth
- Forcing the use of pixels as length units

The pixel size of a high resolution device is too small to be noticeable by even high resolution scanners. The overall accumulation of rounding errors is small. The barcode printed is also very accurate - meaning that you can always achieve the size you want on a high resolution printer.

Another way is to make the width value closest to integral times of a pixel length. Our testing found that this approach worked very well when the bar code element width occupying at least 2 pixels. Based on this theory, when you carefully select the element width value you can still achieve a grade A barcode on a low-resolution printer.

Some software only work with specific barcode printers. They work around the issue by allowing user to enter the length values in the number of pixels. The problem with this approach is that the barcode data is only meaningful to certain category of printers. The barcode data is not portable.

6.2. Magic Numbers

By making the logical units closest to the integral times of the width of a pixel, we calculate the magic numbers as shown in the table below:

Table 6.3. Magic Numbers

dpi (dots per inch)	pixel size in mils	magic numbers (in mils)
96	10.42	21, 31
150	6.67	14, 20
203	4.93	10, 15
240	4.17	8, 13, 17
300	3.33	7, 10, 14
600	1.67	7, 9, 10, 12

From the table above, the smallest width achievable on a 203-dpi thermal printer is 10 mils. On a low-resolution printer you can not achieve high precision - the achievable smallest width on a 203 dpi thermal printer is 5 mils and on the computer screen it is around 10 mils.

When you create barcodes under the screen resolution (96 dpi), you end up with a big X dimension value - 21 mils. As previously analyzed, 1X does not work very well because the accumulated rounding errors may result in a complete loss of an element.

6.3. Solution

To solve the barcode quality issue while retaining maximum portability, our Barcode DLL aligns all lengths to the edge of pixels. Moreover, the drawings are now in the unit of pixels, instead of any logical units. Doing this ensures you get a quality barcode even on low resolution devices when you are drawing a width at 1-pixel's width level.

However, this approach brings some side effects. Due to the priority to ensure the constant width, the overall length achieved may vary from device to device. You may find that the same barcode prints much bigger or smaller on a thermal printer than on a laser printer. You have to give up the precision to gain the quality. Secondly, the pixel level drawing only happens when `ZoomRatio=1`.

As you can see from the table, lengths of the barcodes are almost the same on high resolution printers (300 dpi and 600 dpi). But on low-resolution printers, it varies greatly, especially on the computer display.

During the rasterization process, the printer driver is consulted to get the best rendition. Text character usually appear bigger than their renditions on laser printers, and some printer models have problem rendering bold fonts. We recommend you do more testing when working on these low-resolution devices, and stick to one configuration which meets the quality and performance.

On low resolution devices, the choices for the actual `NarrowBarWidth` are limited, especially when we want the barcode as small as possible. The program calculates the `NarrowBarWidth` based on the resolution of the target device. Although you can specify different values, you may not get different results because the program only draws bars/spaces to the pixel edge. For example, when Barcode DLL draws on a computer display, specifying `NarrowBarWidth` as 8 mils renders the same barcode as with `NarrowBarWidth` set to 14 mils. Both lengths turn into 1 pixel at the time of rendering.

With the overall barcode length varying from device to device, the overall layout may look different too.

6.4. Transferring Images

Graphics exchange file formats are invented to allow the drawings to be transferred from one program to another. Fundamentally there are two categories: vector graphics format and raster graphics format. A vector graphics file stores drawing commands. On the target device, the drawing commands are replayed.

Many people think that the image files produced under high resolutions will have the best quality. This is not correct. When an image produced based on high resolution is rendered on a low resolution printer, the render program has to scale down all lengths. If it is a raster image, several adjacent pixels are compressed into one pixel. If it is a vector image, the lengths are divided by a ratio and rounded to the nearest integer. This transformation process likely causes rounding errors. When the bar width is small, the widths of elements become non-constant, and some drawings may be lost during the compression process.

Transferring images produced based on low resolution to a high resolution device may also have problems, but to a lesser degree. First, if two resolutions are compatible (for example 300 -> 600) there is no distortion or loss at all. And there are no rounding errors converting the logical units to the device units in the second device. Secondly, even if the two resolutions are not compatible (96 dpi -> 600 dpi), the pixel size in the second device is too small that one more or less pixel won't affect the barcode quality at all. And most of important, narrow elements won't be lost when scaled up.

As a result, you should pay special attention when you have to transfer images from one to another.

Obviously, the best approach is to produce images under a resolution that matches the target printer. Barcode DLL automatically rounds all length values to the pixel level.

Needlessly to say, you may not have the control over the process under some circumstances. We list several common cases below:

1. Exporting raster images

Raster images (JPEG, GIF, TIF, PNG and BMP) can be exported through *ExportImage* method. During the export process, drawing commands are converted into an array of color information. The resolution used during the transform is specified in *RasterImageResolution*. You should set *RasterImageResolution* the same value as the resolution of your target printer.

2. Exporting EMF images

EMF is the recommended format for exporting. It produces a small size file. During the transform process, the default printer driver is consulted for the pixel size. As a result, you should set the target printer default when exporting EMF files. There are two methods to export EMF images. The first one is to call *ExportImage* method of the control. The second one is to get *Picture* property and retrieve the handle.

3. Exporting WMF images

WMF is a vector graphics format; however it does not contain frame size information. Barcode DLL uses 1440dpi as the reference resolution when rendering WMF images. Normally you won't encounter readability issues when working on high resolution devices such as laser printers.

4. Creating barcodes at web server side

It is tough to create barcode images at web server and send to the client browser. The problem is that most web browsers only support raster image formats at screen resolution. You can use either screen resolution (96 dpi), or the resolution of target printer, however, in the latter case you will need to calculate the screen size by yourself.

5. Creating barcodes in Microsoft Office

Microsoft Office programs either retrieve either WMF handles, or EMF handles at 1440 dpi. As a result you will need a high-resolution printer to render the image without loss.

It should be noted that some image processing programs have no concept of "resolution". Although they support vector graphics format, however they simply raster the image into pixels at screen resolution. As a result, if the original images were produced based on a higher resolution, the resulted barcodes usually have very low quality. Do not use those programs to edit barcode images exported from Barcode DLL.

Chapter 7. Programming Interface

7.1. General

Although DLL only allows exporting plain “C” functions, object-oriented methodology is employed during the interface design. All functions operate on a special object called the `barcode object`. The object is identified by a 32-bit integer type called `HANDLE`. A `HANDLE` is defined as a `void*` pointer in C language, and a `Long` in classic Visual Basic.

A typical application creates the barcode object through function `CreateBarcodeObject` and retrieves the handle to the object. Subsequently it calls various functions to manipulate the object properties. The function `mbxExportImage` is used to retrieve the barcode image renderings. After the application finishes using the object, it calls the function `DestroyBarcodeObject` to release all resources allocated to the object.

7.2. Creating a Barcode Object

A barcode object is created through function `CreateBarcodeObject`, which has the following prototype:

```
HANDLE CreateBarcodeObject(const char* lpszLicenseTo,  
                           const char* lpszRegCode);
```

The `CreateBarcodeObject` function creates a barcode object using default properties and returns a number that uniquely identifies the object (handle). Through the handle, the client application modifies the object properties, persists the object to a disk file, and retrieves barcode rendering in a variety of image formats.

Two string parameters are required for the function. At the time when you place the order, we assign the *LicenseTo* and *Registration code* to you when the order is completed. You should keep the registration code in a safe place and use them only in the source code. Putting the registration code visible to your user, such as a plain text file, is not allowed.

The function will always succeed unless the program does not have sufficient memory. Therefore the return value does not tell you if the license parameters are correct. Instead the object enters the DEMO mode. Under the demo mode, the barcode image exported has a “DEMO” watermark.

The application can query if the barcode object is under the demo mode by calling function `IsBarcodeObjectDemo`. If the function returns a non-zero value, the barcode object is under demo mode and every barcode image exported carries a “DEMO” watermark.

7.3. Modifying Properties

There are several dozen properties you can modify on the barcode object. Most properties are not related to each other; so changing one won't affect others. However, several exceptions exist. Changing the *Symbology* property will also cause the *Message* property to change due to the fact that different symbologies have different default Message values. Changing the measurement unit (*Measurement* property) will cause values of length properties to change so that the barcode sizes remain intact.

To access a specific property your application calls `get_XXX` and `put_XXX` functions. Here XXX is the property name. For example, to retrieve the value of the Message property, use `get_Message` function. To set the Message property, use `put_Message` function. For a list of all properties, refer to Chapter 8, *Barcode Object Properties and Methods Reference*.

7.4. Loading/Saving Barcode Object

Properties of a barcode object can be saved into a file and loaded at later time. The functions **mbxLoad** and **mbxSave** are used for loading and saving the barcode object, respectively. The first parameter is the file path, and the second parameter indicates the file type.

7.5. Exporting images

The client application calls **ExportImage** or **ExportImage2** functions to export the barcode image to a disk file or a Stream object.

7.6. Destroying the object

After you have done with the object, you should call **DestoryBarcodeObject** to destroy the barcode object. By doing so you release all resources associated with the object.

7.7. Error Handling

Not all functions succeed unconditionally. You should always check the return value for errors when working with the program.

Barcode DLL reports two kinds of errors - operational errors and encoding errors. They are handled in different ways.

Operational errors occur when an operation fails or a property is set to an invalid value. The error is reported as returned value of the function. The error value is expressed in a 32-bit integer form, very similiar to the HRESULT error code in the COM world. Consequently you can use SUCCEEDED and FAILED macros to tell if the operation succeeds or fails.

For a list of applicable error codes, refer to the Chapter 9, *Error Handling*.

When error happens, you can use **GetLastBarcodeErrorMessage** to retrieve the reason, as demonstrated by the VB code below:

```
Dim rc As Long
rc=put_Message(handle, "ABCD123")
If ( rc < 0 ) Then
    GetLastBarcodeErrorMessage(barcode_object_handle, strError)
    MsgBox strError
End IF
```

7.8. Concurrency Issues

The DLL is thread safe - all functions can be called simutaneously from multiple threads. However, the same barcode object handle value should only called by a single thread at any time. If you need to do so, serialize the access using the protection mechanism provided by the operating system.

7.9. Data Type Issues

When working with our DLL interface, pay attention to the `boolean` type and the `string` type used by exported functions. See below for details.

7.9.1. Boolean Type

The version 4 API uses `bool` type which is supported by most C compilers today. The previous version 3 uses `VARIANT_BOOL`, which may require conversion in some cases.

7.9.2. String Type

When you use DLL functions to retrieve a string type property, you need to supply a buffer large enough to hold the data. You can use 1024 as a reasonable buffer size. For example, the following code retrieve the Message property of the barcode object:

```
char message[1024];
get_Message(handle, message);
```

For performance reasons, Barcode DLL does not check if the buffer is long enough. Consequently the buffer overrun may happen if the buffer size is too small.

In Visual Basic 6, the default String type does not have a buffer at all. Passing an empty string will cause the program to crash. You should never use a zero-length string to retrieve a string type property. Instead, use the code below:

```
Dim strDefaultValue As String * 1024
Dim rc As Long
rc = get_Message(barcode_object_handle,
    strDefaultValue)
```

In .Net program, you need to use `StringBuilder` class to retrieve the string results. Refer to our example for details.

7.10. Using Barcode DLL in a .Net Program

.Net programming has gained popularity recently. To use DLL in a .net program, you need to declare functions using `DllImport` first. We have included a .Net sample called `CaseCodePrint` with the product. We wrapped all DLL export functions within a C# class. To make our code easy to understand, we wrap these functions into properties and methods. You may take a look at our sample for details.

Chapter 8. Barcode Object Properties and Methods Reference

8.1. General

Although DLL only allows plain “C” functions exported, the design uses an object-oriented methodology. All functions manipulate the barcode object through a HANDLE which uniquely identifies the barcode object. A HANDLE is defined as a pointer in C language.

A typical application creates the barcode object through function *CreateBarcodeObject* and retrieves the handle to the object. Subsequently it calls various of functions to manipulate the object properties. The function *mbxExportImage* is used to retrieve the barcode rendering from the object. After the application finishes, the function *DestroyBarcodeObject* should be called to release all resources allocated by the object.

All properties and methods are listed in this chapter. Some properties are now obsolete and kept only for backward-compatibility reason.

Some properties may not be modifiable under certain design mode. Some properties may be related to other properties - i.e. change to one property will change other properties. For example, Changing *Symbology* property will also alter the value of *Message* property.

8.1.1. Properties

Table 8.1. List of Barcode Object Properties

AutoLabelSize	Determines whether the current work mode is under barcode design mode or the label design mode.
AutoSize	Determines which party controls the sizing of the control.
BackColor	Specifies the background color for the control.
BarHeight	Specifies the height of the bars in the control, in user units.
BearerBars	Determines whether to include the bearer bars around the barcodes. applicable on selected symbologies.
BorderColor	Specifies the border color for the control.
BorderStyle	Specifies the border style.
BorderWidth	Specifies the border with, in logical units
Code25OptionalCheckDigit	Determines whether to include an optional checksum digit in all Code25 barcodes.
Code39OptionalCheckDigit	Determines whether to include an optional checksum digit in all code 39 barcodes.
Code39StartStopChars	Determines whether to display the start and stop characters in the human readable of all code 39 barcodes.
Comment	Specifies the string for the human readable comment printed around the barcode.
CommentAlignment	Determines how the text paragraphs are aligned within the comment.

CommentFont	Specifies the font used to draw comment text.
CommentMarginBottom	Specifies the margin on the bottom of the comment box.
CommentMarginLeft	Specifies the margin on the left of the comment box.
CommentMarginRight	Specifies the margin on the right of the comment box.
CommentMarginTop	Specifies the margin on the top of the comment box.
CommentOnTop	Determines whether the comment box is placed above the barcode image.
DataMatrixTargetSizeID	Specifies the size of the Data Matrix barcode created.
Font	Specifies the font used to draw human readable text.
ForeColor	Specifies the foreground color of the control.
I2of5OptionalCheckDigit	Deprecated in version 3.4.
LabelHeight	Specifies the height of the label (working area).
LabelWidth	Specifies the width of the label (working area).
MaxicodeClass	Specifies the service class for the structured carrier message in MaxiCode symbols.
MaxicodeCountryCode	Specifies the country code for the structured carrier message in MaxiCode symbols.
MaxicodeMode	Specifies the encoding mode for the Maxicode symbols.
MaxicodeZipCode	Specifies the zip/postal code for the structured carrier message in MaxiCode symbols.
Measurement	Specifies the measurement unit for all length properties.
Message	Specifies a string which represents the data to be encoded.
NarrowBarWidth	Specifies the width of the narrowest module in linear symbologies.
NarrowToWideRatio	Specifies the ratio used to calculate the width of the wide element. Applicable on selected symbologies.
PDFAspectRatio	Specifies the overall height to width ratio of the PDF417 barcodes created.
PDFMaxCols	Specifies the maximum number of codeword columns to allow in the PDF417 barcodes created.
PDFMaxRows	Specifies the maximum number of codeword columns to allow in the PDF417 barcodes created.
PDFModuleHeight	Specifies the height of the module in the PDF417 barcodes created.
PDFModuleWidth	Specifies the width of the module in the PDF417 barcodes created.
PDFSecurityLevel	Specifies the security level for error correction to use in PDF417 barcodes.
PDFTruncatedSymbol	Determines whether to create the truncated version of Pdf417 barcodes.
Picture	Returns a snapshot of the drawing in Windows Enhanced Metafile Format (EMF).
QuietZones	Determines whether to include quiet zones in the barcodes.
RasterImageResolution	Specifies the resolution which is used to export raster images.
Rotation	Specifies the orientation of the working area in degrees.

ShowCheckDigit	Determines whether to include the checksum character in the human readable.
ShowComment	Determines whether the control displays the comment element.
ShowHRTText	Determines whether to display the human readable text.
SymbolMarginBottom	Specifies the margins on the bottom of the symbol.
SymbolMarginLeft	Specifies the margins on the left of the symbol.
SymbolMarginRight	Specifies the margins on the right of the symbol.
SymbolMarginTop	Specifies the margins on the top of the symbol.
Symbology	Specifies the barcode format (symbology)
TexAlignment	Specifies how the human readable text is aligned.
TextOnTop	Determines whether the human readable text is placed above the barcode image.
UccEanOptionalCheckDigit	Deprecated in version 3.4
ZoomRatio	Specifies the scale ratio of the current display area vs. the natural size.

8.1.2. Methods

Table 8.2. List of Barcode Object Methods

About	Displays the About dialog.
ExportImage	Exports the drawing to a graphics file with the specified format.
Load	Loads the control properties from a file in binary or XML format.
Save	Saves the control properties to a file in binary or XML format.

8.1.3. Deprecated Properties

During the evolution of this product, some properties have been deprecated in major releases. Deprecated properties are no longer used in the implement. They are kept in the interface to retain backward compatibility. Existing applications require no change to use newer versions of the product. Setting values to deprecated properties render no effects.

Generally speaking, a property is deprecated because it is redundant, difficult to get it right at the first place, and confusing to our customers.

Table 8.3. List of Deprecated Properties

Name	Deprecated Since	Comment
BarWidthReduction	3.2	Pixel-based rendering method makes it obsolete.
I2of5OptionalCheckDigit	3.4	Interleave 2 of 5 requires input to be even. In version 3.4, a check digit is automatically calculated and appended if the input has an odd length. See Section 10.14, “Interleaved 2 of 5 (ITF25)” for more information.
UccEanOptionalCheckDigit	3.4	Since version 3.4, all GS1-128 applications that are known to the program that have mod 10 check digit will have the check

Name	Deprecated Since	Comment
		digit calculated automatically and appended if necessary. See Section 10.12.3, "Auto Check Digit" for more information.

8.2. AutoLabelSize Property

Description

Returns or sets the value that determines the current work mode.

Syntax

```
HRESULT get_AutoLabelSize(HANDLE handle, bool* pVal);  
HRESULT put_AutoLabelSize(HANDLE handle, bool val);
```

Remarks

Use this property to set/return the current work mode. When *AutoLabelSize* property is TRUE, the current work mode is barcode design mode; otherwise the label design mode is assumed. Under the barcode design mode, the size of the working area is not fixed. It grows or shrinks as the sizes of other components, such as margins, barcode and the comment change. Under barcode design mode, properties *LabelWidth* and *LabelHeight* are read-only and can not be altered.

On the contrary, under the label design mode, the size of the working area is fixed and can not be modified. Under the label design mode, anything beyond the working area is clipped.

If you intend to create a barcode as small as possible, choose the barcode design mode by setting this property to TRUE. If you'd like to print a full label with all components turned on, select the label design mode.

Note When you switch *AutoLabelSize* from TRUE to FALSE, the predefined label size - 2 by 2 inches is assumed.

See Also

Section 8.22, "LabelWidth, LabelHeight Properties"

Section 8.3, "AutoSize Property"

8.3. AutoSize Property

Description

Returns or sets the value that defines how the control size is determined.

Syntax

```
HRESULT get_AutoSize(HANDLE handle, bool* pVal);  
HRESULT put_AutoSize(HANDLE handle, bool val);
```

Remarks

AutoSize controls how the object responds to the sizing request from the container. When *AutoSize* is FALSE, the object redraws itself to the maximum extent that the container specifies; otherwise it calculates the display size by multiplying its natural size with the **ZoomRatio**, and draws itself within the display size.

When *AutoSize* is changed from FALSE to TRUE, the *ZoomRatio* is changed back to 1.0.

See Also

Section 8.50, "ZoomRatio Property"

Section 8.2, "AutoLabelSize Property"

8.4. BackColor, ForeColor Properties

Description

BackColor - returns or sets the background color of the control.

ForeColor - returns or sets the foreground color of the control.

Syntax

```
HRESULT get_BackColor(HANDLE handle, LONG color);  
HRESULT put_BackColor(HANDLE handle, LONG* pColor);  
HRESULT get_ForeColor(HANDLE handle, LONG color);  
HRESULT put_ForeColor(HANDLE handle, LONG* pColor);
```

Remarks

For opening systems we strongly recommend to set the background color to solid white (0xFFFFFFFF) and foreground color to black (0x000000). Note: barcode requires decent contrast between the foreground color and the background color in order to be readable. Always test the readability thoroughly when you select a color pair different from black and white.

8.5. BarHeight Property

Description

BarHeight - returns or sets a value for the height of bars in Barcode control.

Syntax

```
HRESULT get_BarHeight(HANDLE handle, LONG* pVal)  
HRESULT put_BarHeight(HANDLE handle, LONG val)
```

Remarks

The **BarHeight** property specifies the height of the dark elements in all linear symbologies with exception (see notes below). The actual value is affected by the *Measurement* property. If *Measurement* is set to *mbxMeasureEnglish*, the unit for this property is mils (1/1000 inch) otherwise it is 1/1000 cm. The default value is 1000 which translates to 1 inch or 1 cm, depending on the measurement unit specified.

This property have no effect on the size of two-dimensional barcodes, such as *PDF417*, *DataMatrix* and *MaxiCode*.

The height of elements in postal symbologies (*POSTNET* and *RoyalMail*) is fixed. Therefore, changing this property has no effect on those types of barcodes.

The height of bars in a *DataBar Truncated* symbol is fixed at 13X (X is the industry term for *NarrowBarWidth*), and the height of a *DataBar Stacked* symbol is fixed at 50X. Therefore, this property does not affect the height of those two types of symbols.

In stacked symbologies (*DataBar Stacked Omnidirectional* and *DataBar Expanded* (multi-row), the overall height is the number of rows multiplying *BarHeight*, plus the height of any required separator rows.

8.6. BearerBars Property

Description

Returns or sets a value that determines whether to include bearer bars around the barcode.

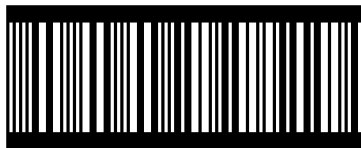
Syntax

```
HRESULT get_BearerBars(HANDLE handle, bool* pVal);  
HRESULT put_BearerBars(HANDLE handle, bool val);
```

Remarks

Bearer bars (see below) are horizontal bars printed across the top and bottom of the barcode image. Bearer bars can help avoid partial reads should the reader move off the top or bottom of the code. Only certain symbologies require bearer bars (for example Interleaved 2 of 5) as the start and stop characters in most bar codes make bearer bars unnecessary.

In Barcode DLL the following symbologies can have bearer bars: *Codabar*, *Code11*, *Code25*, *Code128*, *UCC/EAN-128*, *Code39*, *Code39 HIBC*, *Code 39 Full ASCII*, *Code93*, *Interleaved 2 of 5*, *MSI/Plessey* and *Telepen*. Other symbologies ignore this property.



0 07 07007 07723 6

8.7. BorderColor Property

Description

Returns or sets the color of borders.

Syntax

```
HRESULT put_BorderColor(HANDLE handle, LONG val);  
HRESULT get_BorderColor(HANDLE handle, LONG* pVal);
```

Remarks

Use this property to specify the border color of Barcode DLL. The default value is 0 (black).

See Also

Section 8.9, "BorderWidth Property"

Section 8.8, "BorderStyle Property"

8.8. BorderStyle Property

Description

BorderStyle - returns or sets a value that determines the border style.

Syntax

```
HRESULT put_BorderStyle(HANDLE handle, LONG Val);  
HRESULT get_BorderStyle(HANDLE handle, LONG* pVal);
```

Remarks

Use **BorderStyle** property to specify the border style. This property is set by using one of the BorderStyle enumeration values:

Table 8.4. Border Styles

Constant	Value	Description
mbxBorderStyleNone	0	No border lines
mbxBorderStyleSolid	1	Solid line
mbxBorderStyleDash	2	Dash line
mbxBorderStyleDot	3	Dot line
mbxBorderStyleDashDot	4	Dash dot line
mbxBorderStyleDashDotDot	5	Dash dot dot line

See Also

Section 8.7, "BorderColor Property"

Section 8.9, "BorderWidth Property"

8.9. BorderWidth Property

Description

BorderWidth - returns or sets the value for the border width.

Syntax

```
HRESULT get_BorderWidth(HANDLE handle, LONG* pVal);  
HRESULT put_BorderWidth(HANDLE handle, LONG val);
```

Remarks

Use **BorderWidth** property to specify a border width for the Barcode DLL. The border width is expressed in logical units prescribed by the **Measurement** property. For example, if you set this property to 15 and the *Measurement* is `mbxMeasureEnglish`, the border width is 15 mils (0.015 inch). The default value is 15.

To disable the border, set BorderWidth to 0 or BorderStyle to `mbxBorderStyleNone`.

See Also

Section 8.8, "BorderStyle Property"

Section 8.7, "BorderColor Property"

8.10. Code25OptionalCheckDigit Property

Description

Code25OptionalCheckDigit - returns or sets a value that determines whether to include an optional checksum digit in all Code25 barcodes produced by Barcode DLL.

Syntax

```
HRESULT put_Code25OptionalCheckDigit(HANDLE handle, bool val);  
HRESULT get_Code25OptionalCheckDigit(HANDLE handle, bool* pVal);
```

Remarks

A code 2 of 5 barcode can have an optional check digit. When the **Code25OptionalCheckDigit** is TRUE, a check digit is calculated using modulo 10 algorithm and added to a code 2 of 5 barcode. To display the checksum digit in the human readable text, set ShowCheckDigit to TRUE.

See Also

Section 10.7, “MSI/Plessey, Code 25 and Code11”

8.11. Code39OptionalCheckDigit Property

Description

Code39OptionalCheckDigit - returns or sets a value that determines whether to include an optional checksum digit in all Code39 barcodes produced by Barcode DLL.

Syntax

```
HRESULT put_Code39OptionalCheckDigit(HANDLE handle, bool val);  
HRESULT get_Code39OptionalCheckDigit(HANDLE handle, bool* pVal);
```

Remarks

A code 3 of 9 symbol can have an optional check digit (character) at the end of the barcode. When the Code39OptionalCheckDigit property is set to TRUE, a check digit is calculated using modulo 43 method and appended to the end of the barcode. To display the checksum digit, set **ShowCheckDigit** to TRUE.

This property affects two symbologies: Code 39 and Code39 Full ASCII.

See Also

Section 10.2, "Code 39"

Section 10.3, "Code 39 Full ASCII"

8.12. Code39StartStopChars Property

Description

Returns or sets a value that determines whether to display the start and stop characters in the human readable text in all Code39 barcodes produced by Barcode DLL.

Syntax

```
HRESULT get_Code39StartStopChars(bool* pVal);  
HRESULT put_Code39StartStopChars(bool val);
```

Remarks

For historical reasons many code39 symbols print start/stop characters (asterisks) at the beginning and the end of the human readable text. While the start and stop characters are always present in the barcode, it is not necessary for them to appear in the human readable. When the Code39StartStopChars property is set to TRUE, the asterisks are displayed at both the beginning and end of the human readable.

This property affects the three Code 39 symbologies - *Code 39*, *HIBC* and *Code39 Full ASCII*.

Note The asterisks are not part of the encoded data; and you should not include asterisks in *Message* property when creating code 39 barcodes.

See Also

Section 10.2, "Code 39"

Section 10.4, "Code 39 HIBC"

Section 10.3, "Code 39 Full ASCII"

8.13. Comment Property

Description

Returns or sets a string for the human readable comment printed around the barcode symbol.

Syntax

```
HRESULT get_Comment(HANDLE handle, char* lpszComment);  
HRESULT put_Comment(HANDLE handle, const char* lpszComment);
```

Remarks

In addition to the human readable, which always reflects the encoded data and conforms to the standard requirements, you can optionally place a paragraph of text beside the barcode image.

You may adjust margins around the comment, place the comment at the top or bottom, modify the font typeface as well as the size, and change how the text is aligned.

Control characters are not printed.

Note It is now possible to enter multiple paragraphs in the comment by inserting `\n` at the end of each paragraph (except the last one). For example, the string **First paragraph\nSecond Paragraph** renders two paragraphs, as illustrated below:



First line of text
Second line

See Also

Section 8.15, "CommentFont Property"

Section 8.14, "CommentAlignment Property"

Section 8.43, "ShowComment Property"

Section 8.17, "CommentOnTop Property"

Section 8.16, "CommentMarginTop, CommentMarginBottom, CommentMarginLeft, CommentMarginRight Properties"

8.14. CommentAlignment Property

Description

CommentAlignment - returns or sets a value indicating how the comment is aligned.

Syntax

```
HRESULT get_CommentAlignment(HANDLE handle, LONG* pVal);  
HRESULT put_CommentAlignment(HANDLE handle, LONG val);
```

Remarks

This property controls how the text in the comment portion is aligned. Valid alignment choices are listed in the table below:

Table 8.5. CommentAlignment Options

Constant	Value	Description
mbxAlignLeft	0	Left alignment (default). Align the text to the left edge of the comment box.
mbxAlignRight	1	Right alignment. Align the text to the right edge of the comment box.
mbxAlignCenter	2	Center alignment. Align the text to the center of the comment box
mbxAlignJustify	3	Justify alignment. Align the text to both edge of the comment box.

8.15. CommentFont Property

Description

Returns or sets the font for comment text.

Syntax

```
HRESULT get_Font(HANDLE handle, IFontDisp** pVal);  
HRESULT put_Font(HANDLE handle, IFontDisp* val);
```

Remarks

The default font used to draw comment text is typeface of Arial, 8 points.

This property is used to retrieve/set the font used for comment text. To set/retrieve the font used for human readable text, use *Font* property.

In Visual Basic you cannot create a Font object using code like `Dim X As New Font`. If you want to create a Font object, you can use the StdFont object, as in the code below:

```
Dim X As New StdFont  
X.Bold = True  
X.Name = "Arial"  
Dim rc As Long  
rc = put_CommentFont(handle, X)
```


8.16. CommentMarginTop, CommentMarginBottom, CommentMarginLeft, CommentMarginRight Properties

Description

These four properties control the margins around the comment box.

Syntax

```
HRESULT get_CommentMarginTop(HANDLE handle, LONG* pVal);  
HRESULT put_CommentMarginTop(HANDLE handle, LONG val);  
HRESULT get_CommentMarginBottom(HANDLE handle, LONG* pVal);  
HRESULT put_CommentMarginBottom(HANDLE handle, LONG val);  
HRESULT get_CommentMarginLeft(HANDLE handle, LONG* pVal);  
HRESULT put_CommentMarginLeft(HANDLE handle, LONG val);  
HRESULT get_CommentMarginRight(HANDLE handle, LONG* pVal);  
HRESULT put_CommentMarginRight(HANDLE handle, LONG val);
```

Remarks

The above four parameters control the margins around the comment text box. Note that under different design modes, the *CommentMarginLeft* and *CommentMarginRight* have different meanings. For detailed information refer to Chapter 5, *Fundamentals*.

8.17. CommentOnTop Property

Description

Returns or sets a value that determines whether the comment box is placed above or below the symbol.

Syntax

```
HRESULT get_CommentOnTop(HANDLE handle, bool* pVal);  
HRESULT put_CommentOnTop(HANDLE handle, bool val);
```

Remarks

The default value for CommentOnTop is FALSE. To place the comment on top of the image, set this property to TRUE.

8.18. DataMatrixModuleSize Property

Description

Returns or sets a value that determines the width and height of a single cell in the data matrix symbols generated by Morovia Barcode DLL.

Syntax

```
HRESULT get_DataMatrixModuleSize(HANDLE handle, LONG* pVal);  
HRESULT put_DataMatrixModuleSize(HANDLE handle, LONG val);
```

Remarks

The “real estate” unit of a data matrix symbol, the *module*, is always square. This property sets both the width and the height of the square. It affects the overall symbol size.

The default value for **DataMatrixModuleSize** is 20 mils. The property can be any numbers between 1 and 100.

See Also

Section 8.19, “DataMatrixTargetSizeID Property”

8.19. DataMatrixTargetSizeID Property

Description

Returns or sets a value that determines the shape of the DataMatrix symbol produced by Morovia Barcode DLL.

Syntax

```
HRESULT get_DataMatrixTargetSizeID(HANDLE handle, LONG* pVal);
HRESULT put_DataMatrixTargetSizeID(HANDLE handle, LONG val);
```

Remarks

Data Matrix only allows a limit number of combinations of rows and columns. You must set the property to 0 (automatic) or one of the following values:

Table 8.6. DataMatrixTargetSizeID options

Constant	Value	Description
mbxDMTargetSizeAuto	0	Automatic selection of data matrix size
mbxDMTargetSize_12X12	1	Rectangle symbol of 12 by 12 modules
mbxDMTargetSize_14X14	2	Rectangle symbol of 14 by 14 modules
mbxDMTargetSize_16X16	3	Rectangle symbol of 16 by 16 modules
mbxDMTargetSize_18X18	4	Rectangle symbol of 18 by 18 modules
mbxDMTargetSize_20X20	5	Rectangle symbol of 20 by 20 modules
mbxDMTargetSize_22X22	6	Rectangle symbol of 22 by 22 modules
mbxDMTargetSize_24X24	7	Rectangle symbol of 24 by 24 modules
mbxDMTargetSize_26X26	8	Rectangle symbol of 26 by 26 modules
mbxDMTargetSize_32X32	9	Rectangle symbol of 32 by 32 modules
mbxDMTargetSize_36X36	10	Rectangle symbol of 36 by 36 modules
mbxDMTargetSize_40X40	11	Rectangle symbol of 40 by 40 modules
mbxDMTargetSize_44X44	12	Rectangle symbol of 44 by 44 modules
mbxDMTargetSize_48X48	13	Rectangle symbol of 48 by 48 modules
mbxDMTargetSize_52X52	14	Rectangle symbol of 52 by 52 modules
mbxDMTargetSize_64X64	15	Rectangle symbol of 64 by 64 modules
mbxDMTargetSize_72X72	16	Rectangle symbol of 72 by 72 modules
mbxDMTargetSize_80X80	17	Rectangle symbol of 80 by 80 modules
mbxDMTargetSize_88X88	18	Rectangle symbol of 88 by 88 modules
mbxDMTargetSize_96X96	19	Rectangle symbol of 96 by 96 modules
mbxDMTargetSize_104X104	20	Rectangle symbol of 104 by 104 modules
mbxDMTargetSize_120X120	21	Rectangle symbol of 120 by 120 modules
mbxDMTargetSize_132X132	22	Rectangle symbol of 132 by 132 modules

Constant	Value	Description
<code>mbxDMTargetSize_144X144</code>	23	Rectangle symbol of 144 by 144 modules
<code>mbxDMTargetSize_8X18</code>	24	Rectangle symbol of 8 by 18 modules
<code>mbxDMTargetSize_8X32</code>	25	Rectangle symbol of 8 by 32 modules
<code>mbxDMTargetSize_12X26</code>	26	Rectangle symbol of 12 by 26 modules
<code>mbxDMTargetSize_12X36</code>	27	Rectangle symbol of 12 by 36 modules
<code>mbxDMTargetSize_16X36</code>	28	Rectangle symbol of 16 by 36 modules
<code>mbxDMTargetSize_16X48</code>	29	Rectangle symbol of 16 by 48 modules
<code>mbxDMTargetSize_10X10</code>	30	Rectangle symbol of 10 by 10 modules

Data matrix offers limited combinations between the number of rows and columns. A valid selection is called a *data matrix size*. A data matrix symbol can be any one of the 30 shapes listed in table Table 8.6, “DataMatrixTargetSizeID options”. In addition, our software uses value 0 for automatic size selection. The value 0 means “do not care”. Barcode DLL selects the smallest size to fit your data when you set `DataMatrixTargetSizeID` to 0.

Note The size ID for the smallest data matrix shape, `mbxDMTargetSize_10X10`, is 30.

See Also

Section 8.18, “DataMatrixModuleSize Property”

8.20. Font Property

Description

Returns or sets the font for human readable text.

Syntax

```
HRESULT get_Font(HANDLE handle, IFontDisp** pVal);  
HRESULT put_Font(HANDLE handle, IFontDisp* val);
```

Remarks

The default font used to draw comment text is typeface of Arial, 9 points. Generally speaking, you should use a sans-serif font for human readable text. Some industries require OCR-B(Optical Character Recognition Revision B) font to be used.

This property is used to retrieve/set font for human readable text. To set /retrieve font used for comment, use *CommentFont* property.

In Visual Basic you cannot create a Font object using code like `Dim X As New Font`. If you want to create a Font object, you can use the `StdFont` object, as demonstrated in the code below:

```
Dim X As New StdFont  
X.Bold = True  
X.Name = "Arial"  
Dim rc As Long  
rc = put_Font(handle, X)
```

See Also

Section 8.44, "ShowHRText Property"

Section 8.47, "TexAlignment Property"

Section 8.48, "TextOnTop Property"

8.21. I2of5OptionalCheckDigit Property

Description

Deprecated since 3.4.

Syntax

```
HRESULT get_I2of5OptionalCheckDigit(HANDLE handle, bool* pVal);  
HRESULT put_I2of5OptionalCheckDigit(HANDLE handle, bool val);
```

Remarks

Previously, in order to add check digit to an Interleaved 2 of 5 barcode, you need to set this property to TRUE. Interleaved 2 of 5 symbology requires the input to be even length. If this property is TRUE and the input already has even length, the previous implement appends a '0' at the end, and calculate the check digit. This is an unexected behavior for many customers.

In version 3.4 and above, this property has no effect on the barcode created. Whether or not a check digit is required depends on if the length of the input is even or odd. If the length is even, no check digit is added and Barcode DLL encodes as is. If the length is odd, however, the program calculates the check digit and automatically appends at the end to make the whole length even.

See Also

Section 10.14, "Interleaved 2 of 5 (ITF25)"

8.22. LabelWidth, LabelHeight Properties

Description

LabelWidth, LabelHeight - returns or sets the width and height of the working area, in measurement units specified by Measurement.

Syntax

```
HRESULT get_LabelHeight(HANDLE handle, LONG* pVal);  
HRESULT put_LabelHeight(HANDLE handle, LONG val);  
HRESULT get_LabelWidth(HANDLE handle, LONG* pVal);  
HRESULT put_LabelWidth(HANDLE handle, LONG val);
```

Remarks

Under label design mode, you need to set these two properties to the size of working area you plan to work on. The drawing can only be done within the working area; anything beyond the boundary will be clipped out.

If you are working on barcode design mode by setting *AutoLabelSize* to TRUE, you can not set these two properties. The working area expands and shrinks when the barcode image and comment expand or shrink.

Same as all other length properties, the actual value depends on *Measurement* property. If *Measurement* is *mbxMeasureEnglish*, the value is measured in mils(1/1000 inch), otherwise it is in high metric (1/1000 cm).

The default working area under *label design mode* is 2" by 2". When you set *AutolabelSize* to FALSE, Barcode DLL uses the default size for the working area by setting both *LabelHeight* and *LabelWidth* to 2000 mils.

8.23. MaxicodeClass Property

Description

Returns or sets a value that determines the service class for structured carrier message to be encoded in Maxicode symbols generated by Morovia Barcode DLL.

Syntax

```
HRESULT get_MaxicodeClass(HANDLE handle, LONG* pVal);  
HRESULT put_MaxicodeCountryCode(HANDLE handle, LONG val);
```

Remarks

The *class of service code* is a 3-digit number which is defined by the carrier and shipper to indentify the service class.

The default value for this property is 001 .

8.24. MaxicodeMode Property

Description

Returns or sets a value that determines the encoding mode of Maxicode symbols generated by Morovia Barcode DLL.

Syntax

```
HRESULT get_MaxicodeMode(HANDLE handle, SHORT* pVal);  
HRESULT put_MaxicodeMode(HANDLE handle, SHORT val);
```

Remarks

Maxicode specification defines 5 encoding modes from 2 to 6. Mode 2 and 3 are reserved for domestic and international carrier messages. Mode 4 and mode 5 both encode generic data. Between the two of them, mode 5 offers slightly better data security. Other Maxicode properties, such as *MaxicodeClass*, *MaxicodeCountryCode*, are meaningful only under mode 2 and 3. Mode 6 is designated for reader programming purposes.

8.25. MaxicodeCountryCode Property

Description

Returns or sets a value that identifies the country in the structure carrier message encoded in Mode 2 or Mode 3 MaxiCode symbols.

Syntax

```
HRESULT get_MaxicodeCountryCode(HANDLE handle, LONG* lCountryCode);  
HRESULT put_MaxicodeCountryCode(HANDLE handle, LONG countrycode);
```

Remarks

The country code is a 3-digit number which represents a country. For a complete list of country codes, see *ISO standard 3166*.

8.26. MaxicodeZipCode Property

Description

Returns or sets a value for the postal code/zip code used in the structure carrier message under Mode 2 or Mode 3 MaxiCode symbols.

Syntax

```
HRESULT get_MaxicodeZipCode(HANDLE handle, char* lpszValue);  
HRESULT put_MaxicodeZipCode(HANDLE handle, const char* lpszValue);
```

Remarks

The *MaxiCodeZipCode* is the postal code/zip code of the delivery address. Only capital letters and digits are acceptable.

Note: Mode 2 encodes a 9-digit U.S. zip code while Mode 3 encodes a 6-character alpha-numeric postal code. You are not allowed to specify a text with length greater than 9. If *MaxiCodeMode* is 2 but an alpha-numeric *MaxicodeZipCode* is specified, the program uses 000000000 as the *MaxiCodeZipCode*.

8.27. Measurement Property

Description

Returns or sets the measurement unit for all length properties.

Syntax

```
HRESULT get_Measurement(HANDLE handle, LONG* pVal);  
HRESULT put_Measurement(HANDLE handle, LONG val);
```

Remarks

You can set this property to one of the two values:

Table 8.7. Measurement Unit Options

Constant	Value	Description
mbxMeasureEnglish	0	All lengths are in mils (1/1000 inch)
mbxMeasureMetrics	1	All lengths are in 1/1000 cm

Most symbologies use U.S. English measurement unit, which is based on mils (1/1000 inch). Some symbologies are metric oriented, such as *Royal Mail* and *MaxiCode*. To produce barcodes as accurate as possible, we recommend that you stick to the measurement unit specified by the standard to avoid float number calculation.

Note When *Measurement* changes, all length properties change accordingly so that the barcode sizes remain intact.

8.28. Message Property

Description

Returns or sets a string for the message to be encoded.

Syntax

```
HRESULT get_Message(HANDLE handle, char* lppsValue);  
HRESULT put_Message(HANDLE handle, const char* lpszValue);
```

Remarks

Not all symbologies are capable of encoding all characters. Some may only encode numeric data. Some impose a limit on the length of the encoded data. If you encode data with invalid characters or length, Barcode DLL returns an error.

- UPC symbologies (UPC-A, UPC-E, EAN-13, EAN-8)

A UPC symbol may have an optional 2-digit or 5-digit add-on barcode. To create an add-on barcode, separate the main data and the extension data with a vertical bar. For example, the input **1-932111-39-5|55999** produces a Bookland barcode with a 5-digit add-on symbol.

- GS1 DataBar Symbologies (DataBar, Truncated, Limit, Stacked and Stacked Omnidirectional)

All GS1 DataBar symbologies excluding DataBar Expanded encode a 14-digit number called GTIN (Global Trade Identification Number). The input must be exactly 13 or 14 digits. AI (01) is part of human readable but should not go into the input.

- GS1-128 Symbology

To ensure the human readable format is correct, the AI and field ID must be enclosed with parentheses (). If the data is formatted incorrectly you may end with an error. For example, the following data is valid for UCC/EAN 128 message input:

```
(01)12345678901231
```

Additional information is also needed to create a shortest possible barcode. Refer to Section 10.12, "UCC/EAN-128" for more details.

- GS1 DataBar Expanded

The data encoded by DataBar Expanded follow the exact rules that outlined in the above bullet (See GS1-128 Symbology). AI must be enclosed with parentheses ().

- Tilde codes

Tilde code sequence can be used to enter special characters, such as extended ASCII characters and symbology-specific characters if supported. See each Symbology section for details.

8.29. NarrowBarWidth Property

Description

Returns or sets a value for the width of the narrowest module in linear symbologies.

Syntax

```
HRESULT get_NarrowBarWidth(HANDLE handle, LONG* pVal);  
HRESULT put_NarrowBarWidth(HANDLE handle, LONG val);
```

Remarks

This property defines the width of the narrowest element in a linear barcode - a.k.a *X-dimension*. The measurement unit is in either 1/1000 inch or 1/1000 cm depending on the Measurement unit used.

By default the value for this property is 13. The valid range is from 1 to 1000. Industry standards require that the barcodes used in an open system have a X-dimension at least of 10 mils (one-hundredth of inch). If the X-dimension is too small, some scanners may have problems reading the barcode.

This property affects most linear symbologies. Height-modulated postal barcodes, such as POSTNET and Royal Mail barcodes, use fixed pitch thus this property has no effect on these symbologies.

8.30. NarrowToWideRatio Property

Description

Returns or sets the ratio of the wide to narrow bar in a barcode.

Syntax

```
HRESULT get_NarrowToWideRatio(HANDLE handle, double* pVal);  
HRESULT put_NarrowToWideRatio(HANDLE handle, double val);
```

Remarks

Some linear symbologies can have two module widths. The width of the wide one is a fixed multiple of the width of the narrow module (*NarrowBarWidth*). You can choose a value ranging from 2.0 to 3.0 for this ratio.

This property is valid only for Code 39, Code 25, Code 11, Codabar and Interleaved 2 of 5 symbologies. All others ignore this property. We also recommend you set a value between 2.5 to 3.0 so the barcode can be easier to be recognized.

Since the value may impact the readability of the barcode, we highly recommend that you test the barcode readability when you set the value to anything below 2.5.

8.31. PDFAspectRatio Property

Description

Returns or sets a value for the overall height to width ratio of PDF417 barcode generated by Barcode DLL.

Syntax

```
HRESULT get_PDFAspectRatio(double* pVal);  
HRESULT put_PDFAspectRatio(double val);
```

Remarks

The PDFAspectRatio determines the overall shape of the PDF417 symbol and is defined as the overall height to width ratio. Higher values for the Aspect Ratio (greater than 1) produce tall, thin PDF417 bar codes and small values (greater than zero and less than 1) produce short, wide bar codes. A value of 1 produces approximately square bar codes.

The default value for this property is 0.5.

8.32. PDFMaxCols Property

Description

Returns or sets a value for the maximum number of codeword columns allowable in all PDF417 barcodes generated. Since version 3.4, this property is also used to specify the number of symbol per row in GS1 DataBar Expanded symbology.

Syntax

```
HRESULT get_PDFMaxCols(LONG* pVal);  
HRESULT put_PDFMaxCols(LONG val);
```

Remarks

The *PDFMaxCols* and the *PDFMaxRows* properties allow you to set the target number of columns and rows. The *PDFMaxCols* property specifies the maximum number of codeword columns in a PDF symbol. It can be set to a value ranging from 1 to 30.

Since version 3.4, GS1 DataBar Expanded is supported in Barcode ActiveX Professional and Barcode DLL product line. DataBar Expanded can be multiple rows, with each row holding even number of symbol characters. This property is used to specify the number of symbols per row in DataBar symbology. This number must be between 2 and 22 otherwise 22 is used. Because DataBar Expanded can have 22 symbols at the most. Setting to 22 or 0 makes the resulted barcode one row only. When used in DataBar Expanded symbology, the number must be even otherwise it is rounded to the closet even integer.

See Also

Section 8.33, “PDFMaxRows Property”

8.33. PDFMaxRows Property

Description

Returns or sets a value for the maximum number of codeword rows allowable in all PDF417 bar code symbols produced by Morovia Barcode DLL.

Syntax

```
HRESULT get_PDFMaxRows(HANDLE handle, LONG* pVal);  
HRESULT put_PDFMaxRows(HANDLE handle, LONG val);
```

Remarks

The PDFMaxCols and the PDFMaxRows properties allow you to set the target number of columns and rows. The default value for PDFMaxRows is 0, meaning that program automatically picks the number of rows. The valid range for this property is 3 to 90.

See Also

Section 8.32, “PDFMaxCols Property”

8.34. PDFModuleHeight Property

Description

Returns or sets a value for the height of the modules in the PDF417 barcode generated.

Syntax

```
HRESULT get_PDFModuleHeight(LONG* pVal);  
HRESULT put_PDFModuleHeight(LONG val);
```

Remarks

The recommended value for *PDFModuleHeight* is approximately three times the value of *PDFModuleWidth*. You can set this property to any values greater than 10 mils.

The valid range for *PDFModuleHeight* is from 1 to 100. The default value is 30.

See Also

Section 8.35, “PDFModuleWidth Property”

8.35. PDFModuleWidth Property

Description

Returns or sets a value for the width of the modules in the PDF417 barcodes generated.

Syntax

```
HRESULT get_PDFModuleWidth(LONG* pVal);  
HRESULT put_PDFModuleWidth(LONG val);
```

Remarks

The recommended value for *PDFModuleWidth* is between 10 and 30 mils. To achieve the best read rate, we recommend that you set a value which is integral times of the pixel width on the target device (the width of a pixel on a computer display is 10.42 mils and the one on a laser printer is 3.33 mils).

The valid range for this property is between 1 and 100. The default value is 13.

See Also

Section 8.34, “PDFModuleHeight Property”

8.36. PDFSecurityLevel Property

Description

Returns or sets a value for security level used in all PDF417 barcodes generated.

Syntax

```
HRESULT get_PDFSecurityLevel(SHORT* pVal);  
HRESULT put_PDFSecurityLevel(SHORT val);
```

Remarks

The PDFSecurityLevel property allows you to select a PDF417 error correction level from 0 to 8 (or 9 for automatic). Each higher security level up to 8 adds additional overhead to a PDF417 symbol thereby requires large symbol space.

The default value for this property is 9 (automatic).

8.37. PDFTruncatedSymbol Property

Description

Returns or sets a value that determines whether to generate the truncated version of all PDF417 bar codes generated by the Barcode DLL.

Syntax

```
HRESULT get_PDFTruncatedSymbol(bool* pVal);  
HRESULT put_PDFTruncatedSymbol(bool val);
```

Remarks

You can produce a truncated version of PDF417 barcode by setting PDFTruncatedSymbol to TRUE. A truncated PDF417 symbol reduces the stop pattern to a single termination bar. Truncated symbols should be used only in a clean and controlled environment.

The default value for this property is FALSE.

8.38. Picture Property

Description

Readonly property. Returns a snapshot of the drawing in Windows Enhanced Metafile Format (EMF).

Syntax

```
HRESULT get_Picture(IPictureDisp** ppVal);
```

Remarks

The *Picture* property provides a convenient method to retrieve the drawing without first saving it to disk. The picture object contains an enhanced metafile handle which can be passed to *clipboard* or played on a device.

Note after version 3.2, drawing units are measured from the default printer. If you print to a low-resolution printer, set the target printer as the default before retrieving the EMF handle.

8.39. QuietZones Property

Description

Returns or sets a value that determines whether to include quiet zones on the barcodes generated.

Syntax

```
HRESULT get_QuietZones(bool* pVal);  
HRESULT put_QuietZones(bool val);
```

Remarks

In linear barcodes, *quiet zone* is defined as a clear space that precedes the start character of a barcode symbol and follows the stop character. In two-dimensional barcodes quiet zones are clear area around the barcode. The space is required to help scanner determine where the barcode starts and stops.

The width of space added is 10 times the *NarrowBarWidth* value for all linear barcodes, 2 times *PDFModuleWidth* value for PDF417 barcodes, 2 times *DataMatrixModuleSize* value for DataMatrix barcodes and 1 element width for MaxiCode barcodes. Setting this property substantially increase the barcode length for linear symbologies.

You can also use symbol margins to create effective quiet zones. By default Barcode DLL set the symbol margins to 100 mils at 4 directions. When this is the case, you may safely set this property to `FALSE` to better align the comment and human readable text.

8.40. RasterImageResolution Property

Description

RasterImageResolution - returns or sets a value that corresponds to the resolution (in pixels per inch) of the target device when export barcode images to a raster graphic file format (JPEG, GIF, PNG, TIF and BMP).

Syntax

```
HRESULT get_RasterImageResolution(LONG* pVal);  
HRESULT put_RasterImageResolution(LONG val);
```

Remarks

When you export the barcode image into a raster graphics file format such as *JPG*, *GIF* and *PNG*, you are converting the drawing commands (device independent) to an array of pixels which are device dependent. The size of a pixel varies based on the device and usually is measured by dot per inch (dpi). A laser printer usually has a high resolution of 300 dpi while the screen has a low resolution of 96 dpi. As a result, an image may have different physical size when displayed on the screen than printed. The greater the resolution is, the bigger the file size and the accurate of the details. We suggest you set this property to the value which matches your printer.

The default value for this property is 300.

See Also

Section 8.52, "ExportImage Method"

8.41. Rotation Property

Description

Returns or sets a value indicating how to rotate the working area.

Syntax

```
HRESULT get_Rotation(LONG* pVal);  
HRESULT put_Rotation(LONG val);
```

Remarks

This property controls how the working area is rotated. Valid rotation choices are:

Table 8.8. Rotation Options

Constant	Value	Description
mbxRTZeroDegree	0	No rotation
mbxRTAntiClockwise_90	1	Rotate at 90 degrees angle counterclockwise
mbxRTAntiClockwise_180	2	Rotate at 180 degrees angle counterclockwise(upside down)
mbxRTAntiClockwise_270	3	Rotate at 270 degrees angle counterclockwise

8.42. ShowCheckDigit Property

Description

ShowCheckDigit - Determines whether the checksum characters will be shown on the human readable portion. This option is effective to selected symbologies only.

Syntax

```
HRESULT get_ShowCheckDigit(bool* pval);  
HRESULT put_ShowCheckDigit(bool val);
```

Remarks

Different symbologies have different rules regarding check digit. For some symbologies, check digit is part of the data and should always be included in the human readable text. Some symbologies allow optional check digit. Some symbologies require check character not be displayed at all.

- Check digit is part of data and is always displayed.
This category includes UPC-A, UPC-E, EAN-13, EAN-8, Bookland, and UCC/EAN-128. This property has no effect on these symbologies.
- Check digit is required, but not treated as part of data and is never included in the human readable text.
This category includes Code 128, Telepen and Telepen Numeric.
- Check digit is required, and can be optionally included into the human readable text.
This category includes Code 93, Code 11, POSTNET, PLANET and MSI/Plessey.
- Check digit is optional, and can be optionally included in the human readable text.
This category includes Code 39, Code 39 Full ASCII, HIBC.

In version 3.4, there are some notable changes:

- Previously, HIBC barcodes may turn off its check digit in the human readable display (although the check digit always appear in the barcode). After 3.4, the check digit is always included in the human readable text, as required by the standard.
- Before 3.4, whether a UCC/EAN-128 check digit is calculated depends on *UccEanOptionalCheckDigit*. After version 3.4, the check digit is always included in the human readable text.

See Also

Section 8.44, "ShowHRText Property"

Section 8.10, " Code25OptionalCheckDigit Property "

Section 8.11, "Code39OptionalCheckDigit Property"

Section 8.21, "I2of5OptionalCheckDigit Property"

8.43. ShowComment Property

Description

Returns or sets a value that determines whether the control displays the comment portion.

Syntax

```
HRESULT get_ShowComment(bool* pVal);  
HRESULT put_ShowComment(bool val);
```

Remarks

Toggle this property to turn on or turn off the display of the comment portion. Note that if this property is `FALSE`, comment margin properties will not be included during the position calculation of other components, such as working area, bar code image etc. If you want to have these margins participate the calculation and do not want to see the comment, set *Comment* property to an empty string instead.

8.44. ShowHRTText Property

Description

Returns or sets a value that determines whether the control displays the human readable portion.

Syntax

```
HRESULT get_ShowHRTText(bool* pVal);  
HRESULT put_ShowHRTText(bool val);
```

Remarks

Toggle this property to turn on or turn off the display of the human readable text.

Two dimensional barcodes do not have the concept of “human readable”. Consequently this property has no effect on 2D barcodes.

Per standard, UPC-A, UPC-E, EAN-13, EAN-8 barcodes and their supplements should always have the human readable as an integrated part of the image. In versions prior to 3.2, you can not produce those barcodes without visible human readable, even you set *ShowHRTText* to FALSE. This behavior has changed since version 3.2. Unless you have absolute reason to turn it off, you should set *ShowHRTText* to TRUE when creating these types of barcodes.

Note The width of human readable text portion never exceeds the barcode length. If the barcode length is too small, the text will wrap into multiple lines. If this is not desired, reduce the font size, or place the human readable text into comment and adjust comment margin properties to increase the width of comment box so that the text stays in one line.

8.45. Symbology Property

Description

Returns or sets a value indicating the type of the bar code format (symbology) to be generated by the ActiveX control.

Syntax

```
HRESULT get_Symbology(LONG* pVal);
HRESULT put_Symbology(LONG val);
```

Remarks

The Barcode DLL currently supports the following symbologies:

Table 8.9. Symbology Options

Constant	Value	Description
mbxCode39	0	(default) Code 39 (43 character set)
mbxCode39_Full_ASCII	1	Code 39 Full ASCII
mbxCode39_HIBC	2	Code 39 Mod 43 (Health Industry Bar Code)
mbxCodaBar	3	Codabar
mbxCode93	4	Code 93
mbxCode128	5	Code 128
mbxUCC_EAN_128	6	UCC/EAN 128
mbxInterleaved_2of5	7	Interleaved 2 of 5 (ITF25)
mbxUPC_A	8	UPC-A
mbxUPC_E	9	UPC-E
mbxEAN_13	10	EAN/JAN-13
mbxEAN_8	11	EAN/JAN-8
mbxBookland	12	Bookland
mbxTelepen	13	Telepen
mbxTelepenNumeric	14	Telepen Numeric (double density)
mbxPostnet	20	PostNET (barcode used by USPS)
mbxPlanet	21	Planet (used by USPS for package tracking)
mbxRoyalMail	22	Royal Mail (U.K. Postal)
mbxMSI_Plessey	30	MSI/Plessey
mbxCode25	31	Code 25
mbxCode11	32	Code 11
mbxCode11	32	Code 11
mbxDataBar	33	DataBar

Constant	Value	Description
mbxDataBarTruncated	34	DataBar Truncated
mbxDataBarStacked	35	DataBar Stacked
mbxDataBarStackedOmni	36	DataBar Stacked Omnidirectional
mbxDataBarLimited	37	DataBar Limited
mbxDataBarExpanded	38	DataBar Expanded
mbxPDF417	40	PDF 417 (2D symbology)
mbxDataMatrix	41	DataMatrix (2D symbology)
mbxMaxiCode	42	MaxiCode (2D symbology)

Some symbologies only encode certain limited set of characters, such as digits. Some symbologies impose limit of the data length; some require checksum characters. If you are not familiar with the symbologies you are working on, refer to Chapter 10, *Barcode Technologies* to get some hands-on information.

8.46. SymbolMarginTop, SymbolMarginBottom, SymbolMarginLeft, SymbolMarginRight Properties

Description

These four properties control the margins around the symbol boundary (including barcode, human readable and comment).

Syntax

```
HRESULT get_SymbolMarginTop(LONG* pVal);  
HRESULT put_SymbolMarginTop(LONG val);  
HRESULT get_SymbolMarginBottom(LONG* pVal);  
HRESULT put_SymbolMarginBottom(LONG val);  
HRESULT get_SymbolMarginLeft(LONG* pVal);  
HRESULT put_SymbolMarginLeft(LONG val);  
HRESULT get_SymbolMarginRight(LONG* pVal);  
HRESULT put_SymbolMarginRight(LONG val);
```

Remarks

These four parameters control the margins around the symbol (barcode, human readable and comment).

8.47. TextAlignment Property

Description

Returns or sets a value indicating how the human readable text is aligned.

Syntax

```
HRESULT get_TextAlignment(LONG* pVal);
HRESULT put_TextAlignment(LONG val);
```

Remarks

This property controls how the text in the human readable portion is aligned. Valid alignment choices are:

Table 8.10. TextAlignment Options

Constant	Value	Description
mbxAlignLeft	0	Left alignment (default). Align the text with left edge of the comment box.
mbxAlignRight	1	Right alignment. Align the text with the right edge of the comment box.
mbxAlignCenter	2	Center alignment. Align the text with the center of the comment box
mbxAlignJustify	3	Justify alignment. Align the text to both edge of the comment box.

Because of the unique character arrangement in UPC/EAN symbologies, this property does not apply on those symbologies: UPC-A, UPC-E, EAN-13 and EAN-8.

The barcodes below illustrate the effects of *TextAlignment*:



Note When our first Barcode component product was released, the property was misspelled as *TextAlignment* instead of the correct spelling *TextAlignment*. For compatibility reasons we keep using the misspelled word as the property name. Check the spelling when you find that your application did not achieve the desired result.

See Also

Section 8.20, “Font Property”

Section 8.44, “ShowHRText Property”

8.48. TextOnTop Property

Description

Returns or sets a value that determines whether the human readable text is placed above the barcode image or below the image.

Syntax

```
HRESULT get_TextOnTop(bool* pVal);  
HRESULT put_TextOnTop(bool val);
```

Remarks

The default value for TextOnTop is FALSE which places the human readable text below the barcode. To place the human readable text above the barcode, set *TextOnTop* to TRUE.

Two-dimensional symbologies (PDF417, Data Matrix and MaxiCode) do not support human readable text. This property has no effects when the current symbology is PDF417, Data Matrix or MaxiCode.

See Also

Section 8.20, "Font Property"

Section 8.44, "ShowHRText Property"

Section 8.47, "TexAlignment Property"

8.49. UccEanOptionalCheckDigit Property

Description

Deprecated since version 3.4.

Syntax

```
HRESULT get_UccEanOptionalCheckDigit(bool* pVal);  
HRESULT put_UccEanOptionalCheckDigit(bool val);
```

Remarks

Before version 3.4, this

This property kicks in only when all the following conditions are met: (1) the current symbology is set to UCC_EAN_128. (2) the data element contains an AI of 00 (SSCC-18) or 01(SCC-14). (3) the data length is 1 less than the required. The check digit is calculated based on Mod 10 algorithm and appended to the end of the data part. The check digit also appears in the human readable text.

For example, when the property UCC_EAN_128 is set to TRUE, the data **(01)3001234567890** becomes **(01)30012345678906** where the last digit of 6 is the calculated mod10 check digit.

8.50. ZoomRatio Property

Description

Returns or sets the value that determines the actual display size of the control.

Syntax

```
HRESULT get_ZoomRatio(double* pVal);  
HRESULT put_ZoomRatio(double val);
```

Remarks

When the *AutoSize* is FALSE, the container controls the size of the display area. The Barcode DLL displays itself to the full extent in the display area prescribed by the container while keeping the aspect ratio. The user can not set *ZoomRatio* under this mode; instead the user changes the property by dragging the tracking box using the mouse pointer. When the *AutoSize* property is set to TRUE, the Barcode DLL decides the size of the display area. You can change the size of the display area by modifying the natural size, or the *ZoomRatio*. To make sure that barcode created has a high quality, do not set *ZoomRatio* to any values other than 1 at the print time. To modify the barcode size it is highly recommended to do so through length properties, such as *NarrowBarWidth*, *BarHeight*, and *PDF417ModuleHeight* etc.

8.51. About Method

Description

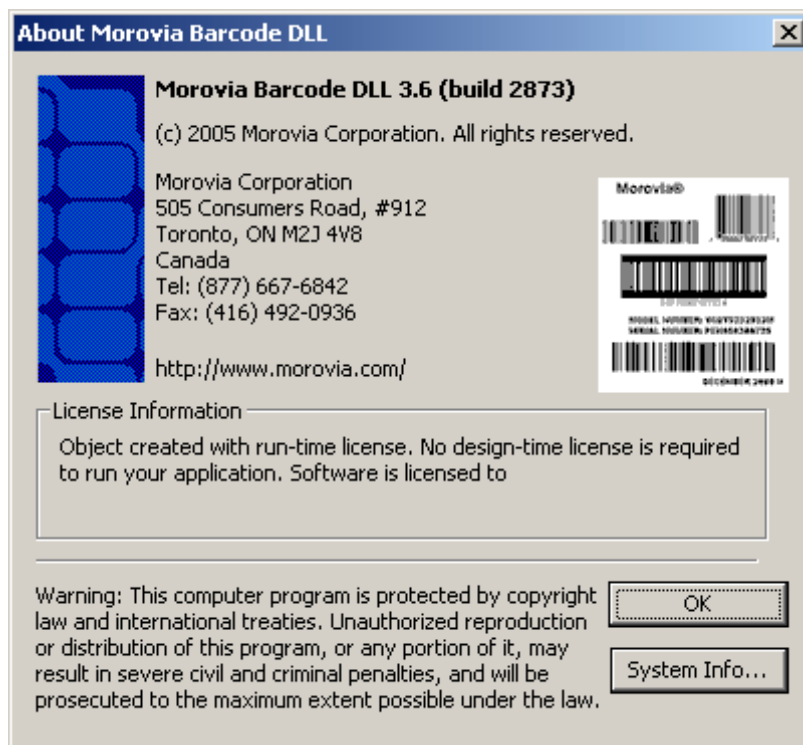
Display the About Dialog.

Syntax

```
void mbxAbout(HANDLE handle);
```

Remarks

The About dialog displays the version information. It also displays the license information used to create the barcode object.



8.52. ExportImage Method

Description

Exports the image to a graphics file with the specified format. This allows other programs to use the barcode images generated.

Syntax

```
HRESULT mbxExportImage(HANDLE handle,
    const char* lpszFileName, LONG imageFormat)
HRESULT mbxExportImage2(HANDLE handle,
    IDispatch* pStream, LONG imageFormat)
```

Remarks

Exports the image into the file specified by the filename in a standard format. Barcode DLL supports the export to the following graphic formats:

Constant	Value	Description
mbxFormatBMP	0	Windows bitmap
mbxFormatJPG	1	JPEG
mbxFormatGIF	2	GIF
mbxFormatTIF	3	TIFF
mbxFormatPNG	5	PNG
mbxFormatEMF	6	EMF (Windows Enhanced MetaFile)
mbxFormatWMF	7	WMF (Windows MetaFile)

Use *ExportImage* to save the barcode image to a disk file or a *Stream* object, with the specified graphic format. If the file already exists, the method overwrites the file.

When Barcode DLL exports to *EMF* format, the resolution of the default printer is used as the basis to create the barcode. If you use the EMF handle to print to a low resolution printer, set this printer as default before calling *ExportImage*.

When Barcode DLL exports to *WMF* format, a high resolution of 1440 dpi is assumed. A high resolution printer is needed to replay the WMF handle.

All other file formats are raster image format. The property *RasterImageResoluton* determines the resolution to use during the rasterization. PNG or GIF are preferred formats because they are lossless and compressed well for barcode images.

Note The WMF file format does not contain frame size information. To find out the exact size, retrieve *LabelHeight* and *LabelWidth* from the object. The bitmap rendering process utilizes printer drivers. To avoid rounding errors between the drawing units (target resolution) and the device units (for text measurement), we recommend that you have the printer driver installed on the computer that creates the barcode image. For example, if you create bitmap images targeting a thermal printer (203 dpi), you should install such a printer driver on the computer you are working. During the *ExportImage* process, the printer driver is consulted to make sure that the bitmap created reflects the actual print out. You do not need

to connect the printer to the computer. The barcode quality is ensured when *RasterImageResolution* is set to a value compatible with screen or an installed printer device.

Note The behavior of this method changed in version 3.6.

In versions before 3.6, the Save method overwrite the file if it exists. After 3.6, an existing file is overwritten only when all conditions below are true:

- None of the following attribute bits is set: `read-only`, `hidden` and `system`.
- The extension of the file must be one of the following: `bmp`, `jpg`, `jpeg`, `gif`, `tif`, `tiff`, `png`, `emf` or `wmf`.

If any of the conditions is not met, the method returns an error.

However, if the path specified does not exist, a new file will be created, and the rules on the attributes and extensions do not apply.

See Also

Section 8.40, “*RasterImageResolution* Property”

Chapter 6, *Working with Low Resolution Devices*

8.53. Load Method

Description

Load the Barcoce ActiveX object from a file, in binary or XML format.

Syntax

```
HRESULT mbxLoad(HANDLE handle,  
               const char* lpszFileName, LONG persistFormat);
```

- **lpszFileName**

A string that represents the complete path name of the file where the object is to be saved.

- **PersistFormat**

A PersistFormatEnum value that specifies the format in which the object is to be saved (XML or Binary). The default value is mbxPersistBinary.

Remarks

The Load method loads the property data and draws the barcode image based on the data loaded. The PersistFormat can be one of these values:

Table 8.11. PersistFormat Options (Load method)

Constant	Value	Description
mbxPersistBinary	0	Binary Format
mbxPersistXML	1	XML Format

8.54. Save Method

Description

Saves the Barcode ActiveX object in a file, in binary or XML format.

Syntax

```
HRESULT mbxSave(HANDLE handle,
               const char* lpszFileName, LONG persistFormat);
```

- **lpszFilename**

A string that represents the complete path name of the file where the object is to be saved.

- **PersistFormat**

A *PersistFormatEnum* value that specifies the format in which the object is to be saved (XML or Binary). The default value is `mbxPersistBinary`.

Remarks

The *Save* method saves the property data into a disk file or a *Stream* object in binary or XML format. The *PersistFormat* can be one of these values:

Table 8.12. PersistFormat Options (Save method)

Constant	Value	Description
<code>mbxPersistBinary</code>	0	Binary Format
<code>mbxPersistXML</code>	1	XML Format

If the file already exist, the method overwrites the file.

Note The behavior of this method changed in version 3.6.

In versions before 3.6, the *Save* method overwrites the file if it exists. After 3.6, an existing file is overwritten only when all conditions below are true:

- None of the following attribute bits is set: `read-only`, `hidden` and `system`.
- The extension of the file must be one of the following: `bax`, `bax3`, `mbx` or `xml`.

If any of the conditions is not met, the method returns an error.

However, if the path specified does not exist, a new file will be created, and the rules on the attributes and extensions do not apply.

See Also

Section 8.53, “Load Method”

Chapter 9. Error Handling

Barcode DLL reports two kinds of errors: operational errors and encoding errors. They are handled in different ways in Barcode DLL.

Operational errors occur when an operation fails or a property is set to an invalid value. The error is reported through the return code of the function. Your program can then retrieve the error message through `GetLastBarcodeError` function.

You can tell if the function operates as expected by comparing the return code with zero. If the return code is a negative number, the operation fails.

For performance reasons Barcode DLL does not attempt to encode every time that a property changes, especially for two dimensional symbologies. There is no fixed algorithm to tell whether the size specified can hold the data until the program encodes with all the properties specified. The encoding errors are reported at the time of the actual rendering. An image containing the error codes and messages is rendered in the place of the barcode, as the image below illustrates:

```
DataMatrix Encoding  
Error 30. The  
target size must be  
increased to hold  
the data. The  
minimum size ID is  
4 (18X18) .
```

The error message tells that the size specified is too small to hold the data encoded. It also tells you in order to encode the current data, the minimum size ID is 11 (`mbxDmTargetSize_40X40`). Normally you handle encoding errors at design time by modifying the properties.

9.1. Error Codes

Operational Error codes specific to Barcode DLL are listed below:

Table 9.1. Error Codes (Operational)

Value	Description
6101	This symbology is not supported by the current version of Morovia Barcode DLL.
6102	The message is either empty or contains invalid character for the chosen symbology.
6103	The measurement unit must be either English (0) or Metric (1).
6104	The Zoom Ratio should ranged between 0.1 and 100
6105	The NarrowToWideRatio should be between 2.0 and 3.0
6107	Failed to overwrite file or create the new file. It may be caused by insufficient privilege or disk is full.
6108	Invalid Raster Image Resolution. The resolution must be greater than 50.
6109	The label size you specified is invalid, or you can not edit the size under the current mode.

Value	Description
6112	The current symbology required fixed length. The length you specified does not meet this requirement.
6113	The message string contains characters that can not be encoded under the current symbology.
6114	The object can not be loaded from the media specified. See Error Log for details.
6115	The object failed to save itself to the specified media.
6116	Invalid value for BarHeight property.
6117	The MaxicodeZipCode property must consist a valid zip code with up to 6 alpha-numeric characters.
6118	The MaxicodeCountryCode must between 000 and 999
6119	The value range for MaxicodeMode must be between 2 and 6.
6120	The value range for MaxicodeClass must be between 000 and 999.
6121	Error happens during the PDF417 encoding process. For more information retrieve the Error object.
6122	Can not generate bitmap handle. Possible reasons include insufficient memory, too big size of the bitmap etc.

Table 9.2. Error Codes (Encoding)

Value	Description
10	The AI portion of an EAN-128 structure must be numeric. (EAN-128)
11	The data portion of the EAN-128 data must be alpha-numeric. (EAN-128)
12	Invalid UCC/EAN-128 structure. Check whether the AI and field ID are enclosed with parenthesis. (EAN-128)
13	The length of the data portion of the EAN-128 is incorrect. (EAN-128)
14	Based on the AI, the data portion must be numeric. However, this is not the case in the message encoded. (EAN-128)
20	No control block was found in the message. (PDF417)
21	Invalid Raster Image Resolution. The resolution must be greater than 50.
22	Error in the segment index. Either the segment index is not numeric, or the index is bigger than the total segment count. (PDF417)
23	Error in total segment count field. (PDF417)
24	Incorrect MacroPDF417 optional field format. (PDF417)
30	The size required is too small to hold the data. (PDF417, Data Matrix)
40	GLI must be 6 digits in the range between 0 and 811,799. (PDF417)
41	The security level is too high to encode all the data. (PDF417)
50	A required structural append field is missing. (Data Matrix Structural Append)

Value	Description
51	No control block was found in the message. (Data Matrix Structural Append)
52	Error in the sequence indicator field. Either it is not numeric, or the value is out of range. (Data Matrix Structural Append)
53	Error in the total number of total symbols field. This field can only be a number between 1 and 16. (Data Matrix Structural Append)
54	Invalid file ID. Either it contains invalid characters, or the value is out of range. (Data Matrix Structural Append)
60	Error in the specified ECI. Either it contains invalid characters, or the value is out of range. (Data Matrix ECI)
70	Macro 05 or Macro 06 should appear at the first position of the input and should not be used in conjunction with structural append. (Data Matrix Macro)
71	Reader programming ~3 should appear at the first position of the input and should not be used in conjunction with structural append. (Data Matrix Reader Programming)
200	Carrier message prefix detected but not all required fields are present. (MaxiCode)

Chapter 10. Barcode Technologies

10.1. Introduction

Barcode has been widely adopted across all major industries. A conventional barcode is a machine readable symbol consisting of a series of parallel, adjacent bars and spaces. The basic barcode structure features leading and trailing quiet zones, a *start character*, one or more data characters, one or more *check characters* (optional) and a *stop character*.

Barcode has a long development history that spanned half a century. During the evolution process, many formats have been developed and adopted by industries. Around a dozen of them are actively used today.

The term “symbology” is the scientific name for the barcode format. Different symbologies have different characteristics, such as the encoding efficiency and character set. The *character set* defines what kind of data the symbology encodes. Typically there are four types of character sets: (1) numeric. Only digits can be encoded. (2) alpha-numeric. The symbology is capable of encoding numbers, letters plus several punctuations. (3) full ASCII. All characters in the ASCII set, with value between 0 and 127, can be encoded. (4) Binary. Binary character set includes all 256 characters in a 8-bit single byte character set. Most of two dimensional symbologies are capable of encoding arbitrary binary data.

Some symbologies may impose length requirements. For example, UPC-A encodes the numeric data of exact 12 digits.

Two dimensional symbologies are usually capable of encoding thousand of characters.

Barcode DLL supports the following symbologies:

Table 10.1. Symbologies Supported by Barcode DLL

Symbology	Also Known As
Code 39	Code 3 of 9, AIAG, USS Code 39
Code39 Full ASCII	Code 39 Extended
HIBC Code 39	HIBC, LOGMARS
Codabar	Rationalized Codabar
Code 93	
Code 128	USS-128, C-128
GS1-128	UCC 128, EAN 128
Interleaved 2 of 5	ITF, ITF-14, I 2 of 5
UPC-A	
UPC-E	
EAN-13	
EAN-8	
Bookland	
Telepen, Telepen Numeric	
Postnet, Planet	

Symbology	Also Known As
Royal Mail	UK Postal Code, RM4SCC
MSI/Plessey	Plessey Code
Code 25	Industry 25, Code 2 of 5
Code 11	
PDF417	
DataMatrix	
MaxiCode	UPS barcode
DataBar	RSS-14, DataBar-14, DataBar Omindirectional
DataBar Truncated	RSS-14 Truncated, DataBar-14 Truncated
DataBar Stacked	RSS-14 Stacked, DataBar-14 Stacked
DataBar Stacked Omnidirectional	RSS-14 Stacked Omnidirectional
DataBar Limited	RSS-14 Limited
DataBar Expanded	RSS Expanded, DataBar Expanded Stacked, RSS Expanded Stacked

You can purchase symbology standards directly from AIM Inc. The web address of AIM is <http://www.aimglobal.org>¹.

10.2. Code 39

Code 39 (also known as USS Code 39, Code 3 of 9) is the first alpha-numeric symbology developed to be used in non-retail environment. It is widely used to code alphanumeric information, such as the model number etc. It is designed to encode 26 upper case letters, 10 digits and 7 special characters:

A, B, C, D, E, F, G,
H, I, J, K, L, M, N, O, P, Q,
R, S, T, U, V, W, X, Y, Z
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
-, ., *, \$, /, +, %, SPACE.

Each code 39 symbol begins with a start character and ends with a stop character. Traditionally the start/stop characters are represented by asterisk character (*). Due to this reason, some applications include asterisks in the human readable text. The asterisks are not part of the encoded message and should not appear within the message.

Code 39 allows an optional checksum digit based on modulo 43 algorithm. The health industry has adopted the use of the check character for health applications and these types of barcodes are often referred as **HIBC**.

Property *Code39OptionalCheckDigit* specifies whether an additional *check digit* should be added to the barcode. Another property, *Code39StartStopChars*, when it is set to TRUE, adds the traditional start/stop characters (*) to the beginning and the end of the human readable text.

¹ <http://www.aimglobal.org>



10.3. Code 39 Full ASCII

The Code 39 Full ASCII (sometimes also referred as Code 39 extended) is an extension to normal code 39. It is capable of encoding all 128 *ASCII* characters. It uses shift characters to combine two normal code 39 characters to encode a character not in the normal code 39 character set. The barcode generated is compatible with normal code 39 so the scanner must be configured to Full ASCII mode to read the barcode correctly.



Code 39 Full ASCII supports entering control characters using special character input method. You can use a back slash \ plus 3-digit decimal ASCII code to enter a control character. For example, the following input encodes digits **123**, followed by a **NUL** character and letters **abc**:

Data Input: 123\000abc



10.4. Code 39 HIBC

Code39 HIBC is exactly the same as normal Code39, with `Code39OptionalCheckDigit` property set to true. The standard also says that the starting character in the message must be a plus (+) symbol. Barcode DLL automatically adds this plus sign (+) if the encoded message does not meet this requirement.

The HIBC standard requires that the checksum digit to appear in the human readable. To satisfy this requirement, your program should explicitly set property `ShowcheckDigit` to TRUE to create a compliant HIBC symbol.

Setting `Code39OptionalCheckDigit` to FALSE does not affect the resulted barcode.



10.5. Codabar

Codabar is a variable length symbology which encodes a character set of 16 letters (0-9, -, \$, :, /, +). It is dubbed as NW-7 in Japan. You may choose one of these four start/stop characters in your symbol: A, B, C and D. If you do not specify the start/stop characters, Barcode DLL uses A and B as the start/stop characters, respectively. No check digit is required.



A 0 345-67 9 900001 1234 B

10.6. Code 93

Code 93 is a variable length symbology that is capable of encoding all 128 ASCII characters. Code 93 offers higher density than Code 39. It has the same native character set as Code 39 (43 characters) but it uses additional 4 shift characters to encode other characters. Code 93 features 2 checksum characters. Start/Stop characters are also required.

Code 93 supports special character input method. See Code 39 Full ASCII section for details on how to escape control characters.



505 Consumers Road

10.7. MSI/Plessey, Code 25 and Code11

These are rather obsolete symbologies which only encode numeric data. There is no advantage to use them except for application backward compatibility. More information can be found at Morovia barcode library at <http://www.morovia.com/education/>².

² <http://www.morovia.com/education/>

10.8. UPC-A, UPC-E and UPC Supplements

The UPC-A barcode is the most common and well-known symbology in North America. You can find it on the cartons of virtually every consumer goods in your local supermarket, as well as books, magazines, and newspapers. A short form is called UPC-E. Each symbol may have 2-digit or 5-digit supplement to encode additional information.



UPC-A encodes 11 digits of numeric data along with a trailing check digit, for a total of 12 digits of barcode data.

A UPC-A number consists of four areas: (1) The Number System; (2) The manufacturer code; (3) the product code; (4) The check digit. Normally the number system digit is printed to the left of the barcode, and the check digit to the right. The manufacturer and product codes are printed just below the barcode, separated by the guard bar.

The UPC-E barcode is the short form representation of a UPC number. It reduces the data length from 12 digits to 6 digits by compressing the extra zeros. It is suited for identifying products in small packages.

A UPC-E barcode has 6 digits with an implied number system 0. The first 5 digits are calculated based on a conversion algorithm described below. The last digit is the check digit of the original UPC-A symbol.

Both UPC-A and UPC-E symbols allow for a supplemental two or five digit add-on barcode. This add-on barcode usually encodes the price or a sequence number. To include a supplemental message, append it to the main message with a vertical bar (|) separating it from the main message. The supplemental message must consist of exact two or five digits.

Table 10.2. Examples of UPC-A, UPC-E and Supplement

Message	Symbol Created
90123678812	UPC-A
90123678812 02	UPC-A with 2-digit add on
0123456	UPC-E
0123456 95000	UPC-E with 5-digit add on

10.9. EAN-13, EAN-8 and EAN Supplements

EAN is designed by the International Article Numbering Association (EAN) in Europe. It is an extension to UPC-A to include the country information. The only difference between UPC-A and EAN-13 is that the number system in UPC-A is a single digit varying from 0 through 9 whereas an EAN-13 number system consists of two digits ranging from 00 to 99.



EAN-13 encodes 12 digits of numeric data along with a trailing check digit, for a total of 13 digits data.

An EAN-13 number consists of four areas: (1) the Number System; (2) the manufacturer code; (3) the product code; (4) the check digit. Normally the number system digit is printed to the left of the barcode, and the check digit to the right. The manufacturer and product codes are printed just below the barcode, separated by the guard bar.

EAN-8 is the short version of EAN-13, the same as UPC-E vs. UPC-A. While they look very similar, some differences exist. UPC-E does not explicitly encode the first digit (NS) while EAN-8 encodes all 8 digits. From barcode encoding/decoding perspective, an EAN-8 is not compatible with UPC-E. Moreover, although a UPC-E number can be converted back to UPC-A, this is not the case for EAN-8. There is no defined method for conversions between EAN-13 to EAN-8. An EAN-8 number is assigned in the same way as EAN-13.

An EAN-8 number contains 7 digits of message plus 1 check digit. The first two or three digits identify the numbering authority; the remaining 4 or 5 digits identify the product.

Table 10.3. Examples of EAN-13, EAN-8 and Supplement:

Message	Symbol Created
97802161594	EAN-13
978020161594 02	EAN-13 with 2-digit add on
71245126	EAN-8
71245126 95000	EAN-8 with 5-digit add on

10.10. ISBN/Bookland

The International Standard Book Number (ISBN) has been invented for more than 30 years. It has experienced exponential growth and remarkable success. Today, every book, magazine, cassette and CD bear an ISBN number. Every item to be sold in bookstore is required to furnish an ISBN. The ISBN is used extensively by publishers, retailers as well as libraries to manage inventory. The ISBN is represented through an EAN barcode, a.k.a. Bookland barcode plus an optional 5-digit (2-digit for magazines) add-on.



An ISBN is a 10 digit number preceded by the letters ISBN. The text is usually printed with an OCR-A font. The ten-digit number is divided into four parts of variable length, which are separated by hyphens or spaces. The four parts are Group Identifier, Publisher Identifier, Title Identifier and check digit respectively. Note that the length of each part is not fixed, though the total length must be 10.

A Bookland symbol may have an optional 2-digit or 5-digit add-on symbol. To add the supplement, add them at the end of the main message and separate the two parts with a vertical bar(|), in the same way as the UPC and EAN supplements. For example, to encode an ISBN number 0-201-61595-9 with pricing information 53995, set the Message property to 0-201-61595-9|53995.

Note on 13-digit ISBN

Beginning on January 1, 2007, all 10-digit ISBNs are required to be re-expressed as a 13-digit number (EAN-13). To convert a 10-digit ISBN to 13-digit EAN number, drop the last checksum digit of the 10-digit ISBN number and add prefix 978 at the beginning. Calculate the EAN-13 check digit based on the result and append this checksum digit to the end of the result. To create the barcode, use EAN13 symbology instead and assign the 13-digit ISBN number to the *message* property.

10.11. Code 128



Code 128 is a high-density alpha-numeric symbology. Since introduced in early 1980s, it has gained wide popularity in many industries. UCC/EAN derives its retail carton tracking standard UCC/EAN 128 based on Code 128 symbology.

Code 128 is a variable length, continuous symbology with multiple element widths. Every Code 128 symbol has a check character. Each character is encoded with three bars and spaces, in total 11 modules.

In the most recent standard *ISO/IEC 15417*, Code128 is extended to encode all 256 characters of a single-byte character set³. The default character set is *ISO 8859-1* (Latin Alphabet No. 1).

Code 128 standard also defines four function codes for special purposes. FNC2 is used to tell barcode reader to store the data and transmit with next symbol; FNC4 is used as a latch code word to switch into extended ASCII mode. FNC3 is reserved for future use. FNC1 is used in UCC/EAN128 to act as UCC/EAN-128 identifier and field delimiter.

10.11.1. How Barcode DLL Implements the Code128

Barcode DLL allows encoding all 256 characters as well as 4 special symbol characters: FNC1, FNC2, FNC3 and FNC4.

Internally Code128 defines 3 character sets (A, B and C) to allow efficient encoding. Each character set contains 103 characters (including special symbol characters). A code128 symbol starts with one character set and latches to a different set with a latch codeword. Since these three character sets overlap, it is possible to get different barcodes with the same data encoded.

To allow space efficiency, during the encoding process, Barcode DLL selects the proper character sets and inserts necessary shift characters to make the symbol generated as short as possible.

Code128 requires a checksum character to ensure the data integrity. The checksum character has no meaning to the end user. Barcode DLL does not transmit the checksum digit back to the human readable text.

Note Barcode DLL always tries to create the shortest barcode. For example, if Barcode DLL sees that some portion of the data is best fit encoded with Code128 C it automatically select character set C. The end user does not have the control on how the data is encoded.

10.11.2. Tilde Codes

Under some circumstances it is necessary to represent some characters with an ASCII-only format. This kind of representation format is called escape sequence. For example, the four special Code128 symbol characters, FNC1~FNC4, do not have corresponding ASCII values. Consequently the only way to enter them into the *Message* property is through their escaped forms. Some applications and programming environments may only accept printable ASCII characters, and control characters must be escaped.

Tilde code sequence is the only escape method supported in Barcode DLL version 3.2 and above. The special character input method (escaping a character using a back-slash character followed by 3-digit character value) present in previous versions is now deprecated.

³Published in year 2000, this standard is relative new to the industry. Not all scanners in the market support this feature.

The tilde code consequences used in Code128 are listed as below:

~dnnn

When nnn corresponds to a numeric value between 0 and 255, the tilde code sequence represents a character with value equal to nnn. For example, ~d032 represents a space character.

~~

Represents a tilde (~) character.

~1

Represents a FNC1 character. The tilde escape sequence can appear anywhere in the input.

~2

Represents a FNC2 character.

~3

Represents a FNC3 character.

~4

Represents a FNC4 character. FNC4 is used to encode extended ASCII characters. You do not need to enter the FNC4 in most circumstances. Just pass the *extended characters* you'd like to encode.

~X

Represents a character value from 0 to 26. Replace the X like in the following example ~@ means character ascii 0, ~A means character 1, ~B means character 2, ~C means character 3 and so on.

Note Due to the fact that each symbology encodes different character set, the tilde code sequence varies from symbology to symbology. Refer to the tilde codes section of each symbology to understand how to escape the character.

10.12. UCC/EAN-128

10.12.1. Introduction

UCC/EAN-128⁴ encodes structured data proposed by various industry standard bodies and authorized by GS1 organization. Each data type is identified with a numeric value, called *Application Identifier (AI)*. Multiple AIs and values can be concatenated together into one barcode, such as:

(01)19421123450011(15)991231(10)101234

The data above contains multiple AIs and values:

- 01 indicates that the value followed 19421123450011 is a SCC-14 number.⁵
- 15 is the AI for Sell by Date. The value followed 991231 indicates that the Sell By Date is December 31, 1999.
- 10 is the AI for Batch Number. According to the specification, it is a variable length AI. Here the value is 101234.

The AI value determines the meaning and the length of the value part. Many of them encode a predefined length of data. For example, the SCC14 requires exact 14 digits and the Sell By Date requires exact 6 digits in YYMMDD format.

When the data length can be derived from AI, it is not necessary to add field separator (FNC1) in the barcode to separate two adjacent fields. However, if the first field has a variable data length, such a field separator is required. And in many applications it is often desirable to have a field separator between two fixed-length fields. The Code128 symbol character FNC1 serves this purpose.

10.12.2. How Barcode DLL Implements UCC/EAN-128

To understand each data field, Barcode DLL requires you to enter the data in a special format. The AI must be enclosed with parentheses. From the AI Barcode DLL knows whether a field has a fixed length or a variable length. For all variable-length fields, Barcode DLL inserts field separator unless it ends the input.

Barcode DLL also performs data validation on the AI and the data, if the AI is known to the program.

Table 10.4. List of Known AIs

AI	Name	Constraint	Short Name
00	SSCC (Serial Shipping Container Code)	n2+n18	SSCC
01	Global Trade Item Number	n2+n14	GTIN
02	GTIN of Trade Items Contained in a logistic unit	n2+n14	CONTENT
10	Batch or lot number	n2+an..20	BATCH/LOT
11	Production date (YYMMDD)	n2+n6	PROD DATE
12	Due date (YYMMDD)	n2+n6	DUE DATE
13	Packaging date (YYMMDD)	n2+n6	PACK DATE
15	Best before date (YYMMDD)	n2+n6	BEST BEFORE or SELL BY
17	Expiration date (YYMMDD)	n2+n6	USE BY OR EXPIRY
20	Product variant	n2+n2	VARIANT

⁴As UCC/EAN organization changed its name to GS1, now the symbology is also called as GS1-128.

AI	Name	Constraint	Short Name
21	Serial number	n2+an..20	SERIAL
22	Secondary data for specific health industry products	n2+an..29	QTY/DATE/BATCH
240	Additional product identification assigned by the manufacturer	n3+an..30	ADDITIONAL ID
241	Customer part number	n3+an..30	CUST. PART NO.
242	Made-to-Order Variation Number	n2+n..6	Variation Number
250	Secondary serial number	n3+an..30	SECONDARY SERIAL
251	Reference to source entity	n3+an..30	REF. TO SOURCE
253	Global Document Type Identifier	n3+n13+n..17	DOC. ID
254	GLN Extension component	n3+an..20	GLN EXTENSION
30	Variable count	n2+n..8	VAR. COUNT
310n-369n	(Trade and logistic measurements)	n4+n6	--
337n	Kilograms per square metre	n4+n6	KG PER m2
37	Count of trade items contained in a logistic unit	n2+n..8	COUNT
390(n)	Amount payable - single monetary area	n4+n..15	AMOUNT
391(n)	Amount payable - with ISO currency code	n4+n3+n..15	AMOUNT
392(n)	Amount payable for a Variable Measure Trade Item - single monetary unit	n4+n..15	PRICE
393(n)	Amount payable for a Variable Measure Trade Item - with ISO currency code	n4+n3+n..15	PRICE
400	Customer's purchase order number	n3+an..30	ORDER NUMBER
401	Consignment number	n3+an..30	CONSIGNMENT
402	Shipment Identification Number	n3+n17	SHIPMENT NO.
403	Routing code	n3+an..30	ROUTE
410	Ship to - deliver to Global Location Number	n3+n13	SHIP TO LOC
411	Bill to - invoice to Global Location Number	n3+n13	BILL TO
412	Purchased from Global Location Number	n3+n13	PURCHASE FROM
413	Ship for - deliver for - forward to Global Location Number	n3+n13	SHIP FOR LOC
414	Identification of a physical location Global Location Number	n3+n13	LOC No
415	Global Location Number of the Invoicing Party	n3+n13	PAY
420	Ship to - deliver to postal code within a single postal authority	n3+an..20	SHIP TO POST

AI	Name	Constraint	Short Name
421	Ship to - deliver to postal code with Three-Digit ISO country code	n3+n3+an..9	SHIP TO POST
422	Country of origin of a trade item	n3+n3	ORIGIN
423	Country of initial processing	n3+n3+n..12	COUNTRY - INITIAL PROCESS.
424	Country of processing	n3+n3	COUNTRY - PROCESS.
425	Country of disassembly	n3+n3	COUNTRY - DISASSEMBLY
426	Country covering full process chain	n3+n3	COUNTRY - FULL PROCESS
7001	NATO stock number	n4+n13	NSN
7002	UN/ECE meat carcasses and cuts classification	n4+an..30	MEAT CUT
703(s)	Approval number of processor with ISO country code	n4+n3+an..27	PROCESSOR # s4
7003	Expiration Date and Time	n4+n10	EXPIRY DATE/TIME
8001	Roll products - width, length, core diameter, direction, and splices	n4+n14	DIMENSIONS
8002	Electronic serial identifier for cellular mobile telephones	n4+an..20	CMT No
8003	Global Returnable Asset Identifier	n4+n14+an..16	GRAI
8004	Global Individual Asset Identifier	n4+an..30	GIAI
8005	Price per unit of measure	n4+n6	PRICE PER UNIT
8006	Identification of the component of a trade item	n4+n14+n2+n2	GCTIN
8007	International Bank Account Number	n4+an..30	IBAN
8008	Date and time of production	n4+n8+n..4	PROD TIME
8018	Global Service Relation Number	n4+n18	GSRN
8020	Payment Slip Reference Number	n4+an..25	REF No
8100	GS1-128 Coupon Extended Code - NSC + Offer Code	n4+n1+n5	-
8101	GS1-128 Coupon Extended Code - NSC + Offer Code + end of offer code	n4+n1+n5+n4	-
8102	GS1-128 Coupon Extended Code - NSC	n4+n1+n1	-
90	Information mutually agreed between trading partners (including FACT DIs)	n2+an..30	INTERNAL
91-99	Company internal information	n2+an..30	INTERNAL

If the AI is not listed in the table above ⁶, Barcode DLL can not know whether its data length is fixed or variable. Thus, Barcode DLL treats the data as if its data length is variable and inserts a field separator FNC1 when this field does not end the symbol.

For example, suppose that you set *Message* to (01) 19421123450011(8019)123456(15)051210. Barcode DLL understands that 01 AI requires fixed-length 14 digits data and AI 15 requires fixed-length 14 digits data. However, Barcode DLL does not understand AI 8019 and treats this field as if it has a variable length. Barcode DLL inserts a field separator at the end of this field (before AI 15).

Assuming that AI 8019 requires a fixed data length, you can tell Barcode DLL that the field has a fixed data length by appending a tilde character ~ at the end of the field. For example, you can assign the value below:
(01)94211234500122(8019)123456~(15)051210

When Barcode DLL sees the ~, it treats the current field as fixed-length.

On the other side, if a known AI has a fixed data length but you'd like to have a field separator at the end of the field, you can do so by adding an exclamation character at the end of field, such as:

(01)94211234500122!(8019)123456(15)051210

It forces a field separator to appear after the SCC14 number even AI 01 has a fixed data length and the field separator is not required. Sometimes this field separator is desirable because it is easier for the application to parse the input.

See the table below for the comparison among results produced by different inputs. The[GS] is the scanner output for FNC1 character.

Barcode	Data input/Scanner output
 <p>(01)94211234500122(8019)123456(15)051210</p>	<p>(01)94211234500122(8019)123456~(15)051210 0194211234500122801912345615051210</p>
 <p>(01)94211234500122!(8019)123456(15)051210</p>	<p>(01)94211234500122!(8019)123456(15)051210 0194211234500122[GS]8019123456[GS]15051210</p>
 <p>(01)94211234500122(8019)123456(15)051210</p>	<p>(01)94211234500122(8019)123456(15)051210 01942112345001228019123456[GS]15051210</p>

10.12.3. Auto Check Digit

Before version 3.4, Barcode DLL calculates mod10 check digits on SCC-14 and SSCC-18 numbers, but only when the property *UccEanOptionalCheckDigit* is TRUE. This behavior has been changed since version 3.4. Now the program calculates mod 10 check digit automatically, regardless the value of *UccEanOptionalCheckDigit*. This renders this property useless.

Barcode DLL performs check digit calculation on those AIs: 00, 01, 02, 410, 411, 412, 413, 414, 415, and 8018.

10.12.4. Input Format

⁶This table was updated in version 3.4 to reflect the changes made since the first version came out.

To create the barcode correctly you must enclose the AI with parentheses (). Barcode DLL only accepts numeric AI values. It reports an error when encountering a non-numeric character in the AI part. If AI does not appear in the known list (see the table above), and you do not want the data treated as variable length, you should tell so by appending a tilde character at the end of the field.

Sometimes, it is desirable to have the data separated by spaces in the human readable text. For example, you may like to see the human readable text (8101) 0 54321 1200(21)123456 instead of (8101)0543211200(21)123456. You can create the desirable human readable text by entering the *message* exactly like the one you'd like the human readable to be. Barcode DLL ignores the spaces during the encoding, but preserves them in the human readable text, as the one below illustrates:



10.12.5. Validation

Barcode DLL performs the following validations during the encoding process:

- Check whether the AI is numeric.
- Check whether a data part follows the AI.
- If the AI is known to Barcode DLL and requires a fixed length of data part, check if the data part has the correct length.
- Check whether the AI is enclosed with parentheses.
- If the AI is known to Barcode DLL and requires only numeric or alpha-numeric data, check if the data part meets the requirement.
- If the AI is known to Barcode DLL and requires variable length of the data, check if the length of the data exceeds the maximum size allowed.

10.12.6. Non-standard Application

If your application does not pass the validation, you can not use UCC/EAN-128 to encode the data. Nevertheless, since UCC/EAN-128 encoding is based on Code128 symbology, you can encode the data directly with Code128. Here are several hints you may consider when converting the EAN-128 data into a Code128 input:

- An EAN-128 barcode starts with a FNC1 character. FNC1 can be entered with tilde code sequence ~1.
- If you'd like to have the field separator encoded between two adjacent fields, using FNC1 character to separate two fields.
- Code128 is capable of encoding spaces. Do not enter spaces in the input if you do not want them appear in the barcode.
- You may use tilde code sequences to enter extended ASCII characters. See Section 10.11, "Code 128" for details.

For example, Code128 with message ~18101054321120021123456 produces the exact barcode as the one using UCC/EAN-128 with message (8101)0 54321 1200(21)123456.



Some non-standard applications do not encode the FNC1 at the starting message. When this is the case, remove ~1 at the beginning of the input.

10.13. DataBar Symbology Family

DataBar family formerly referred to as Reduced Space Symbology, or RSS, adopted its official new name *GS1 DataBar* on February 12, 2007. The GS1 board, formerly known as UCC/EAN organization, has declared that “GS1 DataBar symbols and GS1 Application Identifiers shall be available in all trade item scanning systems beginning Jan 1, 2010.”⁷

GS1 DataBar is really a family of bar code symbologies. Some are very small, intended for produce and small consumer packages. And some are larger, intended to carry more data needed for identifying variable-measure foods and the required content on coupons. Some can be read omnidirectionally, which makes them perfectly suitable for POS applications.

Table 10.5. GS1 DataBar Family

Variant	Data Encoded	POS	Applications	Sample Barcode
DataBar Omnidirectional	14-digit GTIN	Yes	Packaged goods	
DataBar Stacked Omnidirectional	14-digit GTIN	Yes	Packaged goods, Produce	
DataBar Expanded	Any GS1-128 data, up to 74 digits or 41 alphanumeric	Yes	Variable-measure food, Coupons	
DataBar Expanded Stacked	Any GS1-128 data, up to 74 digits or 41 alphanumeric	Yes	Variable-measure food, Coupons	
DataBar Truncated	14-digit GTIN	No	Health care item	
DataBar Stacked	14-digit GTIN	No	Health care item	
DataBar Limited	14-digit GTIN	No	Health care item	

⁷Dubbed as “GS1 DataBar Sunrise 2010.” For more information, see <http://www.gs1.org/databar/>.

Among the seven variants, four, DataBar-14, DataBar Stacked Omni, DataBar Expanded and DataBar Expanded Stacked were designed and specifically to work at retail POS because they can be omnidirectionally read. The remaining three, DataBar Truncated, DataBar Stacked, and DataBar Limited, are not recommended to work at retail POS and were design for very very small products (such as healthcare items).

All DataBar symbologies except DataBar Expanded and DataBar Expanded Stacked require 13 digit or 14 digits as input (the GTIN number). The last check digit is not actually encoded into the barcode. Scanners are required to calculate the check digit and transmit it together upon reading the barcode.

Note The GS1-128 application identifier for GTIN, 01, is required to transmit back with 14-digit data. Therefore, a standard complaint scanner will transmit 0104412345678909 upon reading a DataBar barcode with GTIN number 04412345678909 encoded.

10.13.1. What is GTIN?

GTIN is the acronym for Global Trade Item Number, a 14-digit number that identifies trade items developed by GS1 organization. This number have many names, such as SCC-14 (Serial Container Code), UCC-14.

GTIN can be derived from UPC-A or EAN-13 numbers. The first digit is package indicator. Digit '0' and '9' have special meanings here - '0' often means that there is one item in the box, and '9' indicates a variable measure item. The package indicator is followed by GS1 company prefix (assigned by GS1) and item number (assigned by the company). They should be in total of 12 digits. This portion is the same as the first 12 digits in an EAN-13 number, or '0' plus the first 11 digits in a UPC-A number. The last digit is checksum, which is calculated based on Mod10 algorithm on previous 13 digits.

Because UPC-A and EAN-13 numbers can be thought as special cases of GTINs (the package indicator is '0'), a 14-digit GTIN unquely identifies any trade item (a single item or a container).

GTIN is often depicted using Interleaved 2 of 5 or GS1-128 symbologies. This is expected to change as GS1 is endorsing DataBar. The benefits of using DataBar is that it produces more compact barcodes, espically when comparing with UPC-A and EAN-13 symbols. Furthermore, DataBar Expanded allows additional information to be encoded, such as serial number, weight or price.

10.13.2. Barcode Height

DataBar Truncated and DataBar Stacked symbols have their height fixed to its X-dimension. That is, you can not change the height by modifying *BarHeight* property. They are always 13X.

In DataBar Stacked Omnidirectional and DataBar Expanded Stacked symbols, the overall size of the barcode (excluding human readable text and other elements) is not the same as *BarHeight*, because multiple rows exist.

To achieve omnidirectional scannability and standard conformance, the following minimum height should be observed for DataBar-14, DataBar Stacked Omnidirectional, and DataBar Expanded ($X=NarrowBarWidth$):

- DataBar: 33X
- DataBar Stacked Omni: 33X
- DataBar Limited: 10X
- DataBar Expanded: 34X
- DataBar Expanded Stacked: 34X

10.13.3. Human Readable Text

The widths of stacked symbols (DataBar Stacked etc.) can be very small to hold the human readable text in one line. When this happens, the human readable text will wrap into multiple lines. If this is not desired, turn the human readable text off and set it to the comment. You can set comment margin properties *CommentMarginLeft* and *CommentMarginRight* to adjust the width of comment block.



Normal (comment off, human readable on)



human readable off, comment on,
CommentMarginLeft=200 mils

10.13.4. DataBar Expanded and DataBar Expanded Stacked

DataBar Expanded and DataBar Expanded Stacked usually encode a GTIN number plus additional information, such as price, weight, expiration date and so on. Moreover, any GS1-128 data can be encoded in DataBar Expanded and DataBar Expanded Stacked.

Although they are referred separately, DataBar Expanded Stacked is a superset of DataBar Expanded. Any DataBar Expanded symbols are also DataBar Expanded Stacked symbols. Therefore, in our implementation we use DataBar Expanded for both cases, you create DataBar Expanded Stacked symbols by setting symbols per row value to a non-zero value.

Because the support for DataBar Expanded is added in version 3.4, and we did not want to add additional properties to break backward compatibility, we choose an existing property, *PDFMaxCols* for this purpose. By default, *PDFMaxCols* is set to zero, which creates a non-stacked DataBar Expanded barcode. To create a DataBar Expanded Stacked barcode, set this property to an even number between 2 and 22.⁸

10.13.4.1. Input Format

For all DataBar symbologies except DataBar Expanded and DataBar Expanded Stack, the input is required to be a 13-digit GTIN number. You can also enter 14 digits, however, the last digit is ignored. Excessive input will be truncated.

The input format for DataBar Expanded and DataBar Expanded Stacked is identical to the one specified in Section 10.12, "UCC/EAN-128". AIs must be enclosed in parentheses. Barcode DLL will parse the data according to rules set by Application Identifiers. For example, input **(01)90012345678908(3103)012233(15)081231** is valid, which encodes the following information: GTIN 0012345678908, weight 12.233Kilogram, and production date Dec. 31, 2008.

In the input for DataBar Expanded, spaces can appear as part of input, and they are preserved in the human readable text. However, spaces are not part of the data and are not encoded into the barcode.

⁸A DataBar Expanded symbol can hold 22 symbol characters. Setting *PDFMaxCols* to 22 or a large number effectively creates non-stacked barcodes.

10.14. Interleaved 2 of 5 (ITF25)

Interleaved 2 of 5 is a high-density numeric symbology. Some applications require a modulo 10 checksum digit at the end of the message. Interleaved 2 of 5 uses an “interwinded” method to create barcodes and consequently it requires the data length to be even. In order to meet this requirement, Barcode DLL appends a MOD10 check digit when it finds that the input is in odd length. Otherwise, it encodes the data as is.

Note that this behavior has changed since version 3.4. Previously, check digit is added only when property *I2of5OptionalCheckDigit* is TRUE, otherwise a '0' is appended instead. The new implement allows you to enter 13 digits SCC-14 number to get a complete barcode with the check digit.

If the check digit is added, it always appears in the human readable text.

The input for Interleaved 2 of 5 allow spaces. The spaces are preserved in the human readable text but not encoded into the barcode. Fro example, the barcode below is created on input 0 07 70007 0723. Note that the last digit '9' is the check digit, which is calculated by the program.

Interleaved 2 of 5 is widely used to encode Shipping Container Code (SCC-14), which contains exact 14 digits. When it is used for encoding SCC-14 numbers, it is also called *ITF-14*.



You can add bear bars to the barcode by setting *BearerBars* to TRUE.

10.15. POSTNET



POSTNET (Postal Numeric Encoding Technique) encodes a US numeric address code (also called Zip code) which may contain 5, 9 or 11 digits - frequently referred as *Zip*, *Zip+4* and *Zip+6*.

POSTNET is a height-modulated symbology which encodes the data in the height of the barcode instead of the width. Barcode DLL produces POSTNET barcode based on *USPS* standard. The height of each bar and the *pitch* between two adjacent bars are fixed and can not be modified - changing *NarrowBarWidth* and *BarHeight* yields no effect. Although Barcode DLL produces human readable if you desire, keep in mind that *USPS* standard does not allow human readable text under the barcode.

Barcode DLL accepts non-numeric input but filters them out at the time of the encoding. It adjusts the length by adding trailing zeros to meet the length requirement. You may take the advantage by assigning the full address line to the *Message* instead of passing only digits. For example, data input **Monterey Park, CA 91755-1688** yields an identical barcode as message **917551688**.

10.16. PDF 417

Figure 10.1. Example PDF417 Barcode



PDF417 is a multi-row, variable-length symbology with high data capacity and error-correction capability. PDF417 offers some unique features which make it the widely used 2D symbology. A PDF417 symbol can be read by linear scanners, laser scanners or two-dimensional scanners. PDF417 is capable of encoding more than 1100 bytes, 1800 text characters or 2710 digits. Large data files can be encoded into a series of linked PDF417 symbols using a standard methodology referred to as *Macro PDF417*.

The data is encoded using one of three compaction modes: *Text compaction mode*, which encodes alphanumeric characters and punctuations; *Binary compaction mode*, which encodes all 8-bit characters; *Numeric compaction mode*, which achieves the highest density by only allowing digits. The default mode is Text compaction mode. Using special code words, the compaction mode can be switched from one to another. Barcode DLL automatically selects the compaction mode based on data encoded and shifts accordingly.

Each PDF417 symbol contains 2 to 512 error correction code words corresponding to error correction level 0 (the least) to 8 (the highest).

10.16.1. Security Level

In PDF417 the security level is selectable. You can specify a value between 0 and 9 for *PDF417SecurityLevel*. Value 9 means *automatic* and the program selects the security level based on the data encoded and the recommendation from the PDF417 specification.

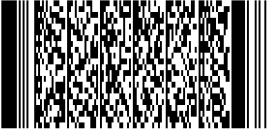
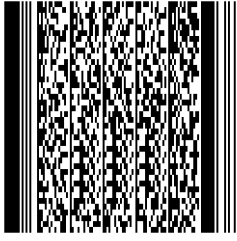
10.16.2. Size Control

There are several properties determining the size and the shape of the symbol. **PDFMaxRows** sets the maximum number of rows allowable and **PDFMaxCols** sets the maximum number of columns. A PDF417 symbol can have 30 columns and 60 rows. It should be pointed out that the row and column here really mean code words, instead of modules. The width of the representation of a code word is much longer than its height. Value 0 allows Barcode DLL to select the value based on the amount of the data and the *aspect ratio*.

The data capacity is directly linked to the number of columns and rows. Setting *PDFMaxRows* and *PDFMaxcols* to small values results smaller data capacity. When the program is unable to encode the data within the limits, an error is reported.

Other related properties include **PDFModuleHeight**, **PDFModuleWidth** and **PDFAspectRatio**. The smallest unit in a PDF417 symbol is called a module. The *PDFModuleWidth* and *PDFModuleHeight* reflect the height and width of the module respectively.

PDFAspectRatio impacts the shape of the final symbol. The *PDFAspectRatio* is defined as the ratio of the height to the overall width of the symbol. Barcode DLL locates the solution that close matches the specified value. Note: in many cases to match the aspect ratio Barcode DLL has to increase the overall symbol size. Smaller *PDFAspectRatio* value usually produces more compact symbols.

		
PDFAspectRatio=0.5 (default)	PDFAspectRatio=0.2	PDFAspectRatio=1.0

10.16.3. Input Format

PDF417 is capable of encoding all characters with ASCII values between 0 and 255. Depending on programming environment you may need tilde codes to escape some characters.

When creating PDF417 barcodes you can use the following tilde codes:

~dnnn

When nnn corresponds to a numeric value between 0 and 255, the tilde code sequence represents a character with value equal to nnn. For example, ~d032 represents a space character.

~~

Represents a tilde (~) character.

~2

Indicates that a MacroPDF417 control block follows. The ~2 tilde codes format is a Morovia extension.

~3

Indicates the start of a GLI block. This escape sequence must be followed by exact 6 digits, which corresponds to the GLI value.

~X

Represents a character value from 0 to 26. Replace the X like in the following example ~@ means character ascii 0, ~A means character 1, ~B means character 2, ~C means character 3 ...

10.16.4. Truncated PDF

In a relatively *clean* environment where label damage is not likely, the right column indicators can be omitted and the stop pattern can be reduced to one module bar. This truncation reduces the data overhead and saves some space at the cost of performance and robustness.

Figure 10.2. Truncated PDF417 Barcode



To produce truncated PDF417 symbols, set the property *PDF417TruncatedSymbol* to TRUE.

10.16.5. Global Label Identification (GLI)

GLI was introduced to allow output data stream to have interpretations different from the default character set (*ISO8859-1*). Since version 3.2, Barcode DLL has been capable of encoding GLI values. A GLI can be any number between 0 and 999999. The tilde code sequence ~7nnnnnn is used to enter the ECI value. The tilde code sequence can appear at any places of the input, provided that exact 6 digits follows ~7. For example to start an interpretation 10, enter ~7000010.

10.16.6. Macro PDF417

Using Macro PDF417, large amount data are splitted into several file segments and encoded into individual symbols. To create Macro PDF417 symbols, you need to enter the control block information using ~2 tilde code sequence. A sample input looks like this:

```
12345678901234567890~2[3][LA-CONFIDENTIAL][6][fn:part2|ts:199044|ad:Justin Power|fs:110990]
```

Syntax

The tilde code sequence for Macro PDF417 control block information is as follows:

```
~2[SI][FID][TS][fn:string|...]
```

The ~2 must appear at the end of the message. The data after the control block is ignored. The first three fields are required. The last field is optional and can contain several additional sub-fields.

Segment Index (SI)

In Macro PDF417, each symbol represents a segment of the whole file. To rebuild the whole file, the segment must be constructed in proper order. The value of segment index is 0 based. For example, for a file divided into k segments, the segment index can be any number between 0 and k-1.

The value allowed for SI is between 0 and 99.

File ID (FID)

All symbols belong to the same group have the same file ID. The File ID can be any string, such as archive2. Although the standard does not set a limit on the length of the File ID, keep in mind that the control block reduces the overall symbol capacity.

Total Segments (TS)

The “total segments” field is required for every symbol in the group. It should remain constant among all symbols.

Optional Fields

Macro PDF417 defines several optional fields to encode additional file information such as file name, timestamp, file size and checksum. All these fields must be at the end of the control block. If more than two optional fields are present, they should be separated with vertical bars |. Within a field, a colon : divides the name part and value part.

The acceptable field names are listed below:

Table 10.6. Optional Fields in Macro PDF417

Field Designator	Field Name (abbreviated)	Field Name (normal)	Comment	Data Type
0	fn	filename	File Name	string
1	sc	segmentcount	Segment Count	number
2	ts	timestamp	Time Stamp	number
3	sd	sender	Sender	string
4	ad	addressee	Addressee	string
5	fs	filesize	File Size	number
6	cs	checksum	Checksum	number

For example, **[fn:archive1.zip|ts:20051231|sd:user@example.com|cs:9901234]** encodes 4 optional fields: file name (archive1.zip), time stamp (20051231), sender (user@example.com) and checksum (9901234).

10.17. Data Matrix

Figure 10.3. Example Data Matrix Barcode



Data Matrix is high density two dimensional symbology capable of encoding up to 2,000 characters of data. It used to have a group of error correction schemes from ECC00 to ECC140; now the standard mandates ECC200 to be used. Barcode DLL creates ECC200 Data Matrix barcodes.

10.17.1. Enhanced Feature Support

In version 3.2, several optional features are added to Barcode DLL:

- Extended Channel Interpretations. This optional feature enables characters from other character sets (e.g. Arabic, Cyrillic and Greek) and other data interpretations or industry-specific requirements to be represented. This feature requires reader support.
- FNC1 character. The FNC1 character, when appearing at the start of the symbol, indicates the data contains a specific industry format authorized by AIM. It can also appear in other positions acting as a field separator.
- Macro Character 05 and 06. Data matrix provides a means of abbreviating an industry specific header and trailer in one symbol character.
- Reader Programming. A reader programming character indicates that the symbol encodes a message used to program the reader system. Requires reader support.
- Structural Append. The Structural Append feature enables encoding large amount of data using multiple symbols. Requires reader support.

10.17.2. Size Control

Data Matrix defines 30 different sizes. Most sizes are square, and a couple of them are rectangle. Regardless the final shape, the “real estate” unit, called module, is always square.

The size id, number of row and columns as well as the data capacity are listed in the table below.

Table 10.7. Data Matrix Symbol Sizes

Size ID		Symbol Size		Data Capacity		
Enum	Value	Row	Column	Numeric	Alpha-numeric	Binary
mbxDMTargetSize_10X10	30	10	10	6	3	1
mbxDMTargetSize_12X12	1	12	12	10	6	3
mbxDMTargetSize_14X14	2	14	14	16	10	6
mbxDMTargetSize_16X16	3	16	16	24	16	10
mbxDMTargetSize_18X18	4	18	18	36	25	16
mbxDMTargetSize_20X20	5	20	20	44	31	20

Size ID		Symbol Size		Data Capacity		
Enum	Value	Row	Column	Numeric	Alpha-numeric	Binary
mbxDMTargetSize_22X22	6	22	22	60	43	28
mbxDMTargetSize_24X24	7	24	24	72	52	34
mbxDMTargetSize_26X26	8	26	26	88	64	42
mbxDMTargetSize_32X32	9	32	32	124	91	60
mbxDMTargetSize_36X36	10	36	36	172	127	84
mbxDMTargetSize_40X40	11	40	40	228	169	112
mbxDMTargetSize_44X44	12	44	44	288	214	142
mbxDMTargetSize_48X48	13	48	48	348	259	172
mbxDMTargetSize_52X52	14	52	52	408	304	202
mbxDMTargetSize_64X64	15	64	64	560	418	278
mbxDMTargetSize_72X72	16	72	72	736	550	366
mbxDMTargetSize_80X80	17	80	80	912	682	454
mbxDMTargetSize_88X88	18	88	88	1152	862	574
mbxDMTargetSize_96X96	19	96	96	1392	1042	694
mbxDMTargetSize_104X104	20	104	104	1632	1222	814
mbxDMTargetSize_120X120	21	120	120	2100	1573	1048
mbxDMTargetSize_132X132	22	132	132	2608	1954	1302
mbxDMTargetSize_144X144	23	144	144	3116	2335	1556
mbxDMTargetSize_8X18	24	8	18	10	6	3
mbxDMTargetSize_8X32	25	8	32	20	13	8
mbxDMTargetSize_12X26	26	12	26	32	22	14
mbxDMTargetSize_12X36	27	12	36	44	31	20
mbxDMTargetSize_16X36	28	16	36	64	46	30
mbxDMTargetSize_16X48	29	16	48	98	72	47

In Barcode DLL you use **DataMatrixTargetSizeID** property to set the size you desire. If the property is set to 0, Barcode DLL picks up the size that fits the data encoded.

Previous to version 3.2, when *DataMatrixTargetSizeID* is too small to encode the whole data, Barcode DLL automatically increases the overall size. This behavior has changed in version 3.2. Now the program reports an error instead.

If *DataMatrixTargetSizeID* is more than holding the data, extra padding characters are added to the barcode. It is sometimes desirable if you want to have all the symbols created have the same size at the same time the data encoded vary from symbol to symbol.

10.17.3. Module Size

The property *DataMatrixModuleSize* determines both the width and height of the smallest unit - a module. By default it is 20 mils. Same as all other length properties, the real value depends on *Measurement*.

10.17.4. Input Format

Data Matrix is capable of encoding all characters in a single-byte character set, plus some symbol-specific characters. Depending on the programming environment you may need tilde codes to escape some characters. When creating data matrix barcodes, you can use the following tilde codes:

~dnnn

When nnn corresponds to a numeric value between 0 and 255, the tilde code sequence represents a character with value equal to nnn. For example, ~d032 represents a space character.

~~

Represents a tilde (~) character.

~1

Represents a FNC1 character. The tilde escape sequence can appear anywhere in the input.

~2

Indicates that a structural append control block follows. The ~2 tilde codes format is a Morovia extension.

~3

Represents a symbol character which means that message followed is used for reader programming. This escape sequence must appear at the beginning of the input.

~5

Represents a symbol character which encodes Macro 5 abbreviation. Must appear at the beginning of the message.

~6

Represents a symbol character which encodes Macro 6 abbreviation. Must appear at the beginning of the input.

~7

Indicates the start of an ECI block. This escape sequence must be followed by exact 6 digits, which corresponds to the ECI value.

~X

Represents a character value from 0 to 26. Replace the X like in the following example ~@ means character ascii 0, ~A means character 1, ~B means character 2, ~C means character 3 ...

10.17.5. Macro 5 and 6

Data Matrix provides a way of abbreviating two industry specific header and trailer in one symbol character. This feature exists to reduce the overall symbol size. They must appear at the beginning of the input. You can use ~5 and ~6 to escape them respectively.

Table 10.8. Macro 5 and 6

Tilde Sequence	Name	Interpretation	
		header	trailer
~5	05 Macro	[>][RS]05[GS]	[RS][EOT]
~6	06 Macro	[>][RS]06[GS]	[RS][EOT]

10.17.6. Extended Channel Interpretation (ECI)

ECI was introduced to allow output data stream to have different interpretations different from the default character set (ISO8859-1). Since version 3.2, Barcode DLL has been capable of encoding ECI values.

An ECI can be any number between 0 and 999999. The tilde code ~7nnnnnn is used to enter the ECI value. The tilde code sequence can appear at any places of the input, but there must be exact 6 digits following ~7. For example to start an interpretation of 10, enter ~7000010.

10.17.7. Structural Append (SA)

The structural append feature allows up to 16 symbols in a structure. A capable reader can either buffer the contents of each symbol until all symbols are read.

To encode structural append, you must supply these items for each symbol:

- Symbol Sequence Indicator (SI). The sequence indicator is 1-based index which identify the position of this particular symbol in the group. Can be any number between 1 and 16.
- Total number of symbols (TS). This value indicates the number of total symbols. Can be any number between 1 and 16. The value should be consistent among all symbols in the group.
- File Identification Number(FID). Identify the symbol group. This number must remain the same among all the symbols in the group.

The tilde code sequence is expressed in the following format:

~2[SI][FID][TS]

For example, tilde code sequence ~1[1][126][6] indicates that the current symbol belongs to a group with file identification number as 126, and there are 6 symbols in total in this group.

The ~2 tilde code sequence must appear at the end of the input. All three fields are required and must be enclosed with square brackets ([and]) and must follow the tilde code ~2.

File ID (FID)

The File ID is a number remaining the constant among all symbols in a group. It uniquely identifies the symbol group. The value for this field should be between 1 and 64516.

Sequence Indicator (SI)

Sequence Indicator is the 1-based index number of the current symbol. In a group with total 10 symbols, the first symbol has the SI of 1 and the last has the SI of 10.

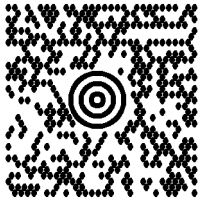
Total Number of Symbols

The Total Number of symbols indicates how many symbols in the group.

10.18. MaxiCode

MaxiCode is a fixed-size (1.1inch x 1.054 inch nominal) two-dimensional symbology made up of offset rows of hexagonal elements around a unique circular finder pattern. A MaxiCode symbol has 884 hexagonal modules arranged in 33 rows with each row containing up to 30 modules. The maximum data capacity for MaxiCode is 93 characters. The unique design enables the symbol quickly picked up by the scanners.

Figure 10.4. Example MaxiCode Barcode



MaxiCode is used by United Parcel Service (*UPS*) for package tracking.

MaxiCode defines 6 modes that determines that how data should be interpreted. The mode 0 and 1 are no longer used. Mode 4 and 5 are used to encode "raw data" with mode 5 offers a slight higher data error correction. Mode 2 and 3 are used to encode "structure message" which comprises two parts: **Primary Message** and **Secondary Message**. The *Primary Message* encodes a postal code, 3-digit country code and 3-digit class of service code. The *Second Message* encodes other data.

Table 10.9. MaxiCode Modes

Mode	Description
mode 2	Structured Carrier Message - Numeric Postal Code (up to 9 digits)
mode 3	Structured Carrier Message - Alphanumeric Postal Code(up to 6 characters)
mode 4	Raw Data, Standard Error Correction
mode 5	Raw Data, Enhanced Error Correction
mode 6	Reader Programming Mode

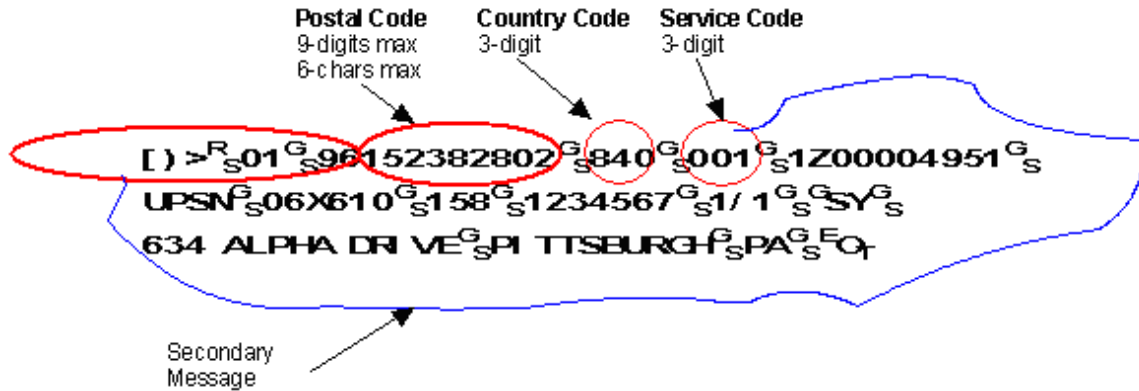
10.18.1. Barcode DLL implementation

Barcode DLL implements the MaxiCode based on *ISO/IEC 16023*. This ISO standard defines three standard fields in the primary message portion. the standard does not define the data structure for the secondary message. UPS adds many fields in the secondary message portion; and because the limit of data capacity, UPS uses a compression algorithm to put the whole fields into the secondary message. Barcode DLL encodes the secondary message as is. To produce a UPS compliant symbol, you need to consult UPS documentation for those additional fields and the compression algorithm.

When the symbol is encoded under mode 2 and 3, MaxiCode properties such as **MaxicodeClass**, **MaxicodeCountryCode**, **MaxicodeZipcode** are used unless the message starts with a UPS carrier prefix (see below). When the symbol is encoded under other modes, these properties are not placed into the symbol.

The MaxiCode requires the printing equipment having at least 200 dpi in resolution. Rasterizing images with a lower resolution will not produce a quality barcode.

10.18.2. Message Structure



If the message start with the standard carrier prefix ([]>[RS] [GS]01[GS]YY), the related properties (*MaxiCodeClass*, *MaxicodeCountryCode* and *MaxicodeZipCode*) will be ignored during the encoding process. The primary and secondary messages are retrieved as follows: The first nine data characters []>[RS] [GS]01[GS]YY are extracted to be encoded in the secondary message. The next three data elements, representing the postal code, country code and service class code respectively are extracted from the source data. The remaining string of data is then encoded in the secondary message after the header []>[RS] [GS]01[GS]YY (excluding three RS characters which separate the three data elements). For example, the message above will be broken into the Primary Message and Secondary Message like this:

- **Primary Message:**
 Postal Code - 152382802
 Country Code - 802
 Class of service Code - 001
- **Secondary Message:**
 []>[RS] [GS]01[GS]961Z00004951[GS]
 UPSN[GS]06X610[GS]158[GS]1234567[GS]1/1
 [GS]Y[GS]634 ALPHA DRIVE[GS]PITTSBURGH
 [GS]PA[GS] [EOT]

10.18.3. Input Format

When creating Maxicode symbols, you can use the following tilde codes:

Table 10.10. Tilde Codes (MaxiCode)

tilde code	description
~dnn	When nnn corresponds to a numeric value between 0 and 255, the tilde code sequence represents a character with value equal to nnn. For example, ~d032 represents a space character.
~~	Represents a tilde (~) character.
~2	Indicates that a structural append control block follows. See the Structure Append section for more details.
~7	Indicates the start of an ECI block. This escape sequence must be followed by exact 6 digits, which corresponds to the ECI value.
~X	Indicates the start of an ECI block. This escape sequence must be followed by exact 6 digits, which corresponds to the ECI value.

Note In version 3.2, the special character format (a back slash followed by 3 digit ASCII value) is no longer supported.

10.18.4. Extended Channel Interpretation (ECI)

ECI was introduced to allow output data stream to have different interpretations different from the default character set (ISO8859-1). Started from version 3.2, Barcode DLL is capable of encoding ECI values.

An ECI can be any number between 0 and 999999. The tilde code ~7nnnnnn is used to enter the ECI value. The tilde code sequence can appear at any places of the input, but there must be exact 6 digits following ~7. For example to start an interpretation 10, enter **~7000010**.

10.18.5. Structural Append (SA)

The structural append feature allows up to 8 symbols in a structure. A capable reader can either buffer the contents of each symbol until all symbols are read.

To encode structural append, you must supply these items for each symbol:

- Symbol Sequence Indicator (SI). The sequence indicator is 1-based index which identify the position of this particular symbol in the group. Can be any number between 1 and 8.
- Total number of symbols (TS). This value indicates the number of total symbols. Can be any number between 1 and 8. The value should be consistent among all symbols in the group.

The tilde code sequence is expressed in the following format:

~2[SI][TS]

For example, tilde code sequence **~1[1][6]** indicates that the current symbol is the first symbol in a group with 6 symbols in total.

The ~2 tilde code sequence must appear at the end of the input. All three fields are required. They must be enclosed within a pair of square brackets [] and must follow the tilde code sequence ~2.

Chapter 11. Technical Support

Morovia offers a wide variety of support services. To help you save time and money when you encounter a problem, we suggest to try to resolve the problem by following the options below in the order shown.

- Consult the documentation. The quickest answer to many questions can be found in the Morovia product documentation.
- Review the tutorial and sample applications. The tutorial steps you through the development process for a typical barcode application. The sample applications provide working code examples in several programming languages. All sample applications are extensively commented.
- Access Morovia Online. Morovia Online provides a knowledge base which documents the frequently asked questions and a web forum.

The web address for knowledge base is <http://support.morovia.com>. And you can ask question at support forum at <http://forum.morovia.com>.

- Contact Morovia Technical Support Services. The Technical Support service is provided for free up to 180 days after the purchase. Email Morovia support engineers at support@morovia.com.

Note If you purchased your software from our reseller, check to see if they provide support services before contacting Morovia.

Support services and policies are subject to change without notice.

Appendix A. Component Software License Agreement

End-User License Agreement IMPORTANT-READ CAREFULLY: This End User License Agreement (this "EULA") contains the terms and conditions regarding your use of the SOFTWARE (as defined below). This EULA contains material limitations to your rights in that regard. You should read this EULA carefully and treat it as valuable property.

This EULA

1. Software Covered by this EULA. This EULA governs your use of the Morovia Corporation ("MOROVIA") component software product(s) enclosed or otherwise accompanied herewith (individually and collectively, the "SOFTWARE"). The term "SOFTWARE" includes, to the extent provided by MOROVIA: 1) any revisions, updates and/or upgrades thereto; 2) any data, image or executable files, databases, data engines, computer software, or similar items customarily used or distributed with computer software products; 3) anything in any form whatsoever intended to be used with or in conjunction with the SOFTWARE; and 4) any associated media, documentation (including physical, electronic and on-line) and printed materials (the "Documentation").
2. This EULA is a Legally Binding Agreement between You and MOROVIA. If you are acting as an agent of a company or another legal person, such as an officer or other employee acting for your employer, then "you" and "your" mean your principal, the entity or other legal person for whom you are acting. However, importantly, even if you are acting as an agent for another, you may still be personally liable for violation of copyright laws.

YOUR LICENSE TO DEVELOP AND TO DISTRIBUTE

As provided in more detail below, this EULA grants you two licenses: 1) a license to use the SOFTWARE to develop other software products (the "Development License"); and 2) a license to use and/or distribute the Developed Software (the "Distribution License"). Both of these licenses (individually and collectively, the "Licenses") are explained and defined in more detail below.

1. Definitions. The following terms have the respective meanings as used in this EULA:

"Network Server" means a computer with one or more computer central processing units (CPU's) that operates for the purpose of serving other computers logically or physically connected to it, including, but not limited to, other computers connected to it on an internal network, intranet or the Internet.

"Web Server" means a type of Network Server that serves other computers more particularly connected to it over an intranet or the Internet.

"Developed Software" means those computer software products that are developed by or through the use of the SOFTWARE. "Developed Web Server Software" means those Developed Software products that reside logically or physically on at least one Web Server and are operated (meaning the computer software instruction set is carried out) by the Web Server's central processing unit(s) (CPU).

"Developed Legacy Software" means those Developed Software products that are not Developed Web Server Software, including, for example, stand-alone applications and applications accessed by a file server only. "Redistributable Files" means the SOFTWARE files or other portions of the SOFTWARE that are provided by MOROVIA and are identified as such in the Documentation for distribution by you with the Developed Software. "Developer" means a human being or any other automated device using the SOFTWARE in accordance with the terms and conditions of this EULA.

"Developer Seat License" means that each Developer using or otherwise accessing the programmatic interface or the SOFTWARE must obtain the right to do so by purchasing a separate End User License.

"Network Server CPU License" means that a separate End User License must be purchased for each CPU operating the computer software at issue in the reference.

"Source Code" shall mean computer software code or programs in human readable format, such as a printed listing of such a program written in a high-level computer language. The term "Source Code" includes, but is not limited to, documents and materials in support of the development effort of the SOFTWARE, such as flow charts, pseudo code and program notes.

2. Your Single User License. You are hereby granted a limited royalty-free, non-exclusive right to use the SOFTWARE on ONE CPU by ONE INDIVIDUAL. The purpose of using SOFTWARE is to produce the results, in digital or printable form, and not to develop custom application. A Single User License does not apply to the Network Server.
3. Your Network Server License. You are hereby granted a limited royalty-free, non-exclusive right to use the SOFT on ONE CPU on a Network Server. The number of Network Server allow 1 CPU and 200 concurrent client accesses. Additional licenses need to be purchased if the usage exceed 1 CPU or 200 concurrent clients. If the Software is not used on a Network Server, the license option for Single User applies.
4. Your Developer License. You are hereby granted a limited, royalty-free, right to use the SOFTWARE to design, develop, and test Developed Software, on the express condition that, and only for so long as, you fully comply with all terms and conditions of this EULA. You are also allowed to distribute the software inside or outside your organization for up to 10,000 copies. When you distribute the software, you adhere to the following terms: (a) You may not resell, rent, lease or distribute the Software alone. The Software must be distributed as a component of an application and bundled with an application or with the application's installation files. The Software may only be used as part of, and in connection with, the bundled application. (b) You may not resell, rent, lease or distribute Software in any way that would compete with Morovia Corporation. (c) you must include the following MOROVIA copyright notice in your Developed Software documentation and/or in the "About Box" of your Developed Software, and wherever the copyright/rights notice is located in the Developed Software ("Portions Copyright (c) Morovia Corporation 2004. All Rights Reserved."). (d) agree to indemnify, hold harmless, and defend MOROVIA, its suppliers and resellers, from and against any claims or lawsuits, including attorney's fees that may arise from the use or distribution of your Developed Software. (e) you may use the SOFTWARE only to create Developed Software that is significantly different than the SOFTWARE.
5. Your 5-Developer License. You are hereby granted the rights of the Developer License for up to 5 developers and 20,000 distribution.
6. Your Unlimited Developer License. With license you can install the component software for an unlimited number of developers within your organization and allow unlimited distribution.
7. Serial Number. Within the packaging of the SOFTWARE, a unique serial number (the "Serial Number") is included, which allows for the registration of the SOFTWARE. The Serial Number is subject to the restrictions set forth in this EULA and may not be disclosed or distributed either with your Developed Software or in any other way. The disclosure or distribution of the Serial Number shall constitute a breach of this EULA, the effect of which shall be the automatic termination and revocation of all the rights granted herein.
8. Evaluation Copy. If you are using an "evaluation copy" or similar version, specifically designated as such by MOROVIA on its website or otherwise, then the Licenses are limited as follows: a) you are granted a license to use the SOFTWARE for a period of thirty (30) days counted from the day of installation (the "Evaluation Period"); b) upon completion of the Evaluation Period, you shall either i) delete the SOFTWARE from the computer containing the installation, or you may ii) contact MOROVIA or one of its authorized dealers to purchase a license of the SOFTWARE, which is subject to the terms and limitations contained herein; and c) any Developed Software may not be distributed or used for any commercial purpose.

INTELLECTUAL PROPERTY

1. Copyright. You agree that all right, title, and interest in and to the SOFTWARE (including, but not limited to, any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE), and any copies of the SOFTWARE, and any copyrights and other intellectual properties therein or related thereto are owned exclusively by MOROVIA, except to the limited extent that MOROVIA may be the rightful license holder of certain third-party technologies incorporated into the SOFTWARE. The SOFTWARE is protected by copyright laws and international treaty provisions. The SOFTWARE is licensed to you, not sold to you. MOROVIA reserves all rights not otherwise expressly and specifically granted to you in this EULA.
2. Backups. You may either: (a) copy the SOFTWARE solely for backup or archival purposes; or (b) install the SOFTWARE on a single computer, provided you keep the original solely for backup or archival purposes. Notwithstanding the foregoing, you may not copy the Documentation.
3. General Limitations. You may not reverse engineer, decompile, or disassemble the SOFTWARE, except and only to the extent that applicable law expressly permits such activity notwithstanding this limitation.
4. Software Transfers. You may not rent or lease the SOFTWARE. You may transfer the SOFTWARE to another computer, provided that it is completely removed from the computer from which it was transferred. You may permanently transfer all of your rights under the EULA, provided that you retain no copies, that you transfer all the SOFTWARE (including all component parts, the media and printed materials, any dates, upgrades, this EULA and, if applicable, the Certificate of Authenticity), and that the recipient agrees to the terms and conditions of this EULA as provided herein. If the SOFTWARE is an update or upgrade, any transfer must include all prior versions of the SOFTWARE.
5. Termination. Without prejudice to any other rights it may have, MOROVIA may terminate this EULA and the Licenses if you fail to comply with the terms and conditions contained herein. In such an event, you must destroy all copies of the SOFTWARE and all of its component parts.

WARRANTIES AND REMEDIES

The Software components provided by MOROVIA are licensed to you as is and without warranties as to performance of merchantability, fitness for a particular purpose or any other warranties whether expressed or implied. You, your organization and all users of the font, assume all risks when using it. Morovia shall not be liable for any consequential, incidental, or special damages arising out of the use of or inability to use the font or the provision of or failure to provide support services, even if we have been advised of the possibility of such damages. In any case, the entire liability under any provision of this agreement shall be limited to the greater of the amount actually paid by you or US \$5.00.

MISCELLANEOUS

1. This is the Entire Agreement. This EULA (including any addendum or amendment to this EULA included with the SOFTWARE) is the final, complete and exclusive statement of the entire agreement between you and MOROVIA relating to the SOFTWARE. This EULA supersedes any prior and contemporaneous proposals, purchase orders, advertisements, and all other communications in relation to the subject matter of this EULA, whether oral or written. No terms or conditions, other than those contained in this EULA, and no other understanding or agreement which in any way modifies these terms and conditions, shall be binding upon the parties unless entered into in writing executed between the parties, or by other non-oral manner of agreement whereby the parties objectively and definitively act in a manner to be bound (such as by continuing with an installation of the SOFTWARE, "clicking-through" a questionnaire, etc.) Employees, agents and other representatives of MOROVIA are not permitted to orally modify this EULA.

2. You Indemnify MOROVIA. You agree to indemnify, hold harmless, and defend MOROVIA and its suppliers and resellers from and against any and all claims or lawsuits, including attorney's fees, that arise or result from this EULA.
3. Interpretation of this EULA. If for any reason a court of competent jurisdiction finds any provision of this EULA, or any portion thereof, to be unenforceable, that provision of this EULA will be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this EULA will continue in full force and effect. Formatives of defined terms shall have the same meaning of the defined term. Failure by either party to enforce any provision of this EULA will not be deemed a waiver of future enforcement of that or any other provision.

Glossary

AIM	Abbreviation for AIM International, a world-wide trade organization for manufacturers and providers of bar code products, services and supplies.
ASCII	The character set and code described in American National Standard Code for Information Interchange, ANSI X3.4-1977. Each ASCII character is encoded with seven bits.
Aspect ratio	The ratio of bar height to the overall length of the barcode.
BMP	BMP is a raster graphics format developed by Microsoft. BMP is the native graphics format for Windows users. A BMP image data can be uncompressed, or compressed using RLE scheme. The file size is generally much bigger than other types since the compression scheme is not very effective.
Check character	Synonymous to "Check digit".
Check digit	A character whose value is calculated based on certain algorithm and used for the purpose of performing a mathematical check to ensure the accuracy of the data. In many symbologies this character has a numeric value hence the name.
Code 39	Code 39 (also known as USS Code 39, Code 3 of 9) is the first alpha-numeric symbology developed to be used in non-retail environment. It is widely used to code alphanumeric information, such as the model number etc. It is designed to encode 26 upper case letters, 10 digits and 7 special characters.
Code 93	Code 93 is a discrete, variable length, self-checking symbology. It is derived from Code 39 with major enhancements. Code93 encodes all 127 ASCII characters and does not require special scanner configuration.
Data matrix	Data matrix is a space-efficient two-dimensional bar code symbology that is made up of square modules. A data matrix symbol is capable of encoding up to 2335 alphanumeric characters, or 1556 characters of 8-bit byte data, or 3116 digits of numeric data.
EAN-13	EAN is designed by the International Article Numbering Association (EAN) in Europe. It is an extension to UPC-A to include the country information. EAN-13 encodes 12 digits of numeric data along with a trailing check digit, for a total of 13 digits of barcode data.
EAN-8	EAN-8 is the short version of EAN-13, the same as UPC-E vs. UPC-A. An EAN-8 number contains 7 digits of message plus 1 check digit. Different from UPC-E, an EAN-8 number is allocated separately and can not be derived from an EAN-13 number.
EMF	Acronym for Enhanced MetaFile. A newer 32-bit version of Windows MetaFile. EMF contains frame information and contain more drawing commands than its predecessor, WMF.

Extended character	A character other than a 7-bit ASCII character. An extended character is a 1-byte code point with the eighth bit set (ordinal 128 through 255).
GIF	Acronym for Graphics Interchange Format. GIF is a bitmap image format encoding up to 256 distinct color in a 24-bit RGB color space. GIF employs LZW data compression, which does not lose image data during the compression process.
GS1	Organization that oversees the allocation of U.P.C. and EAN numbers. Formerly known as Uniform Code Council (UCC).
GS1 DataBar	A family of bar code symbols, including GS1 DataBar-14, GS1 DataBar Limited, GS1 DataBar Expanded, and GS1 DataBar-14 Stacked. Any member of the GS1 DataBar family can be printed as a stand-alone linear symbol or as a composite symbol with an accompanying 2D Composite Component printed directly above the GS1 DataBar linear component. Formerly known as Reduced Space Symbology (RSS).
GS1 DataBar Expanded	A bar code symbol that encodes an GTIN-14 Identification Number plus supplementary AI Element Strings, such as weight and “best before” date, in a linear symbol that can be scanned omnidirectionally by suitably programmed Point-of-Sale scanners.
GS1 DataBar Expanded Stacked	A bar code symbol that is a variation of the GS1 DataBar Expanded Bar Code Symbol that is stacked in multiple rows and is used when the normal symbol would be too wide for the application.
GS1 DataBar Limited	A bar code symbol that encodes an GTIN-14 Identification Number with Indicators of zero or one in a linear symbol; for use on small items that will not be scanned at the Point-of-Sale.
GS1 DataBar Truncated	A truncated version of GS1 DataBar-14. The height is 10X. Used for small packaging and not fit for Point-of-Sale scanners.
GS1 DataBar Stacked	A bar code symbol that is a variation of the GS1 DataBar-14 Symbology that is stacked in two rows and is used when the normal symbol would be too wide for the application. It comes in two versions: a truncated version used for small item marking applications and a taller omni-directional version that is designed to be read by omni-directional scanners. GS1 DataBar Expanded can also be printed in multiple rows as a stacked symbol.
GTIN	Acronym for Global Trade Item Number. A 14-digit number that uniquely identifies a trade item.
HIBC	Acronym for Health Industry Bar Code. A bar code format based on code 3 of 9 adopted by health industry.
JPEG	JPEG stands for Joint Photographic Experts Group. It is commonly referred as an image format.
Macro PDF417	A method to link multiple PDF417 symbols together in order to encode large amount of data.

MaxiCode	MaxiCode is a two-dimensional code, created by UPS for high-speed sortation and tracking of unit loads and transport packages. It is ideal to encode small amount of data since its capacity is fairly limited. On the other side, its fixed size and unique "bull eye" design allow the symbol being picked up very quickly.
Module	In linear symbology, a module refers to the width of the narrowest bars. In two dimensional symbology, a module refers to the cell smallest in size.
PDF417	PDF417 is a multi-row, variable-length symbology with high data capacity and error-correction capability. PDF417 has some unique features which makes it the widely used 2D symbology. A PDF417 symbol can be read by linear scanners, laser scanners or two-dimensional scanners. PDF417 is capable of encoding more than 1100 bytes, 1800 text characters or 2710 digits. Large data files can be encoded into a series of linked PDF417 symbols using a standard methodology referred to as Macro PDF417.
PNG	Acronym for Portable Network Graphics. PNG is a bitmap image format that employs lossless data compression.
POSTNET	POSTNET (Postal Numeric Encoding Technique) encodes a US numeric address code (also called Zip code) which may contain 5, 9 or 11 digits - often called <i>Zip</i> , <i>Zip+4</i> and <i>Zip+6</i> .
Quiet zones	A clear space, containing no machine readable marks, which surrounds the barcode. Sometimes called the "clear area".
Start/Stop character	A special bar/space pattern that provides the scanner with start and stop reading instructions as well as scanning direction indicator. Most linear symbologies require start/stop characters included in the barcode.
TIF	Acronym for Tagged Image File Format. Also abbreviated as TIFF. TIF is a bitmap image format capable of storing multiple images. It is widely used in scanning, faxing and word processing.
UPC-A	The UPC-A barcode is the most common and well-known symbology in North America. UPC-A encodes 11 digits of numeric data along with a trailing check digit, for a total of 12 digits of barcode data.
UPC-E	The UPC-E barcode is the short form representation of a UPC-A number. It reduces the data length from 12 digits to 6 digits by compressing extra zeros.
UPS	Abbreviation for United Parcel Service, the largest carrier company in the US.
USPS	Abbreviation for U.S. Postal Service.
WMF	Acronym for Windows Metafile. WMF is a graphics file format on Microsoft Windows. WMF is a vector graphics format which stores drawing commands instead of color information of pixels.
X dimension	The nominal width dimension of the narrowest element in the bar code - bar or space.

XML

Acronym for eXtensible Markup Language. XML refers to a set of open standards describing data ranging from representation (such as web pages) to business structure. Unlike HTML, XML does not have a set of predefined elements. Instead it provides a common method for describe a document type and the data.

Index

A

About, 77
AutoSize, 26

B

BackColor, 27
BarHeight, 28
BearerBars, 29
BorderColor, 30
BorderStyle, 31
BorderWidth, 32

C

Code25OptionalCheckDigit, 33
Code39OptionalCheckDigit, 34, 88, 89
Code39StartStopChars, 35, 88, 88
Comment, 36
CommentAlignment, 37
CommentFont, 38
CommentMarginBottom, 39
CommentMarginLeft, 39
CommentMarginRight, 39
CommentMarginTop, 39
CommentOnTop, 40
CreateBarcodeObject, 17, 17

D

DataMatrixModuleSize, 41
DataMatrixTargetSizeID, 42, 112
DestoryBarcodeObject, 18
DestroyBarcodeObject, 17

E

ExportImage, 18, 78
ExportImage2, 18
Extended Channel Interpretation, 113, 117

F

File ID, 109, 114
ForeColor, 27

G

Global Label Identification, 108

I

I2of5OptionalCheckDigit, 45
IsBarcodeObjectDemo, 17
ITF-14 (see Interleaved 2 of 5)
ITF25 (see Interleaved 2 of 5)

L

LabelHeight, 46
LabelWidth, 46
Load, 80

M

Macro 5 and 6, 113
Macro PDF417, 109
MaxicodeClass, 47
MaxicodeCountryCode, 49
MaxicodeMode, 48
MaxicodeZipCode, 50
mbxExportImage, 17
mbxLoad, 18
mbxSave, 18
Measurement, 51
Message, 52

N

NarrowBarWidth, 53
NarrowToWideRatio, 54

P

PDF417TruncatedSymbol, 108
PDFAspectRatio, 55
PDFMaxCols, 56
PDFMaxRows, 57
PDFModuleHeight, 58
PDFModuleWidth, 59
PDFSecurityLevel, 60
PDFTruncatedSymbol, 61
Picture, 62

Q

QuietZones, 63

R

RasterImageResolution, 64

S

Save, 81

Segment Index, 109
 Sequence Indicator, 114
 ShowCheckDigit, 66
 ShowComment, 67
 ShowHRText, 66, 68
 Special Character Input Method
 Code 39 Full ASCII, 89
 Code 93, 90
 Structural Append, 114, 117
 SymbolMarginBottom, 71
 SymbolMarginLeft, 71
 SymbolMarginRight, 71
 SymbolMarginTop, 71
 Symbologies
 Bookland, 93
 Codabar, 90
 Code11, 90
 Code 128, 94
 Code 25, 90
 Code 39, 88
 Code 39 Full ASCII, 89
 Code 39 HIBC, 89
 Code 93, 90
 DataBar, 102
 DataBar Expanded, 102
 DataBar Expanded Stacked, 102
 DataBar Limited, 102
 DataBar Stacked, 102
 DataBar Stacked Omnidirectional, 102
 DataBar Truncated, 102
 EAN-13, 92
 EAN-8, 92
 EAN Supplements, 92
 Interleaved 2 of 5, 105
 MaxiCode, 115
 MSI/Plessey, 90
 PDF 417, 107
 POSTNET, 106
 UCC/EAN-128, 96
 UPC-A, 91
 UPC-E, 91
 UPC Supplements, 91
 Symbology, 69

T

Technical Support, 119
 TexAlignment, 72
 TextOnTop, 74
 Truncated PDF, 108

U

UccEanOptionalCheckDigit, 75

Z

Zip (see POSTNET)
 ZoomRatio, 76