



Datenverarbeitung GmbH

Wagnerstr. 6  
A-4400 Steyr  
AUSTRIA/EUROPE

# TBARCODE 5.0 ACTIVE X

## Documentation

Last Update: 06-Aug-2004

e: [office@tec-it.com](mailto:office@tec-it.com)

w: [www.tec-it.com](http://www.tec-it.com)

p: +43 / (0) 7252 / 72 720  
f: +43 / (0) 7252 / 72 720 - 77

# Content

<b>1</b>	<b>GENERAL .....</b>	<b>5</b>
<b>1.1</b>	<b>About TBarCode ActiveX .....</b>	<b>5</b>
1.1.1	Why you should use TBarCode ActiveX: .....	5
1.1.2	Difference Registration / Licensing .....	5
1.1.3	OPT – Optimized Pixel Technology .....	5
1.1.4	Limitation of the demo version .....	5
<b>1.2</b>	<b>Download and Setup .....</b>	<b>6</b>
<b>1.3</b>	<b>Support .....</b>	<b>6</b>
<b>2</b>	<b>LICENSING .....</b>	<b>7</b>
<b>2.1</b>	<b>License procedure .....</b>	<b>7</b>
2.1.1	Manual input .....	7
2.1.2	Licensing via program code (Developer) .....	7
2.1.3	Samples .....	7
<b>3</b>	<b>PROPERTY PAGES ACTIVEX .....</b>	<b>8</b>
3.1.1	Invoking .....	8
3.1.2	General .....	8
3.1.3	Advanced .....	9
3.1.4	PDF417 .....	11
3.1.5	MaxiCode .....	13
3.1.6	Data Matrix .....	14
3.1.7	QR Code .....	15
3.1.8	Codablock F .....	17
3.1.9	Font .....	18
3.1.10	Color .....	19
<b>4</b>	<b>PROGRAM INTERFACE (API) ACTIVEX .....</b>	<b>20</b>
<b>4.1</b>	<b>General .....</b>	<b>20</b>
4.1.1	Prog ID, Class ID .....	20
<b>4.2</b>	<b>Properties .....</b>	<b>20</b>
4.2.1	General (for 1D and 2D types of bar codes) .....	20
4.2.2	PDF417 Properties .....	24
4.2.3	MaxiCode Properties .....	25
4.2.4	Data Matrix Properties .....	26
4.2.5	QR Code Properties .....	26
4.2.6	Codablock F Properties .....	27
4.2.7	Barcode Properties within Event-Handlers .....	28
<b>4.3</b>	<b>Methods .....</b>	<b>28</b>
4.3.1	Methods for Standard Applications .....	28

4.3.2	Methods for Web Applications .....	30
4.3.3	Error Handling.....	31
<b>4.4</b>	<b>Events .....</b>	<b>31</b>
<b>5</b>	<b>PROGRAM INTERFACE (API) DLL.....</b>	<b>32</b>
<b>5.1</b>	<b>General.....</b>	<b>32</b>
5.1.1	Basic Sequence.....	32
<b>5.2</b>	<b>Function Reference .....</b>	<b>33</b>
5.2.1	Init- / Deinit Functions .....	33
5.2.2	Set / Get Functions (Properties) .....	33
	RSS Properties.....	38
5.2.3	Set / Get Functions for PDF417.....	38
5.2.4	Set / Get Functions for MaxiCode.....	39
5.2.5	Set / Get Functions for Data Matrix .....	40
5.2.6	Set / Get Functions for QR Code.....	41
5.2.7	Set / Get Functions for Codablock F.....	43
5.2.8	Methods (Drawing, Licensing.....)	44
5.2.9	Callback functions for user-defined drawing routines.....	47
<b>6</b>	<b>APPENDIX .....</b>	<b>49</b>
<b>6.1</b>	<b>Bar Code Reference .....</b>	<b>49</b>
6.1.1	Enumeration and Default Settings .....	49
6.1.2	Related Bar Code Symbolologies.....	53
<b>6.2</b>	<b>Parameter .....</b>	<b>53</b>
6.2.1	Check Digit Enumeration .....	53
6.2.2	Ratio, RatioHint (Ratio Format) .....	54
6.2.3	Format .....	55
6.2.4	ESC Sequences and Control Characters .....	56
6.2.5	Application Identifier .....	58
<b>6.3</b>	<b>MaxiCode.....</b>	<b>59</b>
6.3.1	Extended Documentation .....	59
6.3.2	Setting SCM parameters .....	59
<b>6.4</b>	<b>Japanese Postal (Customer) Code .....</b>	<b>59</b>
	A) Direct Encoding Mode .....	59
	B) Japanese Extraction Mode .....	59
<b>6.5</b>	<b>Korean Postal Authority Code.....</b>	<b>60</b>
<b>6.6</b>	<b>Error Codes .....</b>	<b>61</b>
<b>6.7</b>	<b>Image Types .....</b>	<b>61</b>
6.7.1	Image Data Format.....	61
6.7.2	Compression Modes.....	62
6.7.3	Resolution and Readability .....	62
<b>7</b>	<b>FAQ'S.....</b>	<b>65</b>

7.1.1	How to add the leading and trailing '*' for Code 39? .....	65
7.1.2	How to add the check digit to Code 39? .....	65
7.1.3	How can I add bar codes to a mail merge document? .....	65
7.1.4	How can I determine the number of PDF417 Code Words? .....	66
7.1.5	How to add the leading and trailing 'A' (or B or C or D) for CODABAR?.....	66
7.1.6	How can I change the module width to 15 mils in my web application (ASP) ? .....	66
7.1.7	How to use a specific subset in Code 128? .....	67
7.1.8	How to use the compressed mode of Code 128? .....	67
7.1.9	How can I get a PDF417 symbol with standard aspect ratio (3:2) ? .....	68
7.1.10	How to license the product from within my application? .....	68
7.1.11	How to use Application Identifiers (AI's) in Code 128 or EAN128? .....	68
7.1.12	How can I set the Module Width to a constant value? .....	69
7.1.13	I am changing the font resolution (large fonts, 120dpi) and get a clipped off bar code with SaveImage? .....	70
7.1.14	I am using PrintForm in VB and the barcode is not readable .....	70
7.1.15	What is the maximum data capacity of PDF417? .....	71
7.1.16	When using ESC-Sequences they are not encoded (and the scanners signals an error) - why? .....	71
7.1.17	How can I save the MaxiCode symbol with a higher image resolution? .....	72
7.1.18	How to draw bar code to image or to printer device context .....	72
7.1.19	I always get an error when calling SaveImage method? .....	73
7.1.20	When using SaveImage, my barcode reader can't scan the symbol / image!.....	73
7.1.21	How can I use TBarCode within FoxPro? .....	74
7.1.22	How to speed up the generation and printing of QR-CODE for large numbers? .....	75
7.1.23	What can I do to optimize PDF417 for sending through a desktop analogue FAX? .....	75

# 1 General

## 1.1 About TBarCode ActiveX

Thank you for choosing our product. TBarCode ActiveX is a tool for generating and printing bar codes directly from within your application.

You can download the latest version from <http://www.tec-it.com/download>

Beside the ActiveX Control we provide also **bar code libraries for programmers**:

- TBarCode DLL 32-Bit DLL for the Win32-API)
- LibTBarCode Linux/Unix library for C/C++ developers

Using TBarCode implies that you are accepting our [license terms](#) - therefore we recommend you to read them at first!

### 1.1.1 Why you should use TBarCode ActiveX:

- You don't need programming knowledge - ActiveX Controls can simply be used with Insert→Object and right-clicking the object to modify the properties.
- You can still program the control if your application needs programming (e.g. within VB, Office...).
- You can create bar codes in the highest possible output resolution and quality (OPT technology).
- You can automate check digit calculation and adjust all bar code specific parameters: module width, bar width ratio, text distance and much more...
- You can use TBarCode ActiveX within web applications running either on the server or at client side. Using TBarCode within Active Server Pages (IIS) and PHP (Windows) is fully supported.
- You have more than 55 different bar code symbologies within one product and one price.
- Our bar code specialists will help you and our tech support is free.

### 1.1.2 Difference Registration / Licensing

The ActiveX Control must be registered in the Windows OS as OLE Control correctly. This registration is executed automatically with the installation program (setup). On demand you can manually register the OCX file with the command line tool "regsvr32.exe".

*Registration is not the same as licensing. The registration process announces ActiveX Controls to the Windows Environment so that other programs can use the Control. But you can use (evaluate) the product without licensing (although in the demo mode with restrictions).*

### 1.1.3 OPT – Optimized Pixel Technology

As novelty TEC-IT introduced "OPT" (Optimized Pixel Technology) with its TBarCode products starting from the version 2.0 - a special calculation method of the output graphic, which uses the resolution of the available printer in the best possible way. That is particularly important if high data densities occur (with small bar code dimensions) or the available printing device has only low output resolution.

If you have to cope with low resolution you can use the new Property "OptResolution" – available in Version 5. It optimizes the module width and ensures that the output quality corresponds to the highest possible results of printing and your scanner can read the bar code.

### 1.1.4 Limitation of the demo version

In the demo version the bar code will be drawn with an additional horizontal bar. In order to enable the full-featured version (without horizontal bars), you need a license key from TEC-IT.

After you have got your license key you can switch the downloaded eval version to a licensed version without limitations.

## 1.2 Download and Setup

First you download the product from our homepage: <http://www.tec-it.com/Download> (Software - > TBarCode ActiveX).

After the download you execute the self-extracting installer to start the setup of the product. Please acknowledge the license conditions! The program files will be installed into the directory: C:\Program Files\TEC IT\TBarCode5 per default, if you don't indicate your own path.

After the installation process the TBarCode ActiveX object can be inserted into the desired application.

## 1.3 Support

If you have troubles → our free support is available via

- Email: <mailto:support@tec-it.com>
- Online Form: <http://www.tec-it.com/support/>
- FAX: [+43] (0)7252 / 72720-77.

## 2 Licensing

Our [License Terms](#) are included at the end of this documentation. The latest **pricing** and the ability of **online ordering** are available at our Website: <http://www.tec-it.com/order/>.

A description of the available **types of license** as well as all necessary information for **ordering** can be found in the **FAQ** area of our web page: <http://www.tec-it.com/FAQ>.

If you don't know the license type according to your application, please ask for our support: <mailto:support@tec-it.com>.

### 2.1 License procedure

Licensing must take place at least once for each system (or installation). It is however no problem to license the product several times - e.g. with each program start in the own code. This way of licensing is recommended.

#### 2.1.1 Manual input

The license data can be entered in the license dialog. Please input the license data exactly as received into the license dialog of the TBarCode ActiveX Control. Blanks and upper/lower case are considered. To avoid errors you can use "cut & paste" (if license data is present as email). This process must take place on each target computer.

The license dialog of the product can be opened as follows:

- right-click on the inserted TBarCode object
- select the menu entry [TBarCode5-Object] - [License].

#### 2.1.2 Licensing via program code (Developer)

The recommended method for developers: Call the method (or function) LicenseMe () from within your application, before bar codes are printed. When calling this function the ActiveX already should be present as an instance, otherwise an error message will be received.

A sample for licensing the ActiveX Control in Visual Basic (or VBA or VBScript) can be found in the [FAQ](#) section!

#### 2.1.3 Samples

For easiest beginning and use: For all common software products we support you with sample applications including the source code.

Download them from the TEC-IT homepage <http://www.tec-it.com/Download>

Note: Precondition for the samples is the installed ActiveX Control!

## 3 Property Pages ActiveX

### 3.1.1 Invoking

In most Windows applications the content and the appearance of the ActiveX Control can be changed with a right mouse-click on it. After right clicking, the appearing menu *[TBarCode5-Object] - [Properties]* offers access to the characteristics of the object by its own dialog window (so called "Property Pages").

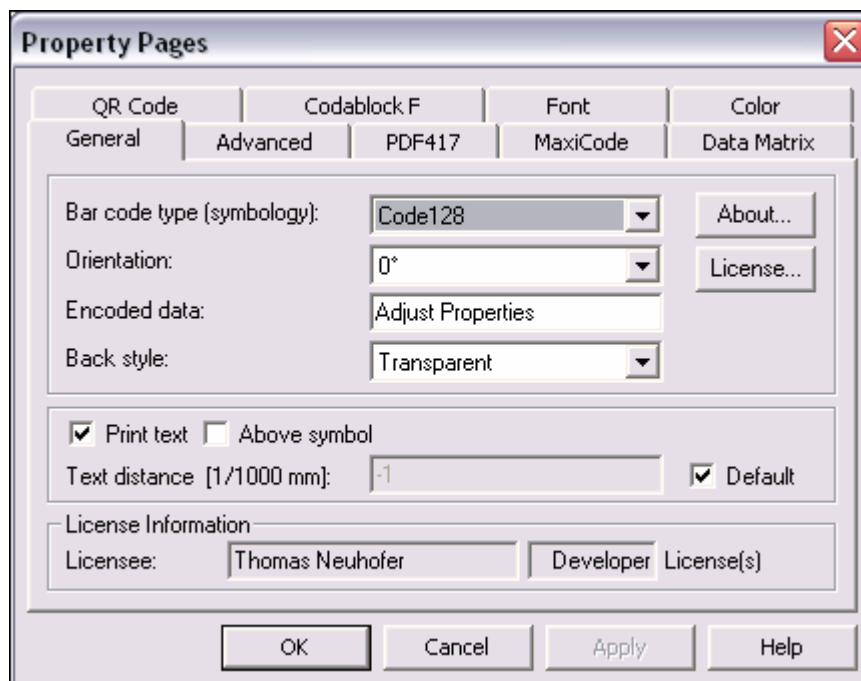
The Property Pages are described in the following sections.

In Microsoft Office you can use the menu option "Properties" too - in this window you can change all the characteristics of the control in a list of "property: value" pairs. Special properties - in particular "data binding" or "OptResolution" - can be set up only in this dialog (and not in the "Property Pages" of TBarCode itself).

### 3.1.2 General

This page contains the adjustments that are absolutely needed for the basic functionality. All other parameters are preset to usual values. For most applications you only have to change settings within this page.

It contains the following general properties:



Property Page "General":

#### Bar code type (Symbology)

Indicates the bar code type you want to generate. The chosen symbology depends on your needs and influences the type of data that can be encoded and how the printed result looks like. Not all bar code types are able to represent alphanumeric data; some can only represent digits (or additionally represent limited special characters like "/" or "\*\*").

Select the type of bar code according to your application: common types are the UPC (USA), EAN (Europe), Code 128, Code 39, 2 of 5 Interleaved and PDF417.

For the list of implemented bar code symbologies check out the [Bar Code Reference](#).

#### Orientation

Defines the orientation of the bar code. If another value is selected (90°, 180°, 270°), the bar code is rotated clockwise.

Note: Some character fonts do not support rotation (e. g. some bitmap fonts – choose a True type Font instead).



## Encoded Data

Enter the data, which should be represented as bar code (letters, digits and special characters). The graphic result depends on the selected bar code type. If the encoded data is invalid for the selected symbology, an "X" is displayed together with an error code and a short description instead of the bar code.

## Back style

Indicates the mode used for drawing the background. The bar code can be painted in a transparent (- > standard, background shines through) or opaque way (background is overwritten with the background color).

## Print text

Indicates if the content of the barcode is printed as "human readable text" in a separate line (below or above). Default: Yes

## Above symbol

Indicates where the "human readable text" should be printed. Selectable values: below or above the bar code (- > standard: below). For some bar codes, like UPC and EAN, the adjustment "Above symbol" is not permitted.

## Text distance

Distance between the "human readable text" and the bar code (in 1/1000 mm). If "default" is marked, a default value is used.

## License Information

Contains information about the licensee and the license type entered in the license dialog.

## Button "About..."

Opens the „About Dialog“ with version and copyright info.

## Button „License..."

Opens the License-Dialog. For more information refer to chapter [License procedure](#).

### 3.1.3 Advanced

The following advanced bar code properties can be set:

The screenshot shows the 'Property Pages' dialog box with the 'Advanced' tab selected for the 'PDF417' barcode type. The dialog has a tabbed interface with 'QR Code', 'Codablock F', 'Font', 'Color', 'General', 'Advanced', 'PDF417', 'MaxiCode', and 'Data Matrix'. The 'Advanced' tab contains the following settings:

Property	Value	Default
Module width [1/1000 mm]:		<input checked="" type="checkbox"/> Default
Print ratio:	1:2:3:4:1:2:3:4	<input checked="" type="checkbox"/> Default
Format/subset:		<input checked="" type="checkbox"/> Default
Guard width [1/1000 mm]:	0	<input checked="" type="checkbox"/> Default
Notch height [1/1000 mm]:	-1	<input checked="" type="checkbox"/> Default
Check digit:	Default	
Translate escape sequences:	<input checked="" type="checkbox"/>	
Symbol must fit into rectangle:	<input type="checkbox"/>	
Suppress error message:	<input type="checkbox"/>	

Buttons at the bottom: OK, Cancel, Apply, Help.

Property Page "Advanced"

Please take care when modifying these properties; some values may result in unreadable bar codes. Always make a test scan in case of doubt!

### Module width [1/1000 mm]

Indicates the width of a module in 1/1000 mm – this parameter is also known as “Narrow Bar Width”.

Each bar code element is divided into individual "modules". One module is the smallest unit of the graphical bar or space segments. The module width functions as fundamental unit, i.e. all lines and spaces are based on this adjustment.

If "default" is marked, the module width is calculated automatically based on the width of the bounding rectangle (set by changing the size of the ActiveX object) and the number of utilizable data characters to be encoded.

By adjusting the module width to a constant value the bar code becomes wider if more data is encoded. The bounding rectangle must be drawn or programmed at least as wide, as necessary for the full and not truncated representation of the largest amount of utilizable data.

A constant module width is recommended if you have varying quantities of data but the optical data density should be constant. Also some label specifications require a constant module width.

When using the default setting: If the module width is computed automatically, the optical data density of the print out increases with higher amounts of encoded data characters. Depending upon the printer resolution, the lower limit for the module width may be exceeded – leading to unreadable bar codes. Ensure to make the bar code bounding rectangle as wide as necessary for the largest data content.

Please contact our support in case of questions about the module width.

### Print ratio

The *Print Ratio* is the relationship between the bar- and gap-widths of a bar code. Another word for the print ratio is “Bar Width Ratio” or “Bar/Space Width Ratio”.

The print ratio must be entered in a special format. The ratio format (evident in the property *RatioHint*) depends on the selected type of bar code and on how many different bar- and gap-widths are used.

The resulting width of a single bar (or gap) is calculated using the indicated Print Ratio and the (calculated or specified) module width.

This property should be used in special applications only.

Example: If a bar code element has 4 different bar widths and 4 different space widths, then the print ratio looks like this (Code 128): 1:2:3:4:1:2:3:4. In the first part ("1:2:3:4") the width ratio of the bars is set, in the second section the relation of the spaces are set (in our case, they are the same). The smallest bar is "1" wide, the next larger is "2" (thus twice as wide as the smaller bar) and so on. Modifications within this area are meaningful only for special applications, since some bar codes may become unreadable when manipulating the print ratio.

Refer to [Bar Code Reference / Enumerations](#) for the print ratios (and *RatioHint* information) according to each bar code type. Also look at [Ratio, RatioHint \(Ratio Format\)](#) for more information.

### Format/subset

This is an image for formatting the utilizable data of the bar code. The format string operates with substitute symbols to indicate how the data is structured. It is also possible to insert constant characters mixed with the bar code data according to defined rules. Also certain control characters that make it possible to change the Subsets for Code 128, EAN 128 and UCC 128 (or to define the desired start/stop character of CODABAR) are defined.

The detailed description of the format string can be found in the References ([Format](#)).

### Guard width [1/1000 mm]

The Guard bar is a horizontal line, which limits the bar code on the upper and lower side. If the value is not set to zero (standard), the Guard bar is printed. Values larger than zero indicate the width of the Guard bar in 1/1000 mm. For some types (like UPC and EAN) the value must be zero.

### Notch height [1/1000 mm]

Sets the protruding of the synchronization bars from the bar code (e.g. the double lines within EAN on the left, centrically and on the right side)

## Check digit

The calculation method of the check digit can be set here. Whether you need a check digit depends on your application and on the selected bar code symbology. TBarCode calculates check digits automatically.

Why check digits? In order to guarantee that the bar code data is read properly, a check digit is intended at the end of the utilizable data. A comparison of the bar code content with the check digit informs the scanning device about incorrect scans and forces the device to repeat (or reject) the scan. The check digit calculation method is standardized for certain common bar codes.

With the default-adjustment the check digit is calculated according to the specification of the selected bar code type. Not all bar codes have a must check digit, it also can be optional.

Modifications within this area are admissible only for special applications or for bar codes with selectable check digit methods. This field is also used to enable check digit calculation, if none is set as standard (a check digit is not standard, but recommended for LOGMARS or Code39).

### Automatic “check digit override”:

You can do a “check digit override” – which means the check digit is already supplied in the bar code data and therefore no automatic check digit calculation is performed by TBarCode.

Automatic “check digit override” happens if the selected bar code has a *predefined number of utilizable data characters* and has a *preset check digit place* and your data source contains already the whole data including the check digit.

Example: For encoding data within the EAN-13 symbology you have 13 digits from your data source. EAN 13 permits encoding of 12 utilizable digits + 1 check digit at the end → the internal calculated check digit of TBarCode will be replaced by the external supplied check digit in your input data.

Please keep in mind that in this case our tool does not check the correctness of the check digit supplied by your application.

*Check digit “override”* was primarily designed for use with article databases, where the article numbers already include the check digit. This method can also be used to calculate and provide own check digits (usually by program code) or to use all possible digits for utilizable data (in non standard applications).

Under normal circumstances you should not use this feature – we recommend calculation of the check digit(s) by TBarCode.

## Translate escape sequences

Indicates whether Escape Sequences (like \n) are to be translated (Default: no).

The use of escape sequences is useful if you need to encode control characters such as Carriage Return or FNC1 into the bar code. Also together with 2D symbologies if you want to *encode binary data* you should use Escape Sequences.

For a detailed list see [Reference Esc-Sequences](#).

## Symbol must fit into rectangle

If the option “*Symbol must fit into rectangle*” is marked, an error code will be shown in place of the bar code, if the produced bar code does not fit into the bounding rectangle. This occurs if the module width property is preset to a constant value (by the user) and the total width of the bar code exceeds the bounding rectangle.

Truncated bar codes can be avoided this way and incorrect printouts can be detected immediately. This option is recommendable in connection with manual adjustments of the module width.

## Suppress error message

If the bar code data contain invalid values (characters which can not be encoded with the selected bar code type e.g. letters like “A” when using 2OF5 Interleaved), then an error message is displayed in place of the bar code. This error output can be suppressed by checking this box -leading to blank fields when an error has occurred.

### 3.1.4 PDF417

The PDF417 - Page is used to modify the standard behavior of the two-dimensional PDF417 bar code.

PDF417 divides data contents into graphical rows and columns. It is a so-called “stacked symbology” with error correction capabilities. This page is not relevant for other symbologies.

Property Page “PDF417”:

Note: Caution when modifying these adjustments. Always make a test scan in case of doubt!

### Rows [3..90]

Indicates the number of graphical rows used for encoding. The value must be between 3 and 90.

Default: the number of lines (rows) is calculated automatically depending on the encoded data.

### Columns [1..30]

Defines the number of graphical columns in the symbol. The value must be between 1 and 30. Start-, stop- and line-indicator columns are not taken into account.

Default: the number of columns is calculated automatically depending on the number of input characters.

You should not set both rows AND columns to a constant value (fix only one of these two parameters).

### Row height [1/1000 mm]

Sets the height of an individual row in 1/1000 mm. Default: the row height is calculated automatically depending on the bounding rectangle and the number of rows.

Some label specifications require a specific ratio between module width and row height (e.g. for 1:3 set the module width to 254 and the row height to 762).

### Error correction level

Defines the error correction level. The value can be between 0 and 8.

0 ... error recognition only (no correction possible)

8 ... error correction (highest level).

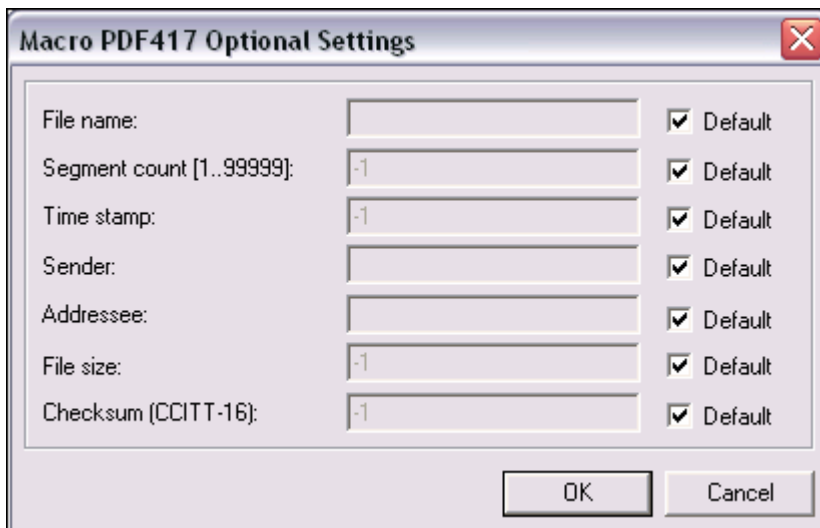
Default: the level is calculated depending on the input data and is set in a range between 2 to 5.

A higher level adds more information to the symbol, which will need more space for printing. If the symbol is distorted through surface damage, bad printing quality or smut the error correction information can help to reconstruct the full information contained in the PDF symbol (reconstruction is done by the scanner).

### Macro PDF417

Macro PDF417 is used for connecting multiple PDF417 symbols into one data chain. You can specify the segment index of the actual symbol, the File-ID that specifies all symbols of one chain and if the current symbol is the last in the chain.

Use the advanced settings button to enter all parameters for Macro PDF:



**Macro PDF417 Optional Settings**

File name:		<input checked="" type="checkbox"/> Default
Segment count [1..99999]:	-1	<input checked="" type="checkbox"/> Default
Time stamp:	-1	<input checked="" type="checkbox"/> Default
Sender:		<input checked="" type="checkbox"/> Default
Addressee:		<input checked="" type="checkbox"/> Default
File size:	-1	<input checked="" type="checkbox"/> Default
Checksum (CCITT-16):	-1	<input checked="" type="checkbox"/> Default

OK Cancel

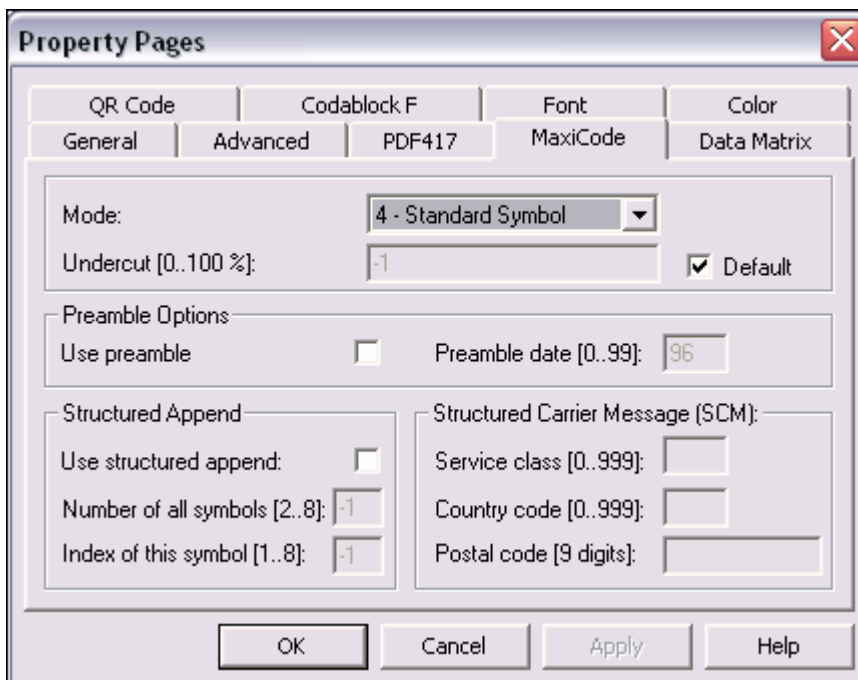
## Error correction

Each PDF417 bar code contains at least 2 code words for error recognition. The error correction level indicates, how many code words are contained in the bar code for error correction. Level 0: 2 code words, level 1 - > 4 code words, level 2 - contains > of 8 code words, etc. (16, 32, 64, 128, 256...) up to level 8 with 512 code words. The used Reed Solomon procedure has the following limit for a successful reconstruction of the data: Formula: ([total of the number of not-decodable characters] + 2 \* [number of read errors]) must be smaller than ([number of error correcting code words] - 2).

### 3.1.5 MaxiCode

MaxiCode represents data by drawing hexagonal items, which are arranged around a circular center. The internal data structure is regulated by different "modes". The mode "Structured Carrier Message" was defined by the parcel transport service UPS®. Data can be encoded with two different error correction levels (SEC = Standard Error Correction and EEC = Enhanced E.C.).

MaxiCode was designed very flexible. With *Structured Append* you can divide larger quantities of data into several MaxiCode symbols – they are joined by the scanner. The maximum data capacity of one symbol is 93 characters. The actual quantity of the utilizable data depends on the selected mode, number of special characters, whether numeric sequences (which can be compressed) are used and error correction level.



**Property Pages**

QR Code	Codablock F	Font	Color
General	Advanced	PDF417	MaxiCode

Mode: 4 - Standard Symbol

Undercut [0..100 %]: -1 ☒ Default

Preamble Options

Use preamble ☐ Preamble date [0..99]: 96

Structured Append

Use structured append: ☐

Number of all symbols [2..8]: -1

Index of this symbol [1..8]: -1

Structured Carrier Message (SCM):

Service class [0..999]:

Country code [0..999]:

Postal code [9 digits]:

OK Cancel Apply Help

**Screenshot Property Page "MaxiCode":**

## Mode

Selects the mode of the actual symbol. Default: Mode 4

Mode 2: SCM Numeric → Structured Carrier Message with 9 digits Postal Code (digits only)

Mode 3: SCM Alphanumeric → Structured Carrier Message with up to 6 characters Postal Code (alphanumeric characters)

Mode 4: No SCM, encoding of numeric and alphanumeric characters (incl. Standard Error Correction).

Mode 5: Full EEC – like mode 4 but with maximum error correction (safer, but fewer data possible).

## Undercut [0..100]

The undercut influences the diameter of the hexagonal bar code elements. In new applications it is recommended (according to the AIM standard) to use an undercut setting of 75% (the default).

## Preamble Options

Used in particular "Open System Standards". Under *Preamble Date* the last two digits of a year can be entered. They are inserted automatically into the data stream in a predefined place.

## Structured Append

If you want to "connect" several MaxiCode symbols in order to encode larger quantities of data, then you can use "Structured Append". A symbol identification number - which is entered in the "*index*" field - can be assigned to each MaxiCode symbol. Its value can range between 1 and 8. This index indicates the order, in which the data is joined after the reading/scanning process. The total number of arranged MaxiCode symbols must be entered in "*Number of all symbols*".

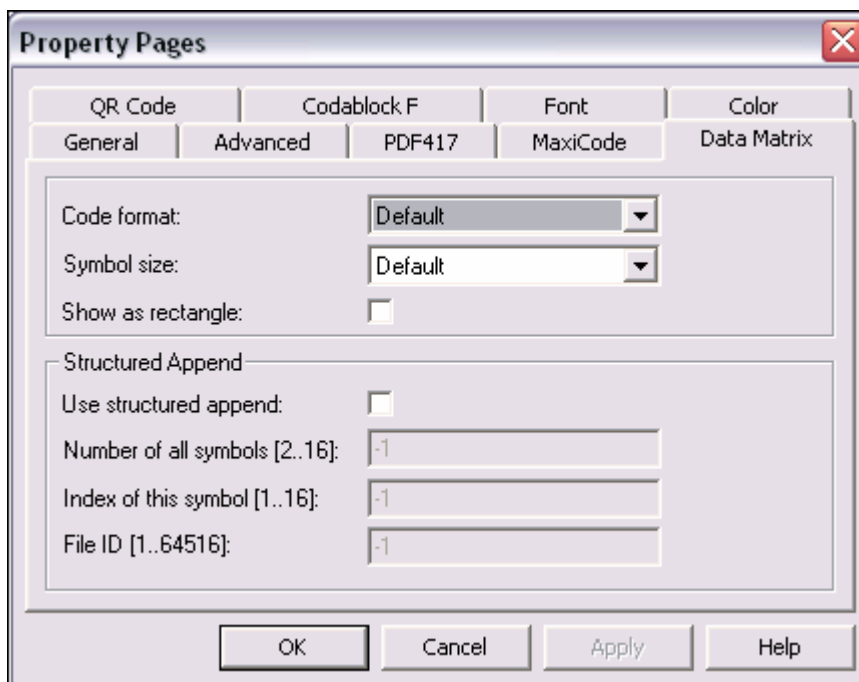
## Structured Carrier Message (SCM).

MaxiCode was originally developed by UPS® (United Parcel Service). In the operating mode "Structured Carrier Message" (mode 2 and 3) there are defined data fields for UPS® - purposes. These can be entered in the fields "*Service Class*", "*Country Code*" and "*Postal Code*". In "Mode 3" you can use digits as well as alphanumeric characters for the "Postal Code".

The SCM contains a header, which includes Date, Preamble, Service class, Country- and Postal Code. These fields can be specified also by Esc-sequences directly in the barcode data (text property). To learn more about this possibility refer to [Setting SCM parameters](#)

### 3.1.6 Data Matrix

In this menu you can set the properties specific to Data Matrix.



Screenshot Property Page  
"Data Matrix":

## Code format

Default: standard format (no special header included)

UCC/EAN: special format defined by UCC and EAN for encoding Application Identifiers. This format adds the function character FNC1 at 1<sup>st</sup> position in the symbol.

Industry: supports peculiar industry formats (adds FNC1 at 2<sup>nd</sup> position).

Macro 05: []>Rs05Gs is encoded at the beginning of the code.

Macro 06: []>Rs06Gs is encoded at the beginning of the code.

TBarCode always encodes data using the newest ECC200 error correction method.

## Symbol size

Defines the size of the symbol in rows and columns. Possible sizes are "10 x 10" to "144 x 144" modules for a square symbol and "8 x 18" to "16 x 48" for a rectangular symbol. When set to default the minimal square size is used (depending on input data).

## Show as rectangle

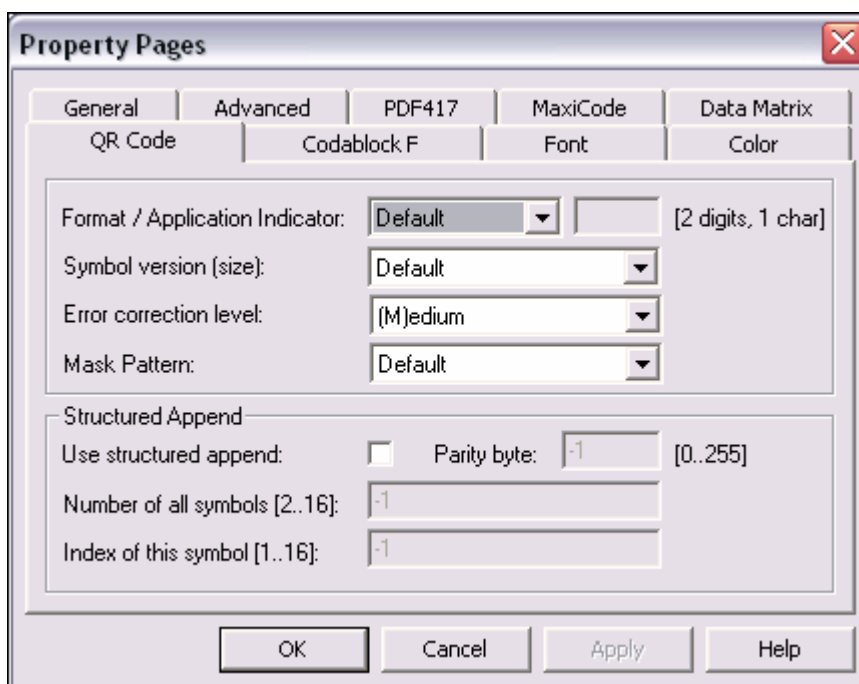
Determines if the Data Matrix Symbol should be displayed as rectangle (checked) or square (unchecked - default).

## Structured Append

If you want to "connect" several Data Matrix symbols in order to encode larger quantities of data, then you can use "Structured Append". A symbol identification number - which is entered in the "index" field - can be assigned to each Data Matrix symbol. Its value can range from 1 to 16. This index indicates the order, in which the data is joined after the reading/scanning process. The total number of arranged Data Matrix symbols must be entered in "Number of all symbols". The File ID has to be the same for all symbols.

### 3.1.7 QR Code

The QR Code symbology is like Data Matrix a 2-dimensional matrix symbology. Remarkable is the large data capacity (up to 3000 ASCII characters or 7000 digits). The QR Code symbology was found to read a lot of data with a bar code scanner within a minimum afford of time (QR Code means **Q**uick **R**eadable code).



Screenshot Property Page  
"QR Code":

## Code format

Default: standard format; additionally you can choose from:

- UCC/EAN: special format defined by UCC and EAN. Used for encoding of so-called Application Identifiers (FNC1 is added at first position).
- Industry: for special industry formats (FNC1 is inserted at second position). If you choose this value, you also have to fill out the field *Application Indicator* (2 digits or 1 letter). It determines to which industry format the input data (should) correspond(s).

## Symbol version

Defines the version (= size) of the QR Code symbol by setting version number and number of rows and columns. Possible values extend from „(1) 21 x 21“ up to „(40) 177 x 177“ modules for a square symbol. If the property is set to default, the size is computed automatically based on the contents.

## Error correction level

Defines the error correction level. You can choose from following possible values:

- (L)ow: Lowest level. Data recovery capacity is approximately up to 7%.
- (M)edium (default): up to 15%
- (Q)uartil: up to 25%
- (H)igh: Highest level. Up to 30%

## Mask Pattern

Defines the mask pattern that is applied to the symbol to improve the readability.

- Default: The mask pattern is calculated automatically.
- 0..7: With the values 0 to 7 you choose the according mask pattern. It does make sense to specify this setting directly, above all then, if you have to generate many symbols with a minimum afford of time because the algorithm to calculate the optimal mask is a highly complex one (according to the factor time).

## Structured Append

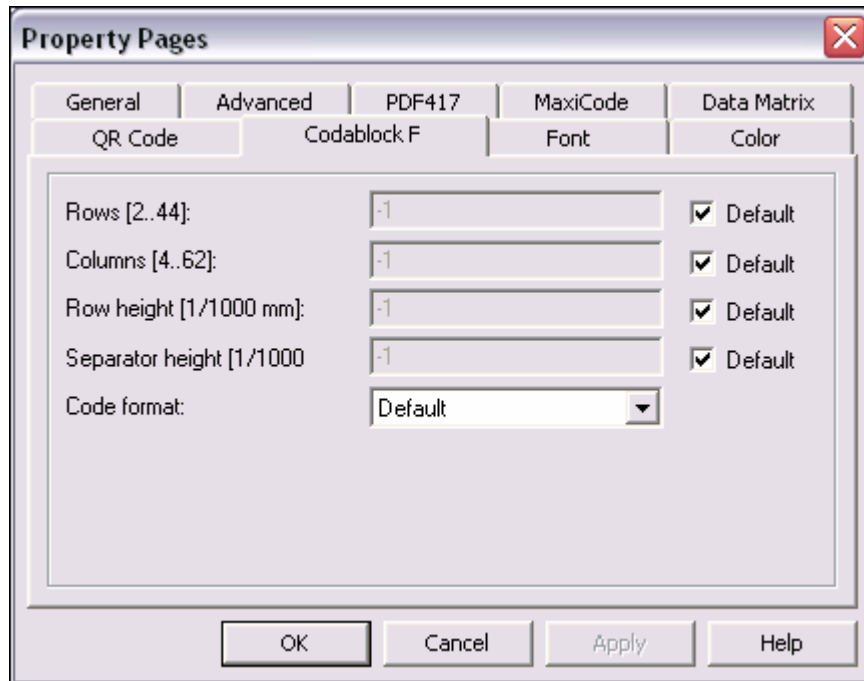
If you want to “connect” several QR Code symbols in order to encode larger quantities of data, then you can use “Structured Append”. A symbol identification number - which is entered in the “*index*” field - can be assigned to each QR Code symbol. Its value can range from 1 to 16. This index indicates the order, in which the data is joined after the reading/scanning process. The total number of arranged Data Matrix symbols must be entered in “*Number of all symbols*”.

Chained QR Code symbols are identified by the *Parity byte*. The *Parity byte* must be identical in all symbols. To calculate its value, use the method „*QR\_StructAppParity*“. You have to pass the whole data string (of all chained symbols) as argument to this function.



### 3.1.8 Codablock F

Codablock F is a stacked symbology (just like PDF417) based upon the Code 128 character set. Each row consists of a Code128 symbol, but extended with row indicators (row count and sequence) and an additional check digit.



Screenshot Property Page  
“Codablock F”:

#### Rows [2..44]

Indicates the number of rows used for encodation. The value must be between 2 and 44. Default: the number of lines is calculated automatically depending on the number of input characters (utilizable data) set in the text property.

#### Columns [4..62]

Defines the number of columns, which are to be printed. The value must be between 4 and 62. Start-, stop-, line-indicator columns, and code subset selectors are not taken into account. Default: the number of columns is calculated automatically depending on the number of input characters.

#### Row height [1/1000 mm]

Sets the height of an individual row in 1/1000 mm. Default: the row height is calculated automatically.

#### Separator height [1/1000 mm]

Sets the height of a row separator in 1/1000 mm. Default: the height of the separator is calculated automatically.

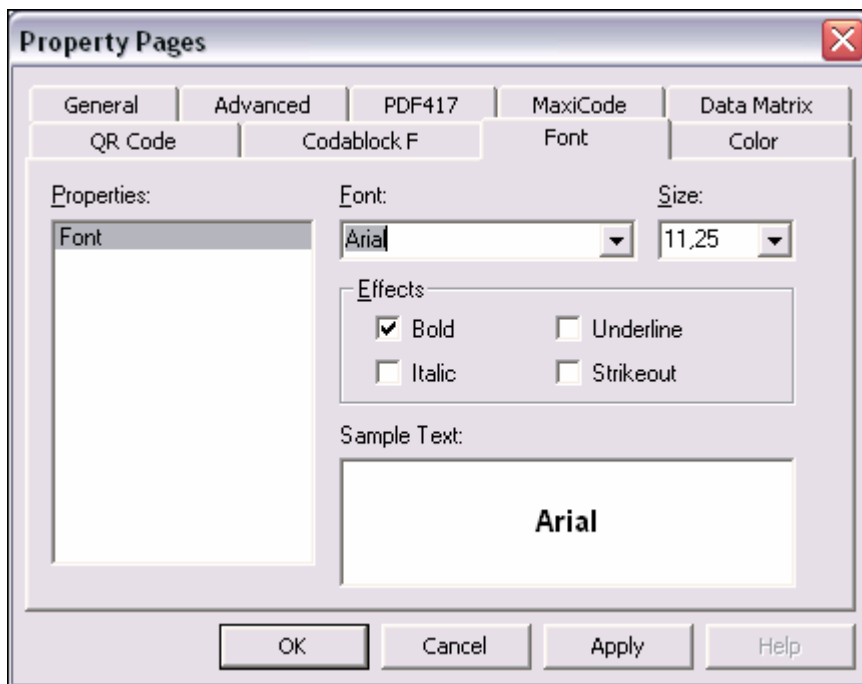
#### Code format

Default: standard format; additionally you can choose from:

- UCC/EAN: special format defined by UCC and EAN. Used for encoding of so-called Application Identifiers (FNC1 is added at first position).

### 3.1.9 Font

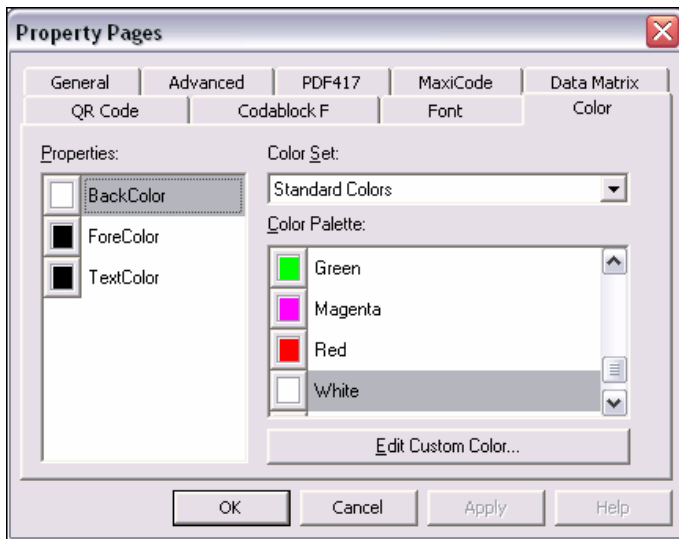
In this menu the font and its attributes used for the “human readable text” can be set. Please consider that some fonts cannot be rotated (TrueType fonts usually feature rotating).



Property Page “Font”:

### 3.1.10 Color

The colors of the bar code, text and the background can be set here. Windows standard colors are available as well as system colors or user-defined colors.



Property Page “Color”:

#### BackColor

Indicates the background color of the bar code (color of the spaces).

#### ForeColor

Indicates the foreground color of the bar code (color of the bars).

#### TextColor

Color of the “human readable text”.

## 4 Program Interface (API) ActiveX

### 4.1 General

In the meantime most programming environments support the use of ActiveX objects. TBarCode ActiveX objects are very comfortable and easy to handle and facilitate programming of bar code applications substantially. The object can be inserted on a form, but also generated as (invisible) instance and used e.g. for printing only. To learn more about object-oriented programming languages and get further information about COM objects we refer to the appropriate technical literature.

#### 4.1.1 Prog ID, Class ID

Prog ID TBarCode5 = „TBarCode5.TBarCode5“

Class ID TBarCode5 = {10ED9AE3-DA1A-461c-826A-CD9C850C58E2}

### 4.2 Properties

The object characteristics (Properties) are discussed in the following reference in alphabetic order. An overview over the enumerators (defined names for bar code symbologies, check digits methods,...) as well as the description of further bar code parameters can be found in the [Bar Code Reference](#). The object properties are essentially identical to those, which are used in the Property Pages – so we recommend the reading of the section [Property Pages](#).

In most cases you only need the properties **Text** and **BarCode**. for standard applications. Visual Basic and other development environments are offering additional "Standard Properties" for ActiveX Controls (or COM objects) that are not described here, e.g. Container, DataField, DragIcon...

#### 4.2.1 General (for 1D and 2D types of bar codes)

Name	Default value	Calling, Description	Get/Set
<b>BackColor</b>	White	Background color of the bar code (used if BackStyle = opaque).	G / S
<b>BackStyle</b>	Transparent (0)	Background representation. Possible values are <i>transparent</i> or <i>opaque</i> (Bar code overwrites background). Enum Type BKStyle or tagBKStyle, possible values: <i>BKS_Transparent</i> (0) und <i>BKS_Opaque</i> (1)	G / S
<b>BarCode</b>	Code128 (20)	Specifies the type of the bar code generated (symbology). For possible values and implemented bar codes refer to the <a href="#">Bar Code Reference</a> . Enumeration type BarCode <i>e_BarCType</i> or <i>tag_BarCType</i> .	G / S
<b>BarWidthReduction</b>	-1	Reduction of the bar width. The bar width is reduced by the given value in percent of the module width. [0.. no reduction, 100..small bars disappear]	G / S

<b>BCHeightAct (...)</b>	-	<p>BCHeightAct (<i>eUnit</i> As tag_MUnit) As Double</p> <p><i>EUnit</i>: unit of the return value [eMUDefault (0)... unit of actual device context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Returns the actual height of the bar code in the specified units.</p>	Get
<b>BCHeightHdc (...)</b>	-	<p>BCHeightHdc (<i>hDC</i> As Long, <i>nWidth</i> As Long, <i>nHeight</i> As Long, <i>eUnit</i> As tag_MUnit) As Double</p> <p><i>hDC</i>: handle to device context; <i>nWidth</i>, <i>nHeight</i>: size of the bar code (set value) in the unit of the hDC, <i>eUnit</i>: unit of the return value [eMUDefault (0)... unit of actual device context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Calculates the bar code height for a given device context in the desired units. That is meaningful with two-dimensional bar codes (like PDF417). With linear (1D) codes like Code 39 or UPC the height always corresponds to the bounding rectangle. The return value of this function corresponds to <i>nHeight</i>, (in the respective unit) if the height of the bar code object is adapted to the bounding rectangle automatically, i.e. if no fixed line height was indicated for PDF417. If a line height was given, then the return value corresponds to the approximated bar code height (which is calculated based on the given values). MaxiCode has always fixed height and width.</p>	Get
<b>BCWidthAct (...)</b>	-	<p>BCWidthAct (<i>eUnit</i> As tag_MUnit) As Double</p> <p><i>EUnit</i>: unit of the return value [eMUDefault (0)... unit of Device-Context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Returns the width of the actual bar code object in the given units.</p>	Get
<b>BCWidthHdc (...)</b>	-	<p>BCWidthHdc (<i>hDC</i> As Long, <i>nWidth</i> As Long, <i>nHeight</i> As Long, <i>eUnit</i> As tag_MUnit) As Double</p> <p><i>hDC</i>: handle to the device context; <i>nWidth</i>, <i>nHeight</i>: size of the bounding rectangle (set value) in the unit of the hDC, <i>eUnit</i>: unit of the return value [eMUDefault (0)... unit of the device context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Calculates the actually needed bar code width for a given device context in the desired units. The return value of this function equals <i>nWidth</i> (in the respective unit) if no fixed module width was given. In this case the width of the bar code object adapts automatically to the bounding rectangle. If a module width was specified, the return value contains the approximated bar code width, which then is calculated on the basis of the module width and the number of modules needed for the representation of the data.</p>	Get
<b>CDMethod</b>	Default	<p>Calculation method of the check digit. Possible values are listed in the <a href="#">Bar Code Reference</a>. Default: the standard method of each bar code type is used (some bar codes have <i>none</i> as standard method, but optional check digits recommended in their specification).</p> <p>CDMethod has the enum type: <i>e_CDMethod</i> or <i>tag_CDMethod</i>.</p>	G / S
<b>CheckDigits</b>	-	Contains the calculated check digits (as ASCII values of the single check digits).	Get
<b>CountCheckDigits</b>	-	Contains the number of check digits calculated. If there was no check digit calculation performed, the property equals zero (Integer).	Get
<b>CountModules</b>	-	Contains the number of modules in the bar code. A module is the smallest segment of which a bar code consists of. Applying a value to the Modulwidth property can set the module width.	Get
<b>CountRows</b>	-	Number of Rows in the bar code (PDF417, Data Matrix). Only useful for 2D bar codes. For PDF417: the number of Columns in the	Get

		symbol is CountModules divided by CountRows.	
<b>DisplayText</b>	Empty	If another text instead of the bar code data should be printed as human readable text, this has to be done with DisplayText.	G / S
<b>DrawStatus</b>	OK (0)	<p>In the event handler code of "BeforeDraw" you can control the draw process by modifying this property. The DrawStatus is checked by the ActiveX Control before drawing and can be used to proceed in a predefined way.</p> <p>If set to <i>OK</i> the bar code will be drawn, if set to <i>Cancel</i> the drawing will be canceled and if set to <i>Retry</i> the drawing start again (useful when bar code properties are changed within the event handler, in this case the event will be fired again).</p> <p>Enumeration Type: tag_DrawStatus, Values: eDST_Cancel (1), eDST_OK (0), eDST_Retry (2)</p>	G / S
<b>Enabled</b>	True	Defines, if the object is enabled or not (disabled). Default = enabled	G / S
<b>EscapeSequences</b>	False	Defines, if Escape-Sequences (occurring in the utilizable data) shall be translated.	G / S
<b>Font</b>	System dep. (MS Sans Serif)	Font used for the bar code text („human readable text“). Class: IFontType	G / S
<b>FontName</b>	System dep. (MS Sans Serif)	Name of font used for the bar code text („human readable text“). Is used if property <i>Font</i> is not accessible.	G / S
<b>FontSize</b>	System dep. (8.75)	Size of font used for the bar code text („human readable text“). Is used if property <i>Font</i> is not accessible.	G / S
<b>ForeColor</b>	Black	Colors of the bars of the bar code (Type: OLE_COLOR)	G / S
<b>Format</b>	None	Format string, applies a format to the bar code content including special control characters before printing. For possible values look at the <a href="#">Bar Code Reference</a> .	G / S
<b>GuardWidth</b>	0	Width of the „Guard Bar“ in 1/1000 mm. A value of 0 draws no Guard Bar.	G / S
<b>InterpretInputAs</b>	eInt_Default	<p>Defines, how input data should be interpreted.</p> <p>Default, ANSI: characters in the input stream are interpreted as ASCII characters from 0 to 255.</p> <p>eInt_ByteStream: Input data is treated as byte stream. That means, a 16-Bit character in the input stream is interpreted as two ASCII characters (for processing binary and UNICODE data).</p> <p>eInt_BYTE_HILO: like eInt_ByteStream, but Hi and LO bytes are exchanged.</p> <p>eInt_UNICODE: The input data is treated as UNICODE stream. This mode is currently only supported in QR Code (KANJI mode).</p>	G / S
<b>LastError</b>	-	The last occurred error is described as error message.	Get
<b>LastErrorNo</b>	-	The last occurred error is described with an error number. (Overview of error numbers: <a href="#">Bar Code Reference</a> ).	Get
<b>ModulWidth</b>	Default	The width of the smallest segment of which a bar code consists. The entered unit is 1/1000 mm. If not set (default), the module width is automatically adapted to the object size (bounding rectangle) and the amount of utilizable data. If a value is set, the width of the	G / S

		modules and therefore of the whole bar code can be controlled (this may also influence the readability!). If set, the bar code width depends on the amount of data characters contained in the bar code. For most bar code types the module width should not fall below 0.19 mms.	
<b>ModWidthAct (...)</b>	-	<p>ModWidthAct (<i>eUnit</i> As tag_MUnit) As Double</p> <p><i>eUnit</i>: unit of the return value [eMUDefault (0)... unit of the device-context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Returns the module width of the actual bar code in the given unit.</p>	Get
<b>ModWidthHdc (...)</b>	-	<p>ModWidthHdc (<i>hDC</i> As Long, <i>nWidth</i> As Long, <i>nHeight</i> As Long, <i>eUnit</i> As tag_MUnit) As Double</p> <p><i>hDC</i>: handle of the device context; <i>nWidth</i>, <i>nHeight</i>: size of the bounding rectangle (set value) in the unit of the hDC, <i>eUnit</i>: unit of the return value [eMUDefault (0)... unit of the device context, eMUPixel (1)... unit in pixel, eMUMM (2)... unit in mm]</p> <p>Calculates the module width for a given Device Context in the desired units. Used if no value is applied to the property <i>module width</i> and therefore the module width is calculated automatically.</p>	Get
<b>MustFit</b>	False	<p>If the <i>ModulWidth</i> is set "manually" to a fixed value the bar code could possibly become larger than the bounding rectangle. If this occurs an error code can be generated by setting <i>MustFit</i> to <i>True</i>.</p> <p>When using a fixed module width the bar code size depends on the number of input characters (text property) and don't adapt to the bounding rectangle of the object.</p> <p>In the reverse case: If you do not specify a fixed module width and the module width is therefore calculated automatically, it could happen that a module width smaller than 1 pixel may result (and bars or gaps can't be drawn properly). If this occurs, an error is also produced if <i>MustFit</i> is set to <i>True</i>.</p>	G / S
<b>NotchHeight</b>	-1	Specifies the additional length of the synchronization bars in codes like EAN and UPC. Unit [1/1000 mm]. Default = automatic adjustment.	G / S
<b>OptResolution</b>	False	<p>Used to adapt the module width to the current pixel resolution.</p> <p><i>Advantage</i>: the readability of the bar code does not depend on the resolution (relevant with small resolutions like screen).</p> <p><i>Disadvantage</i>: the real size of the bar code symbol is not identical with the size of the control (Pay attention to the minimum width!).</p> <p>True = Adapt module width, False = No Optimization</p>	G / S
<b>Orientation</b>	0° (0)	Orientation of the bar code. The value is of the type <i>DEGREE</i> or <i>tag_DEGREE</i> . Possible values are <i>deg0</i> (0), <i>deg90</i> (1), <i>deg180</i> (2), <i>deg270</i> (3) for 0°, 90°, 180°, 270°.	G / S
<b>PrintDataText</b>	True	<p>Print the bar code content (utilizable data) as „human readable text“ in the selected font.</p> <p>True = Yes, False = No</p>	G / S
<b>PrintTextAbove</b>	False	<p>Print the text below or above the bar code.</p> <p>False = Below, True = Above</p>	G / S
<b>Quality</b>	-1	The quality of the bar code in percent. [0..unreadable, 100..perfect, no variance]	Get

<b>QualityHdc</b>	-1	<p>QualityHdc (<i>hDC</i> As Long, <i>nWidth</i> As Long, <i>nHeight</i> As Long) As Long</p> <p><i>hDC</i>: Handle to device context; <i>nWidth</i>, <i>nHeight</i>: size of the bounding rectangle in the current device unit of hDC.</p> <p>Calculates the quality of a bar code for the given device context in percent. [0..unreadable, 100..perfect, no variance]</p>	Get
<b>Ratio</b>	-	Ratio of the bars and gaps (as string). Default ratios are used if empty. Possible values depend on the property <i>RatioHint</i> . For more information refer to <a href="#">Print ratio</a> und <a href="#">Parameters</a> .	G / S
<b>RatioDefault</b>	Default	Value of the default ratio of the selected bar code type (contains the default value of the property <i>Ratio</i> ).	Get
<b>RatioHint</b>	Default	Possible format of the ratio string (used only as hint). Depends on the selected bar code. Learn more about it in the section <a href="#">Parameter</a> of the Barcode-Reference. Default = predefined according to the selected bar code type.	Get
<b>SuppressErrorMsg</b>	False	Suppress all Error Messages (if <i>True</i> ). When an error is occurring, nothing (instead of a "X") will be drawn.	G / S
<b>Text</b>	„Adjust Properties“	Bar code content (utilizable data) as string. The possible values depend on the selected bar code type, therefore alphanumeric or only purely numeric characters are permitted - otherwise an error is produced. Permitted characters: check out the <a href="#">Bar Code Reference</a> . Hint for programming Visual Basic: after setting the text property the command „DoEvents“ should be inserted ( <a href="#">read more about...</a> ).	G / S
<b>TextAlignment</b>	eAIDefault	Alignment of the human readable text. The value is of the type <i>e_BCAlign</i> . Possible values are <i>eAIDefault</i> (0), <i>eAILeft</i> (1), <i>eAIRight</i> (2), <i>eAICenter</i> (3) for default, left, right, center.	G / S
<b>TextColor</b>	Black	Color of the human readable text (VB-type: <i>OLE_COLOR</i> )	G / S
<b>TextDistance</b>	-1	Distance of the human readable text to the bar code. Unit [1/1000 mm]. Default = Distance is calculated automatically	G / S

#### 4.2.2 PDF417 Properties

Name	Default	Description	Get/Set
<b>PDF417_Cols</b>	-1	The number of columns that should be used for data representation. The value must be between 3 and 90. Default value = the number of columns is set automatically (Auto mode).	G / S
<b>PDF417_ECLLevel</b>	-1	Defines the error correction level. The value can be set between 0 (only error recognition) and 8 (highest). Default = the level is set automatically according to the quantity of data (levels 2 to 5).	G / S
<b>PDF417_RatioRowCol</b>	Empty	Defines the ratio between rows and columns in the PDF417 symbol. Default: 2:1. Input mask: „<row>:<columns>“ (e.g. „3:1“). This setting only makes sense, if neither the number of columns nor the number of rows is set to a fixed value (both = AutoMode).	G / S



<b>PDF417_RowHeight</b>	-1	The height of an individual row in 1/1000 mm. Default = the row height is calculated automatically	G / S
<b>PDF417_Rows</b>	-1	The number of rows that should be used for data representation. The value must be between 3 and 90. Default = the number of rows is set automatically (Auto mode).	G / S

#### 4.2.3 MaxiCode Properties

Name	Default	Description	Get/Set
<b>MAXI_AppendCount</b>	-1	Used for "Structured Append" (several MaxiCode symbols connected in series). Sets the number of MaxiCode bar codes, which are used in total for "Structured Append". Default = no "Structured Append"	G / S
<b>MAXI_AppendIndex</b>	-1	Used for „Structured Append“. Sets the index of the actual MaxiCode symbol within the item chain. Depending on this index, the data will be reconstructed by the scanner after the reading process.	G / S
<b>MAXI_CountryCode</b>	-	Country Code. Used in the modes 2 and 3 (SCM). Range: values between 000 - 999 (string). Refer also to <a href="#">Setting SCM parameters</a> .	G / S
<b>MAXI_Date</b>	Actual year	Under <i>Date</i> a year (the last two digits – the century) can be set, which will be inserted automatically into the data stream on a pre-defined place (before the utilized data). Scope: 00 to 99 (string). Relevantly in particular Open System Standards. But the property <i>Preamble</i> must be set to <i>True</i> . Refer also to <a href="#">Setting SCM parameters</a> .	G / S
<b>MAXI_Mode</b>	4	Operating mode (Long Int): Mode 2: SCM Numeric – Structured Carrier Message (Postal Code only numeric, up to 9 digits) Mode 3: SCM Alphanumeric – Structured Carrier Message (Postal Code alphanumeric, up to 6 chars) Mode 4 for numeric and alphanumeric character sequences (Standard Error Correction). Mode 5: Full EEC – like Mode 4, but with Enhanced Error Correction (safer, but less useable data)	G / S
<b>MAXI_PostalCode</b>	-	The Postal Code. Used in the modes 2 and 3 (SCM - "Structured Carrier Message"). String with altogether 9 digits or also different characters (depends on MAXI_Mode). Refer also to <a href="#">Setting SCM parameters</a> .	G / S
<b>MAXI_Preamble</b>	False	Preamble function: can be switched on or off. If <i>True</i> , the year contained in the property <i>MAXI_Date</i> will be inserted. Relevant in particular Open System Standards only. Refer also to <a href="#">Setting SCM parameters</a> .	G / S
<b>MAXI_ServiceClass</b>	-	Service Class. Used in the modes 2 and 3 (SCM). Scope: values between 000 and 999 as string. Refer also to <a href="#">Setting SCM parameters</a> .	G / S

<b>MAXI_Undercut</b>	-1	The <i>Undercut</i> influences the diameter of the bar code items (hexagon). It can be changed, if reading problems with the used output medium occur. Scope (long int): 0 to 100 [%]. Default value is a size of 75%.	G / S
----------------------	----	--	-------

#### 4.2.4 Data Matrix Properties

Name	Default	Description	Get/Set
<b>DM_Format</b>	eDMPPr_Default	Code Format (the value is of the type tagE_DMFormat): Default: standard format. UCC/EAN: used for encoding UCC/EAN application identifiers (FNC1 is encoded as 1 <sup>st</sup> code word). Industry: supports peculiar industry formats (FNC2 at 2 <sup>nd</sup> position) Macro 05: []>Rs05Gs is encoded at the beginning of the code Macro 06: []>Rs06Gs is encoded at the beginning of the code	G / S
<b>DM_Size</b>	eDMSz_Default	Size of the symbol (type: tagE_DMSizes). The size is given in rows and columns and can be between 10 x 10 and 144 x 144 for squares or 8 x 18 and 16 x 48 for rectangles. Default is the minimum square size.	G / S
<b>DM_Rectangular</b>	False	Determines if the symbol should be displayed as square (False: default) or rectangle (true).	G / S
<b>DM_AppendCount</b>	-1	Used for "Structured Append" (several Data Matrix symbols connected in series). Defines the number of symbols used in connection. The value may be between 2 and 16.	G / S
<b>DM_AppendIndex</b>	-1	Used for "Structured Append". Defines the index of the symbol in the connected items. Depending on this index, the data will be reconstructed by the scanner after the reading process. The value may range between 1 and 16.	
<b>DM_AppendFileID</b>	-1	Used for "Structured Append". The FileID is the identifier of the item chain. It must have the same value in all symbols of the chain!	

#### 4.2.5 QR Code Properties

Name	Default	Description	Get/Set
<b>QR_ECLLevel</b>	eQREC_Medium	Error correction level (L)ow: lowest level, recovery capacity up to 7%. (M)edium: 15% (Q)uartil: 25% (H)igh: highest level, 30%	G / S
<b>QR_Format</b>	eQRPr_Default	Code Format (enumeration type tagE_QRFormat): Default: default format. UCC/EAN: for UCC/EAN Application Identifiers (FNC1 is added	G / S

		on first position).  Industry: Industry format (FNC2 at second position). If you choose this setting, you also have to set the Application Indicator property (Property <i>QR_FmtAppIndicator</i> ).	
<b>QR_FmtAppIndicator</b>	Leer	Only in connection with Industry-Format. The Application Indicator determines, to which industry format the input stream corresponds. Possible values: 2 digits or 1 letter (upper and lower case).	G / S
<b>QR_Mask</b>	eQRMask_Default	Mask that is applied to the QR Code to achieve the highest possible readability.  Default: automatic calculation  values from 0 to 7. Makes sense in time-critical applications.	G / S
<b>QR_Version</b>	eQRVers_Default	Symbol version (=size). Possible values are:  default: automatic calculation  Version 1: 21 x 21 modules  Version 2: 25 x 25 modules  ...  Version 40: 177 x 177 modules	G / S
<b>QR_AppendCount</b>	-1	Used in "Structured Append" mode (chaining of 2 or more QR Code symbols). Defines the total number of the symbols in one chain. Possible values: 2 to 16.	G / S
<b>QR_AppendIndex</b>	-1	Used in "Structured Append" mode. Defines the index of a symbol in the symbol chain. The order of the QR Code symbols in this chain depends on this index. Possible values: 1 to 16.	G / S
<b>QR_AppendParity</b>	-1	Used in "Structured Append" mode. The Parity Byte is the identifier of a QR Code chain (important if there is more than one symbol chain). Each symbol in the chain must have the same value for the parity byte. Calculate the Parity Byte with the method <i>QR_StructAppParity</i> .	G / S
<b>QR_StructAppParity</b>	-	QR_StructAppParity (sText As String) As Integer  sText: content of the whole QR Code chain.    Used in "Structured Append" mode. Calculates the parity byte of a symbol chain. After calculating the value set the parity bytes of all the symbols in the symbol chain to this value.	Get

#### 4.2.6 Codablock F Properties

Name	Default	Description	Get/Set
<b>CBF_Columns</b>	-1	The number of columns that should be used for data representation. The value must be between 4 and 62. Default value = the number of columns is set automatically (Auto mode).	G / S
<b>CBF_Format</b>	eCBFPr_Default	Code Format (enumeration type tagE_CBFFormat):  Default: default format.  UCC/EAN: for UCC/EAN Application Identifiers (FNC1 is added on first position).	G / S
<b>CBF_RowHeight</b>	-1	The height of an individual row in 1/1000 mm. Default = the row height is calculated automatically	G / S

<b>CBF_Rows</b>	-1	The number of rows that should be used for data representation. The value must be between 2 and 44. Default = the number of rows is set automatically (Auto mode).	G / S
<b>CBF_RowSeparatorHeight</b>	-1	The height of a row separator in 1/1000 mm. Default = the row separator height is calculated automatically	G / S

#### 4.2.7 Barcode Properties within Event-Handlers

The bar code properties are internally calculated and updated within the Redraw (Paint-Event). If any bar code property is set within an Event Handler (e.g. setting the text property within onClick), then the (new) redrawing of the bar code takes place only after the current Event Handler has been processed. That can lead to non-actual values when getting a property within an event handler.

Example: If you would like to retrieve the property "CountModules" after modifying the bar code text within an event handler, inexact results would be returned (until the current event handler has been finished). Therefore in Visual Basic it would be necessary to insert the instruction "DoEvents" before getting a property. That guarantees that an internal Redraw takes place and valid values are returned. The DoEvents instruction causes that all events (thus also the pending new drawing of the bar code) are processed before the actual process is proceeding. When calling the BCDraw-method, the internal recalculating of the bar code properties takes place in any case (independent of DoEvents).

### 4.3 Methods

In the following outline you find the methods of the TBarCode Control separated according to their use for standard and web applications. Visual Basic and other development environments are offering additional "standard methods" for ActiveX Controls, they not documented here (e.g. SetFocus (), Move (),...).

#### 4.3.1 Methods for Standard Applications

Method (parameter), Result	Description
<b>AboutBox ()</b>	Pops up the About Box with information about TBarCode ActiveX
<b>CopyToClipboard ()</b> On error → standard exception	Copies the bar code into the Windows Clipboard. This gives the ability to insert it into other applications (data format = EMF).
<b>CopyToClipboardEx (hDC As Long, nWidth As Long, nHeight As Long, sFileName As String)</b> On error → standard exception <b>hDC:</b> handle of the Device Context. If hDC = 0 (standard) a screen device context will be used. <b>nWidth, nHeight:</b> width and height of the bar code. If values <= 0 are used, the size of the object will be taken from the actual screen resolution (pixel). <b>sFileName:</b> name of the output file (*.EMF, a path can be used). If sFileName = NULL or an empty string ("") no file will be written. If sFileName is not valid, the return value will be <i>False</i> .	Copies the bar code in EMF (Enhanced Metafile Format) into the clipboard. For extended applications this method allows to specify the device context and an additional output file (optional use). The file will be written in EMF.  Note: Do not use a printer device context with screen standard sizes, because the coordinates do not agree!

<b>Licencing ()</b>	<p>Calls the dialog box for manual licensing of the TBarCode Control. Gives the ability to license the product completely new or afterwards, when the application has been delivered.</p>
<p><b>LicenseMe (sLicensee As String, eKind As tag_licKind, nLicenses As Long, sLicenseKey As String, eProductID As tag_licProduct)</b></p> <p><b>sLicensee:</b> name of the licensee (string).</p> <p><b>eKind:</b> kind of license, enums: <i>eLicKindSingle</i> (1), <i>eLicKindSite</i> (2), <i>eLicKindDeveloper</i> (3)</p> <p><b>nLicenses:</b> number of licenses (usual 1).</p> <p><b>sLicenseKey:</b> license key (to be ordered at TEC-IT)</p> <p><b>EProductID:</b> Product ID = <i>eLicProd1D</i> (8), <i>eLicProd2D</i> (9)</p>	<p>Registers the license data for TBarCode ActiveX in Products in the windows registry (in exchange for manual licensing). The license data (license key) can be ordered at TEC-IT (<a href="http://www.tec-it.com/order">http://www.tec-it.com/order</a>). Suggested calling: once at the application startup. At least once, when doing the first setup of your own application for each windows system.</p> <p>The Product ID indicates which bar code types will be licensed. One-dimensional, linear bar codes (1D = Code128, UPC, EAN, Code39,...) or two-dimensional bar codes (2D = PDF417, MaxiCode, Data Matrix).</p>
<p><b>BCDraw (hDC As Long, nLeft As Long, nTop As Long, nWidth As Long, nHeight As Long)</b></p> <p>on error → standard exception</p> <p><b>hDC:</b> handle to Device Context (where you want to print to).</p> <p><b>nLeft, nTop:</b> coordinates of the left top point of the bar code relative to hDC.</p> <p><b>nWidth, nHeight:</b> width and height of the bar code</p>	<p>This method is used to print a bar code to a given device context. The main usage is to print out a bar code on a printer. The unit of the coordinates and sizes depends on the mapping mode of the selected device context (Pixel, inch, mm).</p>
<b>Refresh ()</b>	<p>Recalculates and redraws the content of the ActiveX Control. You can do this by posting a Paint Event to the message queue.</p>
<p><b>SaveImage (sFileName As String, elmageType As tag_ImageType, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long)</b></p> <p>on error → standard exception</p> <p><b>sFileName:</b> name of the output file (*.JPG, *.BMP, *.EMF,...)</p> <p><b>elmageType:</b> type of the image / bitmap (refer <a href="#">image types</a>).</p> <p><b>nXSize, nYSize:</b> size of the bar code in [pixel]</p> <p><b>nXRes, nYRes:</b> resolution in [dpi]</p>	<p>Saves the content of the bar code object as Image File (Format Bmp, Jpg, Emf,...). The values of <i>XSize</i> and <i>YSize</i> should be increased, if the bar code is not readable or if the bar code contains more data (as usual). The values of <i>XRes</i> and <i>YRes</i> can affect the printing size when printing from graphic- or painting programs. A description of the parameter <i>elmageType</i> can be found later in the section: <a href="#">Image types</a>.</p> <p>Note: If used within web applications (ASP, PHP), ensure that the web server (user) has the rights for writing into the desired directory.</p>
<p><b>SaveImageEx (hDC As Long, sFileName As String, elmageType As tag_ImageType, nQuality As Long, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long)</b></p> <p>on error → standard exception</p> <p><b>hDC:</b> handle to Device Context (to be saved with)</p> <p><b>sFileName:</b> name of the image file (*.JPG, *.BMP, *.EMF,...)</p> <p><b>elmageType:</b> type of image / bitmap</p> <p><b>nQuality:</b> quality of the image in connection with a compression algorithm (refer <a href="#">compression modes</a>).</p> <p><b>nXSize, nYSize:</b> size of the bar code (unit as hDC)</p> <p><b>nXRes, nYRes:</b> resolution (unit as hDC)</p>	<p>Saves the content of the bar code object as Image File in connection with a desired device context (if not needed <i>hDC</i> can be set to 0). The values of <i>XSize</i> and <i>YSize</i> should be increased, if the bar code is not readable or if the data density is high. The values of <i>XRes</i> and <i>YRes</i> can affect the printing size when printing from graphic- or painting programs.</p> <p>The image quality sets the compression algorithm (depending on the image type), too. A description of the parameters <i>elmageType</i> and <i>nQuality</i> can be found later in the section: <a href="#">Image types</a>.</p> <p>With vector EPS files you can choose between using Windows fonts (1) and using postscript compatible fonts (1).</p> <p>Note: If used within web applications (ASP, PHP), ensure that the web server (user) has the rights for writing into the desired directory.</p>

### 4.3.2 Methods for Web Applications

TBarcode ActiveX offers special methods for web applications. With these methods simple integration of bar codes into ASP (Active server Pages), PHP (PHP Hypertext Preprocessor) or other dynamic web extensions become possible.

Note: the methods **ConvertToStream** and **ConvertToStreamEx** are available only in connection with a Developer- or Web-License (but can be tested within the demo version).

PHP: Because the data type „Variant“ is not full supported within PHP, we suggest for PHP applications to use the method *SaveImage* to save the bar code temporarily as image file (which can be referred to at the client-side within the HTML-Code).

We call your attention to the section [Resolution and Readability](#) and to our ASP and PHP samples, obtainable at our download area: <http://www.tec-it.com/Download/>

Method (parameter), Result	Description
<b>ConvertToStream</b> ( <i>elmageType As tag_ImageType, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long</i> )  Returns: DataStream As Variant (supported by ASP)  <b>elmageType</b> : type of the image / bitmap: <i>eIMBmp</i> (0), <i>eIMJpg</i> (4), <i>eIMPng</i> (6), <i>eIMTif</i> (7); not supported: <i>eIMEmf</i> (1), <i>eIMEps</i> (2), <i>eIMGif</i> (3), <i>eIMPcx</i> (5);  <b>nXSize, nYSize</b> : size of the bar code in [pixel]  <b>nXRes, nYRes</b> : resolution in [dpi] (don't affects the browser view).	Returns the content of the object (bar code) as Image data stream, which can be sent to the client browser (e.g. Response.BinaryWrite). The image format is selectable (for web-applications: JPG, PNG or BMP).  The size of the bitmap [Pixel] and the resolution [dpi] can be selected separately for X and Y dimension.  The values of XSize and YSize should be set higher, if the bar code is not readable or if the bar code requires high data density. The values of XRes and YRes can affect the printing size when printing from graphic- or painting programs (but not when printing from the browser). A description of the parameter <i>elmageType</i> can be found later in the section: <a href="#">Image types</a>
<b>ConvertToStreamEx</b> ( <i>hDC As Long, elmageType As tag_ImageType, nQuality As Long, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long</i> )  Returns: DataStream As Variant (supported by ASP)  <b>hDC</b> : handle to Device Context; <b>nQuality</b> : image quality in connection with compression (refer to <a href="#">compression modes</a> ).  Other parameters – look at ConvertToStream above.	Like above it returns the content of the object (bar code) as Image data stream, which can be sent to the client browser. In addition you can control the device context and the quality (relevant with compression algorithms) of the image.  The image data format, the size [Unit defined by hDC] and the resolution [dpi] for X and Y dimension is selectable.
<b>SaveImage</b> ( <i>sFileName As String, elmageType As tag_ImageType, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long</i> )  For more details refer to <a href="#">previous section</a> .	Saves the content of the bar code object as image file. (bitmap data format: Bmp, Jpg, Emf,...).

**SavImageEx (hDC As Long, sFileName As String, elmageType As tag\_ImageType, nQuality As Long, nXSize As Long, nYSize As Long, nXRes As Long, nYRes As Long)**

For more details refer to [previous section](#).

Saves the content of the bar code object as image file. (bitmap data format: Bmp, Jpg, Emf,...). For enhanced use, this method allows to control the device context and quality/compression parameters.

### 4.3.3 Error Handling

Main change since V2.1.8: In V2.1.8 a status code was returned – now from V3.x (and later) an exception is raised if an error occurs.

This exception can be handled in Visual Basic (e. g.) in the following way:

```
On Error Goto ErrHandler
Object.BCDraw (hDC, 0, 0, 500, 300)
...
ErrHandler:
...
```

Within the ErrHandler you can test the TBarCode Properties *LastErrorNo* and *LastError*, or the System *Err*-object.

## 4.4 Events

The TBarCode ActiveX object fires the following events:

Event	Description
<b>Click</b>	Fired when the ActiveX control was clicked
<b>DoubleClick</b>	Fired when the ActiveX control was double-clicked
<b>MouseDown</b>	Fired when a mouse button is pressed when over the control
<b>MouseUp</b>	Fired when a mouse button is released when over the control
<b>MouseMove</b>	Fired when the mouse cursor is moved over the control
<b>BeforeDraw</b>	Fired before the bar code will be drawn. This event is not fired when using SavImage / ConvertToStream methods.

## 5 Program Interface (API) DLL

### 5.1 General

The DLL supplies the underlying data-structure and code-functionality for the ActiveX. So the DLL functions are nearly equivalent to the ActiveX properties and methods. Note: the DLL has the same extension as the ActiveX (= "OCX"), that's because of its internal data structure - but you can use it as DLL!

Hint: when using the DLL make sure that it's located in a path which is included in the "PATH" environment variable. If you are using image functions (SaveImage, ConvertToStream) – you have to put the VIC32.DLL to the same path as the DLL (or a windows system path).

Include the files "**TECBarcode.h**" and "**TECBCenum.h**" to get full access to the DLL functions within C/C++. "**TECBarcode.h**" contains the function definitions; "**TECBCenum.h**" contains all enumerations.

#### 5.1.1 Basic Sequence

The function calls and basic steps to produce a bar code are as following (in the appropriate order):

Function call	Description
<b>BCLicenseMe</b>	This function licenses the DLL and inserts the license key into the registry. Licensing must be performed at least once per computer system.  For developer licenses owning a "Mem:" License Key that is not saved into the registry: call this function each time you start up your application (or before you print a bar code).
<b>BCAlloc</b>	This function sets up and initializes the internal bar code info-structure. It must be called before any other BCxxxx function. You receive a handle that is used for all other function calls.
<b>BCSetBCType</b>	Sets the type of the bar code (symbology) e. g. Code39, Code128, UPC, EAN, 2OF5...
<b>BCGetMaxLenOfData</b>	Optional: This function returns the maximum allowed number of chars for the bar code text (Raw Data) – e. g. for EAN8 it would return 8. This check is optional, but recommended if you do not exactly know the length of data input for your bar code.
<b>BCSetText</b>	Sets the data content (data characters) of the bar code.
<b>BCSetModWidth</b>	Optional: With this function you can specify the module width. If not specified, the module width adapts automatically to the bar code dimensions. Within this step you can set other bar code properties, as you need them.
<b>BCCheck</b>	This function checks if the data characters are valid for the selected bar code type. If invalid data was encountered it returns an error-code <> 0. It must be called before BCCalcCD ().
<b>BCCalcCD</b>	This function computes the check digit(s) for the given data. It must be called before BCCreate ().
<b>BCCreate</b>	This function prepares the bar code info-structure to be drawn with BCDraw. It returns ErrOk if everything is ok. If not, it returns an error code that specifies the error in more detail.
	After BCCreate you can apply the methods GetWidth, GetHeight, CopytoClipboard, etc.



<b>BCDraw</b>	This function draws the bar code into the given device context. The bar code dimensions are set through handling over the coordinates of a bounding rectangle. No special mapping is performed.
<b>BCFree</b>	This function de-initializes the bar code info-structure and frees allocated memory. It must be called as last function.

If any of the BCxxxx functions in the above described order returns an error code not equal to zero than DO NOT call subsequent BCxxxx functions except of BCDeinit (). An error code <> 0 indicates an error condition at subsequent calls (except to BCDeinit) – they may fail and produce unexpected results.

## 5.2 Function Reference

General: to learn more about bar codes and the adaptation of the bar code characteristics to your needs look at the information provided in the other sections of this documentation (the properties of ActiveX and DLL are vastly the same).

### 5.2.1 Init- / Deinit Functions

Function	Description
<b>BCAlloc (t_BarCode** pBarCode)</b> <b>pBarCode:</b> Ptr to bar code structure (OUT); <b>Returns:</b> <i>ERRCODE</i> Err_OK if OK	The <i>BCAlloc</i> function allocates and initializes the internal info structure to store a bar code in memory. It sets the pointer that you handle over to this function. For each bar code you want to use, you have to call this function to get an appropriate handle (pBarCode).
<b>BCFree (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code structure (IN); <b>Returns:</b> <i>ERRCODE</i> Err_OK if OK	De-initializes and frees the bar code info-structure.

### 5.2.2 Set / Get Functions (Properties)

Function	Description
<b>BCSetColorBC (t_BarCode* const pBarCode, COLORREF color)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>color:</b> color; <b>Returns:</b> <i>ERRCODE</i>	Sets bar code color (Default it is black).
<b>BCGetColorBC (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>Returns:</b> <i>COLORREF</i> color	Gets the bar code color.
<b>BCSetFontColor (t_BarCode* const pBarCode, COLORREF color)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>color:</b> color; <b>Returns:</b> <i>ERRCODE</i>	Sets the color of the bar code font.

<b>BCGetColorFont (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>Returns:</b> COLORREF color	Gets the color of the bar code font.
<b>BCSetColorBk (t_BarCode* const pBarCode, COLORREF color)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>color:</b> color; <b>Returns:</b> ERRCODE	Sets the background color of the bar code.
<b>BCGetColorBk (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code structure; <b>Returns:</b> COLORREF color	Gets the background color of the bar code.
<b>BCSetBkMode (t_BarCode* const pBarCode, LONG nMode)</b> <b>pBarCode:</b> Ptr to bar code; <b>nMode:</b> background mode; <b>Returns:</b> ERRCODE	Sets the background mode for the bar code painting (Transparent = 1, Opaque = 2).
<b>BCGetBkMode (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nMode;	Gets the background mode.
<b>BCSetLogFont (t_BarCode* const pBarCode, const LOGFONT* lf)</b> <b>pBarCode:</b> Ptr to bar code; <b>lf:</b> Ptr to logfont; <b>Returns:</b> ERRCODE	Sets the font (face, size...) of the bar code text. Handle over a log font structure whose <i>lfHeight</i> field must be set to the point size of the font! For printing and SaveImage: LOGFONT font size = size [point] Painting on the screen (Screen DC): pFont → lfHeight = -MulDiv (pointsize, dc. GetDeviceCaps (LOGPIXELSY), 72);
<b>BCGetLogFont (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> Ptr to logfont;	Gets the font info as a pointer to a logfont structure.
<b>BCSetBCType (t_BarCode* const pBarCode, e_BarCType eType)</b> <b>pBarCode:</b> Ptr to bar code; <b>eType:</b> bar code type; <b>Returns:</b> ERRCODE	Sets the type of bar code (symbology) you want to use. Look at the <a href="#">Bar Code Reference</a> to get an overview of the enumerations.
<b>BCGetBCType (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_BarCType etype	Gets information of the bar code type currently in use.
<b>BCSetCDMethod (t_BarCode* const pBarCode, e_CDMMethod eMethod)</b> <b>pBarCode:</b> Ptr to bar code; <b>eMethod:</b> check digit method; <b>Returns:</b> ERRCODE	Sets the method of check digit calculation for the bar code. Look at the <a href="#">Bar Code Reference</a> to get an overview of the enumerations.
<b>BCGetCDMethod (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_CDMMethod eMethod	Gets the check digit method currently in use.

<b>BCSetAutoCorrect (t_BarCode* const pBarCode, BOOL bAutoCorrect)</b> <b>pBarCode:</b> Ptr to bar code; <b>bAutoCorrect:</b> on/off (T/F); <b>Returns:</b> ERRCODE	Sets Auto correct for bar code type 2OF5 ITF on / off. Should always be TRUE. For Code 2OF5 ITF it inserts a leading zero if the number of data digits is odd.
<b>BCGetAutoCorrect (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bAutoCorrect;	Gets the Auto correct property (TRUE = On, FALSE = Off).
<b>BCSetRotation (t_BarCode* const pBarCode, e_Degree eRotation)</b> <b>pBarCode:</b> Ptr to bar code; <b>eRotation:</b> rotation of barcode; <b>Returns:</b> ERRCODE	Sets the rotation of the bar code (enumerations for eRotation: deg0, deg90, deg180, deg270).
<b>BCGetRotation (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_Degree eRotation	Gets the rotation of the bar code.
<b>BCSetPrintText (t_BarCode* const pBarCode, BOOL bReadable, BOOL bAbove)</b> <b>pBarCode:</b> Ptr to bar code; <b>bReadable:</b> print readable text (T/F); <b>bAbove:</b> print text above bar code (T/F); <b>Returns:</b> ERRCODE	Sets if the "human readable text" should be printed and (if yes) whether the text should be printed above or below the bar code. Default: text will be printed and below the bar code.
<b>BCGetPrintText (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bReadable	Gets information whether bar code text will be printed (TRUE = Yes, FALSE = No).
<b>BCGetTextAbove (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bAbove	Gets information whether bar code text will be printed above or below the bar code (TRUE = Above, FALSE = Below).
<b>BCSetGuardWidth (t_BarCode* const pBarCode, LONG nGuardWidth)</b> <b>pBarCode:</b> Ptr to bar code; <b>nGuardWidth:</b> guard width; <b>Returns:</b> ERRCODE	Sets the width of the guard bar in [1/1000 mm].
<b>BCGetGuardWidth (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nGuardWidth	Gets the width of the guard bar in [1/1000 mm].
<b>BCSetTextDist (t_BarCode* const pBarCode, LONG nTextDist)</b> <b>pBarCode:</b> Ptr to bar code; <b>nTextDist:</b> text distance; <b>Returns:</b> ERRCODE	Sets the text distance: bar code <--> text [1/1000 mm].
<b>BCGetTextDist (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nTextDist	Gets the distance between bar code and text [1/1000 mm].

<b>BCSetNotchHeight (t_BarCode* const pBarCode, LONG nHeight)</b> <b>pBarCode:</b> Ptr to bar code; <b>nHeight:</b> notch height; <b>Returns:</b> ERRCODE	Sets the notch height for the bar code types EAN and UPC in the unit [1/1000 mm].
<b>BCGetNotchHeight (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nHeight	Gets the notch height [1/1000 mm].
<b>BCSetTranslateEsc (t_BarCode* const pBarCode, BOOL bTranslate)</b> <b>pBarCode:</b> Ptr to bar code; <b>bTranslate:</b> Translation On/Off (T/F); <b>Returns:</b> ERRCODE	Enables or Disables the translation of escape sequences within the bar code text. (See <a href="#">ESC Sequences</a> for more information).
<b>BCGetTranslateEsc (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bTranslate	Gets info about translation of escape sequences (TRUE = On, FALSE = Off).
<b>BCSetMustFit (t_BarCode* const pBarCode, BOOL bMustFit)</b> <b>pBarCode:</b> Ptr to bar code; <b>bMustFit:</b> MustFit On/Off (T/F); <b>Returns:</b> ERRCODE	Sets the "MustFit" flag. "MustFit" means that the bar code must fit into the bounding rectangle (of the OLE container) during drawing. Important when using a fixed module width and the bar code width (not the object width) therefore changes corresponding to the amount of input characters of the Text property. Also "MustFit" verifies if the module width is less than 1 pixel (→Error). If MustFit is set to TRUE an error may be produced due to the above reasons during the Paint method.
<b>BCGetMustFit (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bMustFit	Gets information about the "MustFit" flag.
<b>BCSetOptResolution (t_BarCode* const pBarCode, BOOL bOpt)</b> <b>pBarCode:</b> Ptr to bar code; <b>bOpt:</b> Optimizing On/Off (T/F); <b>Returns:</b> ERRCODE	Sets the "Optimise for Resolution" Flag. If set to true, the readability of the bar code gets independent from the given resolution (interesting when used at small output resolutions (e.g. screen). The width of the bar code may shrink. Be care: watch the minimal size of the bar code.
<b>BCGetOptResolution (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL	Gets information about Optimise flag.
<b>BCSetText (t_BarCode* const pBarCode, LPCTSTR szText, LONG nLen)</b> <b>pBarCode:</b> Ptr to bar code; <b>szText:</b> data content of bar code; <b>nLen:</b> length of input string (0.. end of string is indicated by "\0"); <b>Returns:</b> ERRCODE	Sets the bar code text (data content). String may contain alphanumeric or only numeric data according to the bar code type.
<b>BCSetTextW(t_BarCode* const pBarCode, LPCWSTR szText, LONG nLen)</b> <b>pBarCode:</b> Ptr to bar code; <b>szText:</b> data content of bar code; <b>nLen:</b> length of input string (0.. end of string is indicated by "\0"); <b>Returns:</b> ERRCODE	Like BCSetText, but for UNICODE character interpretation. You can use this function only for QR-Code.
<b>BCGetText (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szText	Gets the bar code text (Raw Data). If the string is not terminated by "\0", you have to use <i>BCGetTextLen</i> to read the correct information.

<b>BCGetTextW (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: LPCWSTR	Like BCGetText, but for UNICODE character interpretation.
<b>BCGetTextLen (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: LONG nLen	Gets the length of the bar code text (Raw Data). Used in connection with BCGetText. If nLen = 0, the szText string is terminated by “\0”.
<b>BCIsTextUnicode (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: BOOL	Determines, if actual interpretation mode is ANSI (standard) or UNICODE (text was set with BCSetTextW)
<b>BCSetModWidth (t_BarCode* const pBarCode, LPCTSTR szModWidth)</b> pBarCode: Ptr to bar code; szModWidth: module width; Returns: ERRCODE	Sets the module width in [1/1000 mm] of the bar code. If set, the bar code adapt in its width to the length of the data content. When drawing (Paint) the size of the bounding rectangle must be wide enough to avoid clipping.
<b>BCGetModWidth (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: LPCTSTR szModWidth	Gets the module width (as it was set by BCSetModWidth) as string [1/1000 mm].
<b>BCSetRatio (t_BarCode* const pBarCode, LPCTSTR szRatio)</b> pBarCode: Ptr to bar code; szRatio: ratio string; Returns: ERRCODE	Sets the ratio of the bars to the gaps in the bar code – look at <a href="#">Ratio</a> , <a href="#">RatioHint (Ratio Format)</a> for more information (only for special applications).
<b>BCGetRatio (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: LPCTSTR szRatio	Gets the ratio as string (as it was set by BCSetRatio).
<b>BCSetDisplayText (t_BarCode* const pBarCode, LPCTSTR szText);</b> pBarCode: Ptr to bar code; szText: text to display; Returns: ERRCODE	Sets the text that should be printed as human readable text instead of the bar code data.
<b>BCGetDisplayText (t_BarCode* const pBarCode)</b> pBarCode: Ptr to bar code; Returns: LPCTSTR display text	Gets the text that should be printed as human readable text instead of the bar code data.
<b>BCSetBarWidthReduction (t_BarCode* pBarCode, LONG nFactor)</b> pBarCode: Ptr to bar code; nFactor: reduction factor; Returns: ERRCODE	Sets the bar width reduction factor (in percent).
<b>BCGetBarWidthReduction (t_BarCode* pBarCode);</b> pBarCode: Ptr to bar code; Returns: LONG	Returns the bar width reduction factor (in percent).
<b>BCSetTextAlignment (t_BarCode* const pBarCode, e_BCAAlign eAlign);</b> pBarCode: Ptr to bar code; eAlign: alignment; Returns: ERRCODE	Sets the text alignment.

<b>BCGetTextAlignment (t_BarCode* const pBarCode</b> <b>);</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> <i>e_BCAlign</i>	Returns the text alignment
<b>BCSet_EPS_SubstwDeviceFont (t_BarCode* const pBarCode, Bool bSubstwDevFnt)</b> <b>pBarCode:</b> Ptr to bar code; <b>bSubstwDevFnt:</b> Substitute with Postscript fonts; <b>Returns:</b> <i>ERRCODE</i>	Used with Savelimage / SavelimageEx: Sets in a vector EPS whether the Windows font names are used for Postscript
<b>BCGet_EPS_SubstwDeviceFont (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> <i>BOOL</i>	Used with Savelimage / SavelimageEx: Returns whether a Windows or a Postscript compatible font name is used.

RSS Properties	
<b>BCGet_RSS_SegmPerRow (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code;	Get Data Segments per Row (RSS Expanded Stacked)
<b>BCSet_RSS_SegmPerRow (t_BarCode* const pBarCode, LONG nSegmPerRow)</b> <b>pBarCode:</b> Ptr to bar code; <b>nSegmPerRow:</b> number of Segments/row [2..22]	Set Data Segments per Row (RSS Expanded Stacked)
<b>BCGet_RSS_XRows (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code;	Get number of vertical modules (rows) with height X (=module width)
<b>BCGet_RSS_XCols (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code;	Get number of horizontal modules (cols) with module width X

### 5.2.3 Set / Get Functions for PDF417

Function	Description
<b>BCSet_PDF417_Rows (t_BarCode* const, pBarCode, LONG nRows)</b> <b>pBarCode:</b> Ptr to bar code; <b>nRows:</b> number of rows; <b>Returns:</b> <i>ERRCODE</i>	Sets the number of graphic rows [3..90] when using PDF417. If not set, the number of rows is calculated automatically (automatic mode).
<b>BCGet_PDF417_Rows (t_BarCode* const)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> <i>LONG</i> nRows	Gets the number of graphic rows (as set with <b>BCSet_PDF417_Rows</b> ). Returns only the property value, but not the number of rows actual used in automatic mode.
<b>BCSet_PDF417_Columns (t_BarCode* const pBarCode, LONG nColumns)</b> <b>pBarCode:</b> Ptr to bar code; <b>nColumns:</b> number of cols; <b>Returns:</b> <i>ERRCODE</i>	Sets the number of graphic columns [1...30] when using PDF417. If not set, the number of columns is calculated automatically (automatic mode).

<b>BCGet_PDF417_Columns (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nColumns	Gets the number of graphic columns (as set with <i>BCSet_PDF417_Columns</i> ). Returns not the number of columns actual used in automatic mode.
<b>BCSet_PDF417_ECLevel (t_BarCode* const pBarCode, LONG nLevel)</b> <b>pBarCode:</b> Ptr to bar code; <b>nLevel:</b> Error Correction Level; <b>Returns:</b> ERRCODE	Sets the "Error Correction Level" of PDF417. Possible values are [0...8] where 0 mean only error recognition (no EC) and 8 mean highest Level of EC. If not set, the ECL is chosen automatically for you.
<b>BCGet_PDF417_ECLevel (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nLevel	Gets the value that is set by <i>BC_Set_PDF417_ECLevel</i> .
<b>BCSet_PDF417_RowHeight (t_BarCode* const pBarCode, LONG nHeight)</b> <b>pBarCode:</b> Ptr to bar code; <b>nHeight:</b> row height; <b>Returns:</b> ERRCODE	Sets the height of a PDF417 row in [1/1000 mm]. If not set, the height is calculated according to the bounding rectangle when drawing (BCDraw). If set, ensure that the bounding rectangle is big enough to avoid clipping.
<b>BCGet_PDF417_RowHeight (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nHeight	Gets the value that is set by <i>BCSet_PDF417_RowHeight</i> .
<b>BCSet_PDF417_RowColRatio (t_BarCode* const pBarCode, LPCTSTR szRatio)</b> <b>pBarCode:</b> Ptr to bar code; <b>szRatio:</b> ratio between rows and cols; <b>Returns:</b> ERRCODE	Sets the ratio between rows and columns. Does only work, if neither rows nor columns are set to a fixed value.
<b>BCGet_PDF417_RowColRatio(t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szRatio	Returns the Ratio set by <i>BCSet_PDF417_RowColRatio</i> .

## 5.2.4 Set / Get Functions for MaxiCode

Function	Description
<b>BCSet_Maxi_Mode (t_BarCode* const pBarCode, LONG nMode)</b> <b>pBarCode:</b> Ptr to bar code; <b>nMode:</b> mode [2..5]; <b>Returns:</b> ERRCODE	Sets the operating mode for MaxiCode. See the documentation of the ActiveX properties for more information ( <a href="#">MaxiCode Properties</a> ).
<b>BCGet_Maxi_Mode (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG mode	Gets the value that is set by <i>BCSet_Maxi_Mode</i> .
<b>BCSet_Maxi_Append (t_BarCode* const pBarCode, LONG nSum, LONG nIndex)</b> <b>pBarCode:</b> Ptr to bar code; <b>nSum:</b> number of symbols [2..8]; <b>nIndex:</b> index of actual symbol [1..8]; <b>Returns:</b> ERRCODE	Needed for "Structured Append" (several MaxiCode items connected in series). Sets the number of MaxiCodes that are used in total for "Structured Append" and the index of the actual MaxiCode symbol within this item chain. If not set, no "Structured Append" is used.
<b>BCGet_Maxi_AppendSum (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nSum	Gets the value of the <i>nSum</i> property set by <i>BCSet_Maxi_Append</i> .

<b>BCGet_Maxi_AppendIndex (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nIndex	Gets the value of the <i>nIndex</i> property set by <i>BCSet_Maxi_Append</i> .
<b>BCSet_Maxi_UnderCut (t_BarCode* const pBarCode, LONG nIndex)</b> <b>pBarCode:</b> Ptr to bar code; <b>nUndercut:</b> undercut [0..100%]; <b>Returns:</b> ERRCODE	Undercut of the MaxiCode hexagons, of which the symbol consists of. Indication in percent [1...100]. For scanning problems only.
<b>BCGet_Maxi_UnderCut (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nIndex	Gets the value of the <i>nUndercut</i> property set by <i>BCSet_Maxi_Undercut</i> .
<b>BCSet_Maxi_UsePreamble (t_BarCode* const pBarCode, BOOL bUse, LPCTSTR szDate)</b> <b>pBarCode:</b> Ptr to bar code; <b>bUse:</b> use Preamble (T/F); <b>szDate:</b> preamble date (last 2 digits of year); <b>Returns:</b> ERRCODE	Sets MaxiCode to use the "Preamble function" ("f">...). If <i>bUse</i> = True, the year contained in <i>szDate</i> will be inserted.
<b>BCGet_Maxi_UsePreamble (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL bUse	Gets the value of the <i>bUse</i> property set by <i>BCSet_Maxi_UsePreamble</i> .
<b>BCGet_Maxi_PreambleDate (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szDate	Gets the value of the <i>szDate</i> property set by <i>BCSet_Maxi_UsePreamble</i> .
<b>BCSet_Maxi_SCM (t_BarCode* const pBarCode, LPCTSTR szServiceClass, LPCTSTR szCountryCode, LPCTSTR szPostalCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>szServiceClass:</b> Service Class ["000".."999"]; <b>szCountryCode:</b> Country Code ["000".."999"]; <b>szPostalCode:</b> Postal Code [9 digits or up to 6 chars]; <b>Returns:</b> ERRCODE	<p>In operating mode SCM (mode 2 and 3) you can specify the Service Class, Country Code and Postal Code with this function call.</p> <p>The allowed values for the Postal Code are depending on the operating mode (Mode 3: Postal Code = alphanumeric, up to 6 chars).</p>
<b>BCGet_Maxi_SCMServClass (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szServiceClass	Gets the value of the <i>szServiceClass</i> property set by <i>BCSet_Maxi_SCM</i> .
<b>BCGet_Maxi_SCMCountryCode (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szCountryCode	Gets the value of the <i>szCountryCode</i> property set by <i>BCSet_Maxi_SCM</i> .
<b>BCGet_Maxi_SCMPostalCode (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR szPostalCode	Gets the value of the <i>szPostalCode</i> property set by <i>BCSet_Maxi_SCM</i> .

### 5.2.5 Set / Get Functions for Data Matrix

Refer to "TECBCenum.h" for possible values / enumerations.



Function	Description
<b>BCSet_DM_Size (t_BarCode* const pBarCode, e_DMSizes eSize)</b> <b>pBarCode:</b> Ptr to bar code; <b>eSize:</b> symbol size; <b>Returns:</b> ERRCODE	Sets the size of the Data Matrix symbol
<b>BCGet_DM_Size (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_DMSizes	Returns the actual Data Matrix symbol size.
<b>BCSet_DM_Rectangular (t_BarCode* const pBarCode, BOOL bRect)</b> <b>pBarCode:</b> Ptr to bar code; <b>bRect:</b> rectangular (true/false); <b>Returns:</b> ERRCODE	Sets the symbol shape to be rectangular (true) or square (false).
<b>BCGet_DM_Rectangular (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BOOL	Returns if the symbol was set to be square (false, default) or rectangular (true).
<b>BCSet_DM_Format (t_BarCode* const pBarCode, eDM_Format eFormat)</b> <b>pBarCode:</b> Ptr to bar code; <b>eFormat:</b> code format; <b>Returns:</b> ERRCODE	Sets the code format: standard, EAN/UCC, Industry, Format 05, 06
<b>BCGet_DM_Format (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> eDMFormat	Returns if the active Data Matrix code format.
<b>BCSet_DM_Append (t_BarCode* const pBarCode, LONG nSum, LONG nIndex, LONG nFileID)</b> <b>pBarCode:</b> Ptr to bar code; <b>nSum:</b> number of symbols [2..16]; <b>nIndex:</b> index of actual symbol [1..16]; <b>nFileID:</b> file id; <b>Returns:</b> ERRCODE	Needed for "Structured Append" (several Data Matrix items connected in series). Sets the number of Data Matrix symbols that are used in total for "Structured Append" and the index of the actual Data Matrix symbol within this item chain. The file id identifies the symbol chain. If not set, no "Structured Append" is used.
<b>BCGet_DM_AppendSum (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the value of the <i>nSum</i> property set by <i>BCSet_DM_Append</i> .
<b>BCGet_DM_AppendIndex (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the value of the <i>nIndex</i> property set by <i>BCSet_DM_Append</i> .
<b>BCGet_DM_AppendFileID (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the value of the <i>nFileID</i> property set by <i>BCSet_DM_Append</i> .

### 5.2.6 Set / Get Functions for QR Code

Refer to "TECBCEnum.h" for possible values / enumerations.

Function	Description
<b>BCSet_QR_Version (t_BarCode* const pBarCode, e_QRVersion eVersion)</b> <b>pBarCode:</b> Ptr to bar code; <b>eVersion:</b> symbol version; <b>Returns:</b> ERRCODE	Sets the version (=size) of the QR Code symbol
<b>BCGet_QR_Version (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_QRVersion	Returns the actual QR Code version.
<b>BCSet_QR_Format (t_BarCode* const pBarCode, e_QRFormat eFormat)</b> <b>pBarCode:</b> Ptr to bar code; <b>eFormat:</b> symbol format; <b>Returns:</b> ERRCODE	Sets the code format (standard, EAN/UCC, Industry)
<b>BCGet_QR_Format (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_QRFormat	Returns the active QR Code format.
<b>BCSet_QR_ECLLevel (t_BarCode* const pBarCode, e_QRECLLevel eECLLevel)</b> <b>pBarCode:</b> Ptr to bar code; <b>eECLLevel:</b> error correction level; <b>Returns:</b> ERRCODE	Sets the error correction level of the QR Code symbol (L, M, Q, H)
<b>BCGet_QR_ECLLevel (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_QRECLLevel	Returns the active QR Code error correction level.
<b>BCSet_QR_Mask (t_BarCode* const pBarCode, e_QRMask eMask)</b> <b>pBarCode:</b> Ptr to bar code; <b>eMask:</b> mask to apply; <b>Returns:</b> ERRCODE	Sets the mask pattern for the QR Code symbol (0..7)
<b>BCGet_QR_Mask (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_QRMask	Returns the applied mask.
<b>BCSet_QR_FmtAppIndicator (t_BarCode* const pBarCode, LPCTSTR szIndicator)</b> <b>pBarCode:</b> Ptr to bar code; <b>szIndicator:</b> application indicator; <b>Returns:</b> ERRCODE	Sets the format application indicator. Used to define the industry format.
<b>BCGet_QR_FmtAppIndicator (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR	Returns the format application indicator.
<b>BCSet_QR_Append (t_BarCode* const pBarCode, LONG nSum, LONG nIndex, BYTE bParity)</b> <b>pBarCode:</b> Ptr to bar code; <b>nSum:</b> number of symbols [2..16]; <b>nIndex:</b> index of actual symbol [1..16]; <b>bParity:</b> parity byte; <b>Returns:</b> ERRCODE	<p>Needed for "Structured Append" (several QR Code items connected in series). Sets the number of QR Code symbols that are used in total for "Structured Append" and the index of the actual QR Code symbol within this item chain. The value for <i>bParity</i> has to be calculated with <i>BCCalcStructApp_Parity</i>.</p> <p>If these settings aren't made, no "Structured Append" is used.</p>
<b>BCGet_QR_AppendSum (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the value of the <i>nSum</i> property set by <i>BCSet_QR_Append</i> .

<b>BCGet_QR_AppendIndex (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the value of the <i>nIndex</i> property set by <i>BCSet_QR_Append</i> .
<b>BCGet_QR_AppendParity (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> BYTE	Gets the value of the <i>bParity</i> property set by <i>BCSet_QR_Append</i> .

## 5.2.7 Set / Get Functions for Codablock F

Refer to “TECBCEnum.h” for possible values / enumerations.

Function	Description
<b>BCSet_CBF_Rows (t_BarCode* const pBarCode, LONG nRows);</b> <b>pBarCode:</b> Ptr to bar code; <b>nRows:</b> symbol version; <b>Returns:</b> ERRCODE	Sets the number of graphic rows [2..44] when using Codablock F. If not set, the number of rows is calculated automatically (automatic mode).
<b>BCGet_CBF_Rows (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the number of graphic rows (as set with <i>BCSet_CBF_Rows</i> ). Returns only the property value, but not the number of rows actual used in automatic mode.
<b>BCSet_CBF_Columns (t_BarCode* const pBarCode, LONG nColumns);</b> <b>pBarCode:</b> Ptr to bar code; <b>nColumns:</b> number of columns; <b>Returns:</b> ERRCODE	Sets the number of graphic columns [4..62] when using Codablock F. If not set, the number of columns is calculated automatically (automatic mode).
<b>BCGet_CBF_Columns (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG	Gets the number of graphic columns (as set with <i>BCSet_CBF_Columns</i> ). Returns not the number of columns actual used in automatic mode.
<b>BCSet_CBF_RowHeight (t_BarCode* const pBarCode, LONG nHeight)</b> <b>pBarCode:</b> Ptr to bar code; <b>nHeight:</b> row height; <b>Returns:</b> ERRCODE	Sets the height of a Codblock F row in [1/1000 mm]. If not set, the height is calculated according to the bounding rectangle when drawing (BCDraw). If set, ensure that the bounding rectangle is big enough to avoid clipping.
<b>BCGet_CBF_RowHeight (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nHeight	Gets the value that is set by <i>BCSet_CBF_RowHeight</i> .
<b>BCSet_CBF_RowSeparatorHeight (t_BarCode* const pBarCode, LONG nHeight)</b> <b>pBarCode:</b> Ptr to bar code; <b>nHeight:</b> row height; <b>Returns:</b> ERRCODE	Sets the height of a Codblock F row separator in [1/1000 mm]. If not set, the height is calculated according to the bounding rectangle when drawing (BCDraw). If set, ensure that the bounding rectangle is big enough to avoid clipping.
<b>BCGet_CBF_RowSeparatorHeight (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LONG nHeight	Gets the value that is set by <i>BCSet_CBF_RowSeparatorHeight</i> .

<b>BCSet_CBF_Format (t_BarCode* const pBarCode, e_CBFFormat eFormat)</b> <b>pBarCode:</b> Ptr to bar code; <b>eFormat:</b> symbol format; <b>Returns:</b> ERRCODE	Sets the code format (standard, EAN/UCC)
<b>BCGet_CBF_Format (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> e_CBFFormat	Returns the active Codablock F format.

## 5.2.8 Methods (Drawing, Licensing...)

Function	Description
<b>BCLicenseMe (LPCTSTR lpszLicensee, e_licKind eKindOfLicense, DWORD dwNoOfLicenses, LPCTSTR lpszKey, e_licProduct eProductID)</b> <b>lpszLicensee:</b> Licensee name; <b>eKindOfLicense:</b> kind of license - <i>eLicKindSingle</i> (1), <i>eLicKindSite</i> (2), <i>eLicKindDeveloper</i> (3); <b>dwNoOfLicenses:</b> number of licenses; <b>lpszKey:</b> License Key (format described beside); <b>eProductID:</b> Product ID - <i>eLicProd1D</i> (8), <i>eLicProd2D</i> (9); <b>Returns:</b> ERRCODE (ErrOK if OK)	<p>Licenses the TBarcode DLL with the license key to inactivate the demo mode and to get rid off the crossbar. You receive the license key from TEC-IT Datenverarbeitung GmbH after you have ordered a license.</p> <p>Format of <i>lpszKey</i>: "xxxxxxx" or "HKLM:xxxxxxx" (x=License Key) registers the License-Key within the section "HK Local Machine" of the Windows Registry (Default).</p> <p>"HKCU:xxxxxxx" (x = License Key) registers the License-Key in the section "HKEY Current User" of the windows registry.</p> <p>For Developer Licenses → Format of <i>lpszLicensee</i> = "Mem: xxxxxxxx" ... the product is licensed until the DLL will be unloaded from memory.</p>
<b>BCGetRatioString (e_BarCType eType)</b> <b>eType:</b> bar code type; <b>Returns:</b> LPCTSTR lpszRatioString	Returns the default print ratio for the bar code type handled over to the function. The function returns a string as shown in the <a href="#">Bar Code Overview</a> .
<b>BCGetRatioHint (e_BarCType eType)</b> <b>eType:</b> bar code type; <b>Returns:</b> LPCTSTR lpszRatioHint	Returns the format description of the print ratio (=ratio hint) for the bar code type handled over to the function. The function returns a string as shown in the <a href="#">Bar Code Overview</a> .
<b>BCGetMaxLenOfData (e_BarCType eType)</b> <b>eType:</b> bar code type; <b>Returns:</b> LONG nMaxLen	Returns the exact number of chars that have to be in <i>szText</i> (Raw Data). Returns zero if there is no specified length or an unknown bar code type.
<b>BCCheck (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> ERRCODE (ErrOK if OK)	Checks bar code input data for validity.
<b>BCCalcCD (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> ERRCODE (ErrOK if OK)	Calculates the Check Digit.
<b>BCCreate (t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> ERRCODE (ErrOK if OK)	Creates internal graphical bar code data structure depending on all previous set bar code parameters.

<b>BCDraw (t_BarCode* const pBarCode, HDC hDC, RECT* pRect)</b> <b>pBarCode:</b> Ptr to bar code; <b>hDC:</b> Handle of Device Context; <b>pRect:</b> Ptr to bounding rectangle (a standard windows structure); <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)	Draws bar code into the given device context. The coordinates of the bounding rectangle define the bar code dimensions. No special mapping is performed.
<b>BCGetBCList ()</b> <b>Returns:</b> <i>LPCTSTR*</i> lpszBarcodeArray	Returns list of implemented bar codes (returns a pointer to an array). Use this function in connection with <i>BCGetBCCount ()</i> .
<b>BCGetBCCount ()</b> <b>Returns:</b> <i>LONG</i> nNumberOfListElements	Returns number of implemented bar code types in the bar code list (= length of array).
<b>BCGetCDListByType (e_BarCType eBCType)</b> <b>Returns:</b> <i>e_CDMMethod*</i> eCDMethod	Returns list of implemented check digits (as a text array) dependent of bar code type.
<b>BCGetNameFromEnum (e_CDMMethod eCDMethod)</b> <b>Returns:</b> <i>LPCTSTR</i> lpszCDMethod	Returns name of check digit method.
<b>BCGetCDList ()</b> <b>Returns:</b> <i>LPCTSTR*</i> lpszCDListArray	Returns list of implemented check digit methods (returns pointer to array). Use this function in connection with <i>BCGetCDCount ()</i> .
<b>BCGetCDCount ()</b> <b>Returns:</b> <i>LONG</i> nNumberOfListElements	Returns size of check digit method list.
<b>BCCopyToClipboard (t_BarCode* const pBarCode, LONG nWidth, LONG nHeight)</b> <b>pBarCode:</b> Ptr to bar code; <b>nWidth:</b> width of bar code to copy; <b>nHeight:</b> height of bar code to copy; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)	Copies bar code into clipboard.
<b>BCCopyToClipboardEx (t_BarCode* const pBarCode, HDC hDC, LONG nWidth, LONG nHeight, BOOL fTransparent, COLORREF crBkCol, LPCTSTR szFileName)</b> <b>pBarCode:</b> Ptr to bar code definition; <b>hDC:</b> device context to copy from (default: NULL); <b>nWidth:</b> width of barcode to copy; <b>nHeight:</b> height of barcode to copy; <b>fTransparent:</b> bar code transparent or not; <b>crBkCol:</b> background color (only if <i>fTransparent</i> == FALSE); <b>szFileName:</b> filename (*.EMF), that the barcode should be saved to (NULL for no saving); <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)	Copies bar code into clipboard with various options. You can specify the Device Context, the transparency and the background color of the bar code. Optional the bar code can be saved as EMF-file (Enhanced Metafile).
<b>BCGetCountModules (const t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code definition; <b>Returns:</b> <i>DOUBLE</i> nNumberOfModules	Computes the number of modules in the bar code. A module is the smallest bar/gap element of which bar codes consist of.
<b>BCGetCountRows (const t_BarCode* const pBarCode)</b> <b>pBarCode:</b> Ptr to bar code definition; <b>Returns:</b> <i>DOUBLE</i> nNumberOfRows	Computes the number of rows in the bar code (for 1D / linear symbologies = 1). Useful for 2D symbologies like PDF417 or Data Matrix.
<b>BCGetModuleWidth (t_BarCode* const pBarCode, LPRECT pRect, HDC hDC, e_MUnit eUnit)</b> <b>pBarCode:</b> Ptr to bar code definition; <b>pRect:</b> bounding rectangle; <b>hDC:</b> Device Context Handle; <b>eUnit:</b> unit of measure; <b>Returns:</b> <i>DOUBLE</i> nModuleWidth	Computes the module width for a given Device Context. You can specify the unit, which should be used for the return value.

<p><b>BCGetBarcodeWidth</b> (<i>t_BarCode* const pBarCode, LPRECT pRect, HDC hDC, e_MUnit eUnit</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>pRect:</b> bounding rectangle; <b>hDC:</b> Device Context Handle; <b>eUnit:</b> unit of measure; <b>Returns:</b> <i>DOUBLE</i> nBarcodeWidth</p>	<p>Computes the barcode width for a given Device Context. You can specify the unit, which should be used for the return value.</p>
<p><b>BCGetBarcodeHeight</b> (<i>t_BarCode* const pBarCode, LPRECT pRect, HDC hDC, e_MUnit eUnit</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>pRect:</b> bounding rectangle; <b>hDC:</b> Device Context Handle; <b>eUnit:</b> unit of measure; <b>Returns:</b> <i>DOUBLE</i> nBarcodeHeight</p>	<p>Computes the barcode height for a given Device Context. You can specify the unit of the return value. Hint: use it to get the height of a PDF417 symbol after you have modified the row height. In this case the height of the symbol could be different from the bounding rectangle.</p>
<p><b>BCSavelImage</b> (<i>t_BarCode* const pBarCode, LPCTSTR lpszFileName, e_IMType elmageType, LONG ISize, LONG IYSize, LONG IRes, LONG IYRes</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>lpszFileName:</b> Filename; <b>elmageType:</b> Enumeration indicating the image type saved with; <b>ISize, IYSize:</b> X/Y-Size of the image [pixel]; <b>IRes, IYRes:</b> X/Y-Resolution [dpi]; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)</p>	<p>Saves the bar code to an Image File (Format Bmp, Jpg, Emf...). The values of <i>ISize</i> and <i>IYSize</i> should be increased, if the bar code is not readable. The values of <i>IRes</i> and <i>IYRes</i> can affect printing size when printing from graphic- or painting programs. A description of the parameter <i>elmageType</i> can be found later in the section: <a href="#">Image types</a>.</p>
<p><b>BCSavelImageEx</b> (<i>t_BarCode* const pBarCode, HDC hDC, LPCTSTR lpszFileName, e_IMType elmageType, LONG IQuality, LONG ISize, LONG IYSize, LONG IRes, LONG IYRes</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>hDC:</b> Device Context Handle; <b>lpszFileName:</b> Filename; <b>elmageType:</b> Enumeration indicating the image type; <b>IQuality:</b> Quality; <b>ISize, IYSize:</b> X/Y-Size of the image [pixel]; <b>IRes, IYRes:</b> X/Y-Resolution [dpi]; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)</p>	<p>Saves the bar code to an Image File using a specific Device Context (default = zero).</p> <p>The image quality parameter sets the compression algorithm or other parameters (depending on image type). A description of the parameters <i>elmageType</i> and <i>IQuality</i> can be found later in the section: <a href="#">Image types</a>.</p>
<p><b>BCSavelImageToBuffer</b> (<i>t_BarCode* const pBarCode, LPTSTR* lpszBuffer, e_IMType elmageType, LONG ISize, LONG IYSize, LONG IRes, LONG IYRes</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>lpszBuffer:</b> (OUT) Ptr to Buffer; <b>elmageType:</b> Enumeration indicating the image type saved with; <b>ISize, IYSize:</b> X/Y-Size of the image [pixel]; <b>IRes, IYRes:</b> X/Y-Resolution [dpi]; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)</p>	<p>Draws barcode and saves it as Image to a memory buffer (pass a pointer variable of an arbitrary type as argument). The buffer can be deleted with the API function <i>GlobalFreePtr</i>.</p> <p>The values of <i>XSize</i> and <i>YSize</i> should be increased, if the bar code is not readable or requires a high printing resolution. The values of <i>XRes</i> and <i>YRes</i> can affect the printing size when printing from graphic- or painting programs (but not when printing from the browser!). A description of the parameter <i>elmageType</i> can be found later in the section: <a href="#">Image types</a></p> <p>The buffer will be allocated on the global heap. Get the buffer size in Bytes with <i>GlobalSize((HGLOBAL)GlobalHandle(szBuffer))</i> – and free memory with <i>GlobalFreePtr(...)</i>.</p>
<p><b>BCSavelImageToBufferEx</b> (<i>t_BarCode* const pBarCode, HDC hDC, LPTSTR* lpszBuffer, e_IMType elmageType, LONG IQuality, LONG ISize, LONG IYSize, LONG IRes, LONG IYRes</i>)</p> <p><b>pBarCode:</b> Ptr to bar code definition; <b>hDC:</b> Device Context Handle; <b>lpszBuffer:</b> (OUT) Ptr to Buffer; <b>elmageType:</b> Enumeration indicating the image type; <b>IQuality:</b> Quality; <b>ISize, IYSize:</b> X/Y-Size of the image [pixel]; <b>IRes, IYRes:</b> X/Y-Resolution [dpi]; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)</p>	<p>Like above it saves the bar code into a memory buffer with a selectable image format (pass a pointer variable of an arbitrary type as argument). The buffer can be deleted with the API function <i>GlobalFreePtr</i>. In addition you can control the device context and the image quality (relevant for compression algorithms).</p> <p>The image data format, the size [Unit defined by hDC] and the resolution [dpi] for X and Y dimension can be set. A description of the parameters <i>elmageType</i> and <i>IQuality</i> can be found later in the section: <a href="#">Image types</a>.</p>
<p><b>BCGetCountBars</b> (<i>e_BarCType eBarCType</i>)</p> <p><b>eBarCType:</b> Bar Code Type; <b>Returns:</b> <i>LONG</i> ICountBars</p>	<p>Returns number of bar widths for the barcode type handled over (or -1 if not successful).</p>

<b>BCGetCountSpaces (e_BarCType eBarCType)</b> <b>eBarCType:</b> Bar Code Type; <b>Returns:</b> LONG ICountSpaces	Returns number of space widths for the barcode type handled over (or -1 if not successful).
<b>BCGetErrorText (ERRCODE eCode, LPTSTR szText, size_t nSize)</b> <b>eCode:</b> Error Code; <b>szText:</b> (IO) error text buffer; <b>nSize:</b> buffer size of sztext; <b>Returns:</b> nothing (VOID)	Returns error text to given error code.
<b>BCCalcStructApp_Parity (LPCTSTR szIntData, LONG nIntData)</b> <b>szIntData:</b> input data; <b>nIntData:</b> size of szIntData; <b>Returns:</b> BYTE	Returns the parity byte for given input stream. Used with QR code and structured append.
<b>BCGetQRCodeVersions ()</b> <b>Returns:</b> LPCTSTR*	Returns all possible QR versions (sizes) in an array
<b>BCGetQRCodeVersionCount ()</b> <b>Returns:</b> LONG	Returns the number of possible versions (returned by <i>BCGetQRCodeVersions</i> )
<b>BCGetQuality (t_BarCode* const pBarCode, HDC hDC, RECT* pRect)</b> <b>pBarCode:</b> Ptr to bar code definition; <b>hDC:</b> Device Context Handle; <b>pRect:</b> bounding rectangle; <b>Returns:</b> LONG	Returns the bar code quality in percent. (0 .. unreadable, 100 perfect)
<b>BCGetCheckDigits(t_BarCode* const pBarCode, LPTSTR lpszCDText, LONG nSize);</b> <b>pBarCode:</b> Ptr to bar code definition; <b>lpszCDText:</b> check digit text buffer; <b>nSize:</b> buffer size; <b>Returns:</b> LONG	Returns the calculated check digits (as ASCII values of the single check digits).
<b>BCSetQuietZone (t_BarCode* const pBarCode, LPRECT const prQuietZone, e_QZMUnit eQZMUnit);</b> <b>pBarCode:</b> Ptr to bar code definition; <b>prQuietZone:</b> Ptr to rectangle that specifies quietzone; <b>eQZMUnit:</b> Unit for Quietzone values (mm, Modules, mils...); <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK)	Modules are added before and after the bar code to generate a quiet zone. <b>Hint:</b> This function isn't fully implemented up to now.

## 5.2.9 Callback functions for user-defined drawing routines

<b>BCDrawCB (t_BarCode* const pBarCode, HDC hDC, RECT* pRect, fn_DrawBar fnDrawBar, fn_DrawText fnDrawText, LPVOID pData)</b> <b>pBarCode:</b> Ptr to bar code; <b>hDC:</b> Handle of Device Context; <b>pRect:</b> Ptr to bounding rectangle (a standard windows structure); <b>fnDrawBar</b> callback -> drawing bars; <b>fnDrawText</b> callback -> drawing text; <b>pData</b> custom data to pass to call back functions; <b>Returns:</b> <i>ERRCODE</i> (ErrOK if OK) typedef <i>ERRCODE</i> (CALLBACK *fn_DrawBar) (VOID*, HDC, RECT*); typedef <i>ERRCODE</i> (CALLBACK *fn_DrawText) (VOID*, HDC, LPLOGFONT, INT, INT, UINT, LPCTSTR, INT);	This is the callback version of the BCDraw function. The function has to look like the declarations on the left side. fn_DrawBar is used for drawing the bars, fn_DrawText is used for drawing the text. Sample code is available: <a href="mailto:support@tec-it.com">support@tec-it.com</a>
--	---

<p><b>BCSetFuncDrawRow (t_BarCode* const pBarCode, fn_DrawRow fn)</b></p> <p><b>pBarCode:</b> Ptr to bar code; <b>fn:</b> function pointer;  <b>Returns:</b> ERRCODE</p>	<p>Sets a callback function for drawing a single row. If the callback function is set, it replaces the standard drawing routine. The function has to look like the declarations on the left side.</p> <p>fn_DrawRow is used for drawing one line.</p> <p>Sample code is available: <a href="mailto:support@tec-it.com">support@tec-it.com</a></p>
<p><b>BCGetFuncDrawRow (t_BarCode* const pBarCode);</b></p> <p><b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> fn_DrawRow</p>	<p>Returns the callback function that was set by BCSetFuncDawRow.</p>
<p><b>BCGetMetaData (t_BarCode* const pBarCode);</b></p> <p><b>pBarCode:</b> Ptr to bar code; <b>Returns:</b> LPCTSTR</p>	<p>With BCGetMetaData you get access to the meta data of the current row. The result of BCGetMetaData is only valid, when it is called in the scope of your callback function.</p>



## 6 Appendix

### 6.1 Bar Code Reference

#### 6.1.1 Enumeration and Default Settings

##### Table heading:

Enumeration: Enumeration of the bar code type, (\* = not supported or under construction)

Nr: Numbering of the bar code type (corresponds to the value defined by the enumerator)

Barcode: Name of the bar code with a short description.  
Shortcuts: N = only used for numeric characters („0“..„9“), A = for alphanumeric characters (text), S = supports additional special characters, P = suggested check digit calculation (partially preset), 7D = 7 digits utilizable data (including leading zeros, without check digit)

Print Ratio: Standard Print Ratio of the bar code predefined corresponding to the bar code type.

Ratio Format: Format of the Ratio string, helpful to understand the definition of the Print ratio.

Check Digit: Enumeration of the preselected check digit method for each bar code.

##### Ratio Format:

- xB (1B, 2B, ...) width of bars
- xS (1S, 2S, ...) width of gaps (or spaces)
- red text not supported or under construction

Enumeration	Nr	Bar Code Name	Print Ratio Default	Ratio Format (RatioHint)	Check Digit Enumeration
eBC_None	0	Not a valid type	-----		-----
eBC_Code11	1	Code 11	1:2.24:3.48:1:2.24	(1B:2B:3B:1S:2S)	eCDNone
eBC_2OF5	2	Code 2 of 5 (Standard)	1:3:4.5:1:3	(1B:2B:3B:1S:2S)	eCDNone
eBC_2OF5IL	3	Interleaved 2 of 5 Standard ... N, P=Mod10	1:3:1:3	(1B:2B:1S:2S)	eCDNone
eBC_2OF5IATA	4	Code 2 of 5 IATA	1:3:1	(1B:2B:1S)	eCDNone
eBC_2OF5M	5	Code 2 of 5 Matrix ... N, P=Mod10	1:3:4.5:1:3	(1B:2B:3B:1S:2S)	eCDNone
eBC_2OF5DL	6	Code 2 of 5 Data Logic	1:3:1:3	(1B:2B:1S:2S)	eCDNone
eBC_2OF5IND	7	Code 2 of 5 Industrial ... N, P=Mod10	1:3:1	(1B:2B:1S)	eCDNone
eBC_3OF9	8	Code 3 of 9 (Code 39) ... AS, P=Mod43	1:3:1:3	(1B:2B:1S:2S)	eCDNone

eBC_3OF9A	9	Code 3 of 9 (Code 39) ASCII ... AS, P=Mod43	1:3:1:3	(1B:2B:1S:2S)	eCDNone
eBC_EAN8	10	EAN8 ... N, 7D, P=EAN8	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN8
eBC_EAN8P2	11	EAN8 - 2 digits add on ... N, 7D + N, 2D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN8
eBC_EAN8P5	12	EAN8 - 5 digits add on ... N, 7D + N, 5D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN8
eBC_EAN13	13	EAN13 ... N, 12D, P=EAN13	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN13
eBC_EAN13P2	14	EAN13 - 2 digits add on ... N, 12D + N, 2D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN13
eBC_EAN13P5	15	EAN13 - 5 digits add on ... N, 12D + N, 5D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN13
eBC_EAN128	16	EAN128 (unterstützt AIS) ... AS, P=C128	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDCode128
eBC_UPC12	17	UPC 12 Digits ... N, 12D, P=UPCA	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCA
eBC_CodaBar2	18	CodaBar (2 width) ... NS	1:3:1:3	(1B:2B:1S:2S)	eCDNone
eBC_CodaBar18*	19	CodaBar (18 widths)	-----		----
eBC_Code128	20	Code128 ... AS, P=C128	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDCode128
eBC_DPLeit	21	Deutsche Post Leitcode	1:3:1:3	(1B:2B:1S:2S)	eCDDPLeit
eBC_DPIident	22	Deutsche Post Identcode	1:3:1:3	(1B:2B:1S:2S)	eCDDPIident
eBC_Code16K*	23	Code 16K	-----		----
eBC_49*	24	Code 49	-----		----
eBC_9OF3	25	Code 93 ... AS, P=Mod47	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCD2Mod47
eBC_UPC25	26	Identical to eBC_UPCA	-----		----
eBC_UPCD1*	27	UPCD1	-----		----
eBC_UPCD2*	28	UPCD2	-----		----
eBC_RSS14	29	RSS-14	1:2:3:4:5:6:7:8:9:1:2:3:4 :5:6:7:8:9	(1B:2B:3B:4B:5B:6B:7B:8B: 9B:1S:2S:3S:4S:5S:6S:7S:8 S:9S)	eCDNone
eBC_RSSLtd*	30	RSS Limited	-----		----
eBC_RSSExp*	31	RSS Expanded	-----		----
eBC_UPCSCC*	32	UPCD6	-----		----
eBC_UCC128	33	UCC128 (= EAN128)	-----		----
eBC_UPCA	34	UPC A ... N, 11D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCA

eBC_UPCAP2	35	UPC A – 2 digit add on ... N, 11D + N, 2D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCA
eBC_UPCAP5	36	UPC A – 5 digit add on ... N, 11D + N, 5D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCA
eBC_UPCE	37	UPC E ... N, 7D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCE
eBC_UPCEP2	38	UPC E – 2 digit add on ... N, 7D + N, 2D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCE
eBC_UPCEP5	39	UPC E – 5 digit add on ... N, 7D + N, 5D	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDUPCE
eBC_PostNet5	40	PostNet ZIP (5d.) ... N, 5D	1:1	(1B:1S)	eCDPostNet
eBC_PostNet6	41	PostNet ZIP (5d.+CD) ... N, 5D	1:1	(1B:1S)	eCDPostNet
eBC_PostNet8	42	PostNet ZIP (8d.) ... N, 8D	1:1	(1B:1S)	eCDNone
eBC_PostNet10	43	PostNet ZIP+4 (5d.+4d.+CD) ... N, 9D	1:1	(1B:1S)	eCDPostNet
eBC_PostNet11	44	PostNet DPBC (5d.+4d.+2d.) ... N, 11D	1:1	(1B:1S)	eCDPostNet
eBC_PostNet12	45	PostNet DPBC (5d.+4d.+2d.+CD) ... N, 11D	1:1	(1B:1S)	eCDPostNet
eBC_Plessey	46	Plessey Code ... N, P=Pless	1:2:1:2	(1B:2B:1S:2S)	eCDPlessey
eBC_MSI	47	MSI Code ... N, P=MSI1	1:2:1:2	(1B:2B:1S:2S)	eCDMSI1
eBC_SSCC18	48	SSCC18	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S )	eCDNone
eBC_FIM*	49	FIM	-----		----
eBC_LOGMARS	50	LOGMARS ... AS (Mil- Std-1189B: P=Mod43)	1:3:1:3	(1B:2B:1S:2S)	eCDNone
eBC_Pharma1	51	Pharmacode One-Track	1:3:2:4:2:3	1B:2B:1C:2C:1S:2S	eCDNone
eBC_PZN	52	Pharmazentralnummer	1:2.5:1:2.5	(1B:2B:1S:2S)	eCDPZN
eBC_Pharma2	53	Pharmacode Two-Track	1:1	1B:1S	eCDNone
eBC_GP*	54	GP	-----		----
eBC_PDF417	55	PDF417 ... 2D Barcode, AS	1:2:3:4:5:6:7:8: 1:2:3:4:5:6	(1B:2B:3B:4B:5B:6B:7B:8B: 1S:2S:3S:4S:5S:6S)	eCDNone
eBC_PDF417Trunc	56	PDF417 Truncated ... 2D Barcode, AS	1:2:3:4:5:6:7:8: 1:2:3:4:5:6	(1B:2B:3B:4B:5B:6B:7B:8B: 1S:2S:3S:4S:5S:6S)	eCDNone
eBC_MAXICODE	57	MaxiCode ... 2D- Barcode, AS	1:1		eCDNone
eBC_QRCODE	58	QR-Code	1:1		eCDNone

eBC_Code128A	59	Code128 (CharSet A) ... AS	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDCode128
eBC_Code128B	60	Code128 (CharSet B) ... ASCII, AS	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDCode128
eBC_Code128C	61	Code128 (CharSet C) ... AS	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDCode128
eBC_9OF3A	62	Code 93 Ascii ... AS. P=M47	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCD2Mod47
eBC_AusPostCustom	63	Australian Post Standard Customer	1:1	(1B:1S)	eCDNone
eBC_AusPostCustom2	64	Australian Post Customer 2	1:1	(1B:1S)	eCDNone
eBC_AusPostCustom3	65	Australian Post Customer 3	1:1	(1B:1S)	eCDNone
eBC_AusPostReplyPaid	66	Australian Post Reply Paid	1:1	(1B:1S)	eCDNone
eBC_AusPostRouting	67	Australian Post Routing	1:1	(1B:1S)	eCDNone
eBC_AusPostRedirect	68	Australian Post Redirection	1:1	(1B:1S)	eCDNone
eBC_ISBN	69	ISBN Code (=EAN13P5)	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDEAN13
eBC_RM4SCC	70	Royal Mail 4 State (RM4SCC)	1:1	(1B:1S)	ECDNone
eBC_DataMatrix	71	Data Matrix	1:1		ECDNone
eBC_EAN14	72	EAN-14	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDNone
eBC_CODABLOCK_F	74	Codablock-F	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	eCDNone
eBC_NVE18	75	NVE-18	1:2:3:4:1:2:3:4	(1B:2B:3B:4B:1S:2S:3S:4S)	ECDNone
eBC_JapanesePostal	76	Japanese Postal... NAS, P=Mod19	1:1	(1B:1S)	eCDNone
eBC_KoreanPostalAuth	77	Korean Postal Authority... N, 6D, P=Mod10	1:3:4	(1B:1S:2S)	eCDMod10Kor
eBC_RSS14Trunc	78	RSS-14 Truncated	1:2:3:4:5:6:7:8:9:1:2:3:4:5:6:7:8:9	("1B:2B:3B:4B:5B:6B:7B:8B:9B:1S:2S:3S:4S:5S:6S:7S:8S:9S")	eCDNone
eBC_RSS14Stacked	79	RSS-14 Stacked	1:2:3:4:5:6:7:8:9:1:2:3:4:5:6:7:8:9	("1B:2B:3B:4B:5B:6B:7B:8B:9B:1S:2S:3S:4S:5S:6S:7S:8S:9S")	eCDNone
eBC_RSS14StackedOmni	80	RSS-14 Stacked Omnidir	1:2:3:4:5:6:7:8:9:1:2:3:4:5:6:7:8:9	("1B:2B:3B:4B:5B:6B:7B:8B:9B:1S:2S:3S:4S:5S:6S:7S:8S:9S")	eCDNone
eBC_RSSExpStacked	81	RSS Expanded Stacked	1:2:3:4:5:6:7:8:9:1:2:3:4:5:6:7:8:9	("1B:2B:3B:4B:5B:6B:7B:8B:9B:1S:2S:3S:4S:5S:6S:7S:8S:9S")	eCDNone

eBC_Planet12	82	PLANET 12 digit	1:1	("1B:1S")	eCDNone
eBC_Planet14	83	PLANET 14 digit	1:1	("1B:1S")	eCDNone

## 6.1.2 Related Bar Code Symbolologies

- [Code 128](#) is the basis for these specifications: USS Code 128, UCC-128, ISBT-128, EAN-128, EAN-14, SSCC-18 und SCC-14.
- ITF-14 is based upon [Interleaved 2 of 5 Standard](#), but with 14 digits.
- ISBN is based upon EAN13P5.

## 6.2 Parameter

### 6.2.1 Check Digit Enumeration

The method for the check digit(s) calculation depends on the respective bar code type. In order to make the ActiveX Control as user-friendly as possible, a standard method for each bar code type is supplied.

The bar code types EAN 8/13, UPC A/E and Postnet 6/10/12 are to mention particularly. When selecting the standard check digit the input can take place with and without check digit. In the latter case the check digit is calculated automatically and added.

Example EAN13: 12 digits input (= utilizable data), the 13<sup>th</sup> Digit (= check digit) is added automatically.

Enumeration	Nr	Check digit calculation method
ECDNone	0	No check digit calculation, no check digit added
ECDStandard	1	The preset method of each bar code type will be used (that means also, that for some types no check digit is applied)
ECDMod10	2	Modulo 10
ECDMod43	3	Modulo 43 (suggested for Code39 and Logmars, consist of 1 digit)
eCD2Mod47	4	Modulo 47 (2 check digits)
ECDDPLeit	5	Methode for Deutsche Post Leitcode
ECDDPIIdent	6	Method for Deutsche Post Identcode
eCD1Code11	7	Method for Code 11 (1 check digit)
eCD2Code11	8	Method for Code 11 (2 check digits)
eCDPostnet	9	Method for USPS Postnet
eCDMSI1	10	Method for MSI (1 digit)
eCDMSI2	11	Method for MSI (2 digits)
eCDPlessey	12	Method for Plessey

eCDEAN8	13	Method for EAN 8
eCDEAN13	14	Method for EAN 13
eCDUPCA	15	Method for UPC A
eCDUPCE	16	Method for UPC E
eCDEAN128	17	Method for EAN 128
eCDCode128	18	Method for Code 128
ECDRM4SCC	19	Method for Royal Mail 4 State
eCDMod10Kor	20	Modulo 10 for Korean PA bar code
eCDPZN	21	Mod-11 (PZN)
eCDMod11W7	22	Mod-11 (W=7)
eCDEAN14	23	EAN 14
eCDMod10Kor	24	Mod-10 (Korean Postal)
eCDMod10Pla	25	Mod-10 (PLANET)

### 6.2.2 Ratio, RatioHint (Ratio Format)

The *Print Ratio* is the relationship between the widths of the bars and gaps of a bar code. The ratio format (evident in the property *RatioHint*) depends on the selected type of bar code and shows, how many different bar- and gap-widths are used. The width of a bar (or gap) is calculated using the indicated Print Ratio and the (calculated or specified) module width (depending on the actual amount of utilizable data).

If no Ratio is indicated, the standard Ratio (refer to [Barcode Overview, Enumerators](#)) is used

(1B:1S)	1 bar, 1 gap
(1B:2B:1S:2S)	2 bars (1B ... narrow, 2B ... wide), 2 gaps
(1B:2B:3B:1S:2S)	3 bars, 2 gaps

Examples:

RatioHint (Format)	Ratio	Result
(1B:1S)	1:1	Same width is used for bars and spaces (gaps)
(1B:2B:1S:2S)	1:2:1:2.4	The print ratio of the wide bars (2B) to the small bars (1B) is specified as 2 : 1 The print ratio of the wide spaces (2S) to the small spaces (1S) is specified as 2.4 : 1
(1B:2B:3B:1S:2S)	1:2.5:4:1:2	There are three bar widths (1B, 2B, 3B) with the ratio 4 : 2.5 : 1 and 2 space widths with the ratio 2 : 1

### 6.2.3 Format

Format is used for formatting the bar code data prior to printing it (please do not mix up the *Format* with the *Ratio Format*).

Placeholders in the format string can be mixed with constant data characters to build a final bar code data string. Also control characters are supported. With this feature it's possible to:

- Select subsets in Code 128, EAN 128 and UCC 128 (even within the code!)
- Select the required start/stop character for CODABAR
- Change the position of the check digit (for special applications only)
- Set the values of Date, Preamble, Service Class, postal and country code directly in the barcode data (in conjunction with special Esc-sequences)

The placeholders are as follows:

Placeholder character	Description
#	Stands for the next data character
&	Stands for all remaining data characters
^	Stands for the next check digit (use only if check digits will be computed!)
A	Switch to Subset A (used in: Code 128, EAN 128, UCC 128)
B	Switch to Subset B (used in: Code 128, EAN 128, UCC 128)
C	Switch to Subset C (used in: Code 128, EAN 128, UCC 128)
A	Start- or stop character A (only in: CODABAR)
B	Start- or stop character B (only in: CODABAR)
C	Start- or stop character C (only in: CODABAR)
D	Start- or stop character D (only in: CODABAR)
S	Only for MaxiCode: enables setting the values of Date, Preamble, Service Class, Postal- and Country- Code directly in the barcode data (in conjunction with predefined Esc-sequences – refer to <a href="#">Setting SCM parameters</a> )
J	For Japanese Postal Code can be an automatically conversion of the Address B data field, that means that Japanese characters will be changed into ASCII characters. Look at <a href="#">#Japanese Postal (Customer) Code</a> .

Examples:

Data input	Barcode type	Format string	Final bar code data used as content
123	Irrelevant		123
123	Irrelevant	5&	5123
123	Irrelevant	&6	1236
123	Irrelevant	Q#w#e#	q1w2e3
123	Irrelevant	#q&	1q23
123	Irrelevant	&^	123c
123	Irrelevant	^&	c123
Hello	Code 128	A&	Hello
Hello	Code 128	A##B&	Hello
Hello4711	Code 128	A##B&	Hello4711
Hello4711	Code 128	A##B###C&	Hello4711

red characters    represented in Subset A

gray characters    represented in Subset B

green characters    represented in Subset C

c                    presents the place of the check digit

## 6.2.4 ESC Sequences and Control Characters

If you want to use non-printable or special characters in your bar code, you have to use “Escape Sequences”. They always start with a backslash ('\') followed by the sequence. You can use them also for encoding binary data (Bytes) into your bar code (if the symbology you are using offers this feature – e. g. PDF417 or Data Matrix).

Implemented Escape Sequences:

Esc-Sequence	Description
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab



\\	The Backslash \ itself
\0ooo	ASCII-character in octal notation
\ddd	ASCII-character in decimal notation
\xhh	ASCII-character in hexadecimal notation
\F	FNC1 or Gs (\x1d), used in UCC/EAN codes as field separator
\E	ECI ( <b>E</b> xtended <b>C</b> haracter <b>I</b> nterpretation), used in 2D codes like MaxiCode, Data Matrix and QR Code. Is used for switching between various code pages (multiple character sets) – contact us to get further information.
\EB, \EE	special ECI identifiers for nesting ECIs. \EB ( <b>E</b> CI <b>B</b> egin) opens a nesting level, \EE ( <b>E</b> CI <b>E</b> nd) closes it. Used in QR Code
\G	GLI ( <b>G</b> lobal <b>L</b> anguage <b>I</b> dentifier), similar to ECI, but only used in PDF417.

ooo     octal digits (0..7)

ddd     decimal digits (0..9)

hh      hexadecimal digits (0..F)

Following table shows a list of control characters, their escape sequences and in which symbology they may be used. The usage of escape sequences is code dependent and differs in different symbolgies. It is recommended to use named escape sequences if they are available (e.g. \F instead of FNC1 or Gs – dependent on the used symbology).

Control character	Escape sequence	Barcode type(s)
FNC1	\210	Code 128, EAN128, UCC128, 2D Codes
FNC2	\211	Code 128, EAN128, UCC128
FNC3	\212	Code 128, EAN128, UCC128
FNC4	\213	Code 128, EAN128, UCC128
DC1	\x11	Code93, Code93Ext
DC2	\x12	Code93, Code93Ext
DC3	\x13	Code93, Code93Ext
DC4	\x14	Code93, Code93Ext
Rs	\x1E	MaxiCode (Mode 3,4 SCM)
Gs	\x1D	MaxiCode (Mode 3,4 SCM)
Eot	\x04	MaxiCode (Mode 3,4 SCM)

You have to switch the object property "Esc Sequences" to "On" in order to use them. Please keep in mind, that when "translate escape sequences" is enabled, you cannot code a backslash ("\") directly. Use "\\" instead.

### 6.2.5 Application Identifier

An Application Identifier is a prefix used to identify the meaning and the format of the data that follows it (data field).

Als have been defined for identification, traceability data, dates, quantity, measurements, locations, and many other types of information. The data presented can be alphanumeric or numeric and with fixed and variable data length.

You don't have to encode the brackets, which defines the AI's. TBarCode ActiveX/DLL includes the brackets automatically.

With EAN128 you are able to run more than one AI together. If one field has a variable length and you don't use the whole number of characters you have to divide the data fields with the "\F" ESC sequence (you have to switch the object property "Esc Sequences" to "On" in order to encode "\F")

**Hint:** Don't use the "\F" after the last data field.

Please use the link below to get a list of all available AI's :

[http://www.ean.com.au/media/FILES/web\\_site/download\\_centre/eanapp\\_idents.pdf](http://www.ean.com.au/media/FILES/web_site/download_centre/eanapp_idents.pdf)

Examples:

- Batch number:

The AI 10 is used to encode a batch number. The format of this AI is n2 + an..20.

This means the AI (10) is followed by a data field with variable length (max. 20 characters).

Data:	10 + batch-number= 1012345678
Human readable text:	(10)12345678
Encoded data:	1012345678

- Using more AI's within one bar code:

There are two data fields encoded in one bar code. Following fields are used:

Batch number AI (10) - format: n2 + an..20

Item number AI (01) - format: n14

Data:	10 + batch-number + \F + 01 + item-number
Human readable text:	(10)12345678(01)12345678901234
Encoded data:	10123456780112345678901234

„\F“ is used because the batch number (max. 20 characters) only needs 8 characters.

## 6.3 MaxiCode

### 6.3.1 Extended Documentation

For MaxiCode we provide an extended documentation.

Download it from: <http://www.tec-it.com/Download> (Documentation)

### 6.3.2 Setting SCM parameters

The parameters for SCM (Structured Carrier Message - used for UPS purposes) can be set directly in the bar code data. This allows complete control of all necessary parameters e.g. from a database without additional coding.

The values for the properties *postal code*, *country code*, *service class*, *preamble* and *date* are extracted from the bar code data (text property) and the values of the belonging properties are overdriven.

Necessary settings:

The *Format* property must be set to "S" (switches extracting of SCM Data to "on").

The *Text* property should contain the whole text according to UPS standard including preamble, date, postal code, and country code and service class. Special characters and separators must be replaced by Esc-sequences (refer to [ESC Sequences](#)).

Control character	Escape sequence
Rs	\x1e
Gs	\x1d
Eot	\x04

## 6.4 Japanese Postal (Customer) Code

For this bar code type we support two different ways of data input (without and with data extraction from Japanese AddressB field).

### A) Direct Encoding Mode

Format Property = "" (default=empty)

Postal code = 2730102 (no hyphen '-')

Address B = 3-20-5B604

Bar code text = Postal code + Address B (no space between)

Bar code text = 27301023-20-5B604

Encoded data in the symbol: 27301023-20-5B604

### B) Japanese Extraction Mode

Format Property = "J" (=Japanese)

Postal code = 273-0102 (can contain '-')

Address B = 東3丁目 - 20 - 5 郵便・A&b□一ホB604号

Bar code text\* = Postal code + Address B

Bar code text\* = 273-0102 東3丁目 - 20 - 5 郵便・A&b□一ホB604号

Encoded data in the symbol = 27301023-20-5B604 (after internal conversion)

\* in TBarCode DLL provide input data in Japanese SHIFT JIS (MultiByte) char set (Codepage 932), use BCSetText function to pass MB string. In TBarCode ActiveX, select InterpretInputAs = Japanese Shift JIS

## Standard Dimensions

To meet the dimension specification:

- set the module width to 0.6mm  
DLL API: BCSetModWidth (pBC, "600")
- set the bounding rect in the draw function  
to a height of 3.6 mm
- switch off displayment of the human readable text

## 6.5 Korean Postal Authority Code

Example:

Post Number = 305-600

Barcode Text Property = 305600 (no hyphen, 6 digits)

Encoded data in the symbol = 0065036

The **check digit** (7<sup>th</sup> digit) is calculated automatically.

## 6.6 Error Codes

The following table lists all error codes that can be returned by TBarCode. The error codes and error messages are contained in the properties *LastErrorNo* and *LastError*.

Error code (0x=Hex)	Description
0x00000000	Err OK = No Error
0x80004001	Not implemented bar code type
0x80070057	Unsupported bar code or syntax error
0x8007000D	Wrong character
0x80090004	Wrong number of input characters
0x8007007A	Input string too long
0x80040140	Barcode does not fit into bounding rectangle
0x800710d2	No input characters

For example: In Visual Basic you can ask for a specific error code by using the following statement: „*IF objBarcode.LastErrorNo = &H80070057 ...*“.

## 6.7 Image Types

Applying the methods *SaveImage* and *ConvertToStream* to the object, the bar code can be converted to a bitmap or image format. Therefore the following image types with the corresponding compression options (parameter *nQuality*) are available. Please keep in mind that unreadable bar codes may be produced when creating a bitmap with low resolution.

### 6.7.1 Image Data Format

Image format	Enumeration (def. value)	Note
BMP	eIMBmp (0)	
EMF	eIMEmf (1)	In <i>ConvertToStream</i> and <i>ConvertToStreamEx</i> not supported
EPS (Bitmap)	eIMEps (2)	In <i>ConvertToStream</i> and <i>ConvertToStreamEx</i> not supported
GIF	eIMGif (3)	Not supported because of license terms – contact us if you need more information
JPG	eIMJpg (4)	
PCX	eIMPcx (5)	In <i>ConvertToStream</i> and <i>ConvertToStreamEx</i> not supported
PNG	eIMPng (6)	
TIF	eIMTif (7)	
EPS (Vector)	eIMPsVector (8)	In <i>ConvertToStream</i> and <i>ConvertToStreamEx</i> not supported

## 6.7.2 Compression Modes

Image format	Compression / nQuality	Remark
BMP	0..1, 0 = uncompressed, 1 = compressed	
EMF	Bitmap EPS: Not used Vector EPS: substitute w. device fonts: 0, 1	With vector EPS files you can choose between using Windows fonts (0) and using Postscript compatible fonts (1).
EPS	Not used	Not implemented
JPG	0..100, 0=highest compression, worst quality, 100 =lowest compression, best quality	Value of 100 suggested, especially for high data density
PCX	Not used	In <i>ConvertToStream</i> and <i>ConvertToStreamEx</i> not supported
PNG	PNGALLFILTERS (0) -> use best filter for each row (highest compression) PNGINTERLACE (1) -> Interlace filter PNGNOFILTER (2) -> no filter will be used (fastest runtime) PNGSUBFILTER (4) -> Difference filter with adjacent pixel PNGUPFILTER (6) -> Difference filter with pixel from the previous row PNGAVGFILTER (8) -> Average filter PNGPAETHFILTER (10) -> Paeth filter	To save an image in compressed mode and additional as interlaced file, you have to make a bit wise or operation with the defined constants (or simple adding the numbers).  Example: to save a file with maximum compression and interlaced, the quality parameter is calculated as follows:  PNGALLFILTERS   PNGINTERLACE
TIF	0.. No compression 1.. LZW (not supported) 2.. Packbits compression 3.. Group 3 4.. Group 4	LZW compression is not supported because of license terms – contact us if you need more information.

## 6.7.3 Resolution and Readability

Producing a bar code using Images or Bitmaps can decrease the bar code quality and cause problems with scanning - but for web applications it is inevitable to use bitmaps.

Background: the bar code must be converted from its internal resolution (high) to a graphic pixel resolution (low). In this process it can happen that the module width varies due to rounding errors or that bar code modules are truncated if the resolution is too small.

This is a general problem when using bitmap formats with a limited resolution (BMP, JPG, PNG) in contrast to vector based formats (e.g. EMF).

*For web applications: please download the ASP and PHP sample code from our sample download area (<http://www.tec-it.com/download>). This code will help you to produce optimized bar codes.*

The following workarounds are suitable to minimize graphical errors when reproducing the bar code through a bitmap:

#### General:

1. Enlarge the resolution – use significantly higher values for *XSize* and *YSize*. This should avoid readability and scanning problems in most cases.
2. Especially when using Code128 or other codes with high optical density you have to choose an essential higher resolution.
3. Don't use any compression reducing the picture quality.
4. Use multiple bar code symbols for higher data contents – try to reduce the optical density of the single codes symbols.
5. Use the new property "OptResolution" to adapt the module width to the pixel resolution.

#### Workarounds:

1. **Set the property "OptResolution" = true** – this can fix reading problems for low output resolutions.
2. **A module width set to a specific fixed value** can also prevent scanning problems - however not in each case. In order to represent the max. Data capacity it must be ensured that the bar code object is drawn wide enough (*XSize* must be large enough). In screen resolution (96dpi) 1 pixel would be 0,2646 mm large. With "obj.ModulWidth = 265" the module width is adjusted to approximately 1 pixel (in 1/1000 mm).  
If possible at least 2 pixels should be used for representing a single module (obj.ModulWidth = 529). These values are OK, if the device context is adjusted for 96 dpi screen resolution (like in most cases).
3. **CountModules:** the barcode property „CountModules“ helps substantially to cope with readability problems using bitmaps. It gives you the number of bar code modules needed to encode the utilizable data. A module is the smallest graphical element (segment) of which a bar code consists of. If the width of the bar code (= *XSize* [pixels]) output image equals the number of bar code modules (=CountModules), then many problems fall away: no matrix effects, no rounding errors, no module width variations, etc.  
This procedure works for **Web Applications** (ConvertToStream method) as well as for storing image files (SaveImage). With such a setting it is achieved that the module width becomes exactly 1 pixel wide.  
If you need wider bar codes, then select a multiple of CountModules, e.g. *XSize* = Barcode.CountModules \* 2 for the width of the bar code image. This procedure guarantees optimal readability for all bar codes, with integer based print ratios (referring to the reference table actually all but Code11). This workaround is ideal in particular if the bar code is not bound to fixed values in its dimensions.
4. **Within HTML:** in order to increase printing resolution you can **produce the barcode with twice or triple resolution** as used in the browser window. Within the HTML-code at client side it would look like ``. However, the barcode is produced with twice resolution like: *XSize*=500 Px and *YSize*=120 Px (using SaveImage or ConvertToStream). Use a larger font size (property font) to make the text look normal. To avoid big file sizes you could double only the horizontal but not the vertical resolution. A distorted font can be avoided: switch the font in the barcode off and print the text separately using HTML.
5. **Possible combinations:** Workaround 1 should solve most problems. But Workaround 2 and 4 for increasing the bitmap resolution is possible (e.g. ModuleWidth = 0.2 mm \* 5 = 1000, *XSize* = [width in Pixels at max. data content] \* 5, *XRes* = [127dpi] \* 5). Workaround 2 and 3 cannot be combined.

### Printing Resolution:

With BMP files the resolution of the image (dpi) cannot be adjusted, as default a value of 72 dpi is used (the values of the parameters nXRes, nYRes used with SaveImage / ConvertToStream are ignored).

With JPG (and EMF) a resolution for printing can be specified - the size of the printed bar code can be adapted thereby. The resolution should be selected in such a way that the bar code remains readable by the scanner. When using a module width of 1 pixel the highest printing resolution should be 127 dpi (module width = 0.2 mm). Using 2 pixels → 254 dpi, etc. Also the maximum resolution of the printing device should be considered. With Web applications consider the fact that the browser ignores the resolution (dpi) of the JPG image but adapts the bar code size according to the screen representation.

### PDF417 and images (e. g. used in web-applications):

To get a proper image (without matrix effects) you have to calculate the XSize and YSize of the image by using the properties CountModules and CountRows. To get the XSize use the value of CountModules divided by the value of CountRows. Use a multiple of the value of CountRows to get a proper YSize.

VB-Sample for calculating the XSize and YSize values used for SaveImage or ConvertToStream (for PDF417):

```
TBarcode.Text = BarCodeData
Numrows = TBarcode.CountRows
nXSize = TBarcode.CountModules / Numrows
nYSize = Numrows * 5
```



## 7 FAQ's

If these FAQ's don't fit to your problem - check our website <http://www.tec-it.com/FAQ> or contact our support team directly: <mailto:barcode@tec-it.com>.

### 7.1.1 How to add the leading and trailing '\*' for Code 39?

No action is required. The '\*'s are added automatically to the barcode.

### 7.1.2 How to add the check digit to Code 39?

Simply select Modulo 43 as Check-Digit Method. The automatically computed check-digit is appended at the end of the bar code.

### 7.1.3 How can I add bar codes to a mail merge document?

In our samples download area we have a Word Macro, which provides the functionality for adding bar codes to a mail merge document.

The Word Macro "BarcodeSeries" is contained in a dot file, which must be included to the mail merge document.

You can download the Macro with a detailed description and a sample, which shows how it works.

Download:

<http://www.tec-it.com/download/samples>

Designing the mail merge template:

You need a bar code object in the template (insert TBarCode ActiveX and adjust the bar code parameters as desired for your bar code). In the mail merge template, the bar code data must be placed in a paragraph directly before the bar code object.

Start the mail merge:

The mail merge target must be a new document (not the printer).

Process all bar codes: After merging into a new document, start the Word Macro "BarcodeSeries". It copies the text paragraph before the TBarCode Object into the bar code object.

Then print.

Hint: Macros must be enabled in Word 2000 (check out [Tools] [Macro] [Security...]).

#### 7.1.4 How can I determine the number of PDF417 Code Words?

After BCCreate(..)

```
long cm = BCGetCountModules(pbarCode);  
long cr = BCGetCountRows(pbarCode);  
  
long width = cm/cr;  
long height = cr * 5; // not smaller than cr * 3 !!!
```

The number of non-data modules per row depends on whether the PDF417 symbol is truncated or not.

```
long INumNonDataModulesPerRow = 69; //(eBCTYPE == eBC_PDF417) ? 69 : 52;
```

Determine the number of data code words to encode. This is only a rough estimate due because there is no TBarcode DLL function that returns this information. The estimated will be based on the total number of code words and total number of rows, as indicated by the TBarcode DLL.

```
long INumDataCodeWords = (cm - cr*INumNonDataModulesPerRow)/17;
```

#### 7.1.5 How to add the leading and trailing 'A' (or B or C or D) for CODABAR?

Enter A&A in the Format string (& is a placeholder for the barcode raw data).

#### 7.1.6 How can I change the module width to 15 mils in my web application (ASP) ?

Our component allows setting the module width in 0.001 mms steps (e.g. 15 mils = 0.381 mm) but during conversion of the bar code to an image file (ConvertToStream) it will be rasterized in a raster of 96 dpi. After this process one module (smallest bar element) can only be 1 or a multiple of 1 Pixel:

that is  $1/96 \text{ dpi} = 0.0104 = 10 \text{ mils}$  (or twice/triple/... of 10 mils). It can't be 15 mils because 1 and a half pixel is not possible in an image raster format.

For web applications where the bar code is displayed in the browser we have two parameters, which influence size and quality.

- The width [Pixels] of the created bar code image influences the quality (because it influences the width of the modules).
- The width [Pixels] of the image tag in the browser influences the displayed and printed width.

### Example:

In your ASP code you could create the barcode with 600 pixel width:

- `binStream = ConvertToStream (600, 150 ...)`

But you can display it in the browser window 196 Pixels wide.

- `" width="196" height="48">`

→ The created image (600 Px) will be shrunk down to 196 Pixels by the browser.

It is not a good idea to create the bar code always the same width (like in our example 600 Pixels) if the data to be encoded can change. You may need more or less modules (bars...) in a bar code with varying data content.

→ So in our ASP sample code we suggest to always calculate the number of modules before to assign the width.

### Solution:

The only way to get 15 mils module width would be to adjust the symbol size by the img tag in html.

- Use our sample code, which uses the CountModules method for getting the correct width in Pixels for creating the bar code.
- Then “fine-tune” the bar code size with the image tag width parameter. By fine-tuning the bar code size using the image tag you also influence the module width (smallest bar width) of the symbol.

We can help you getting the correct width for the image tag if you send us sample data, desired bar code type and the valid module width range (mailto:support@tec-it.com). You could find out the optimum width by your own using our Barcode Studio Software, which is downloadable from <http://www.tecit.com>

## **7.1.7 How to use a specific subset in Code 128?**

Use the corresponding bar code types Code128A, 128B or 128C. The whole code will then be generated in the corresponding subset. If you want to change the subset within the bar code enter A or B or C in the format string (please refer to References/ [Format](#)).

## **7.1.8 How to use the compressed mode of Code 128?**

Clear the *Format* string. If this field is empty (default) the code will be generated in the subsets that will produce the shortest representation.

### 7.1.9 How can I get a PDF417 symbol with standard aspect ratio (3:2) ?

1)

Set a Row:Col Ratio of 11:1

e.g.

Set Cols = 2

Set Rows = Cols \* 11

2)

Make a constant ratio of row height: module width

→ Set a row height: module width ratio of 3:1 (default)

Set module width = 500 (0.5 mm constant value)

Set PDF row height = 1500 (1.5)

Result: You will get a standard PDF417 symbol with standard aspect ratio of 3:2

### 7.1.10 How to license the product from within my application?

Assume you got the following license key:

Product ID:	13 (1D Barcodes)
Licensee:	Barcode Inc.
License Kind:	2 (Site)
Licenses:	1
License-Key:	12A3B5C6

Now implement a call to the method *LicenceMe* before you draw a bar code (we suppose at startup of your application).

Example in Visual Basic (enumerators are used):

```
TbarCode41.LicenseMe "Barcode Inc.", eLicKindSite, 1, "12A3B5C6",  
eLicProd1D
```

Syntax: [object name].LicenseMe ["Name of licensee"], [license type (1=single,2=site or 3=dev)], [number of licenses (=1)],  
["license key"], [product id (13=1D, 14=2D)]

Product ID is depending on program version (see your license email).

### 7.1.11 How to use Application Identifiers (AI's) in Code 128 or EAN128?

Check *Translate Escape Sequences* (value must set to *True*) and insert the following escape codes into the data text (see also [ESC Sequences](#)):

Control character	Escape sequence	Bar code type(s)
FNC1	\210	Code 128, EAN128, UCC128
FNC2	\211	Code 128, EAN128, UCC128
FNC3	\212	Code 128, EAN128, UCC128
FNC4	\213	Code 128, EAN128, UCC128

FNC1 is the function number character and is used as field separator in connection with variable length data fields.

Note: Application Identifiers (AI) are entered unformatted without parentheses, but the "human readable text" will appear formatted (incl. parentheses).

FNC1 must be appended to all application identifiers with variable length when the maximal length of the field in question was not filled completely and when it was not the last AI in the bar code. Do not append an FNC1 if there is a fixed length AI or it is the last AI in bar code. Please note that FNC1 must be inserted by your application (TBarCode can't determine correct placements of required FNC1-characters).

### 7.1.12 How can I set the Module Width to a constant value?

Per default the bar code width adapts automatically to the object width (= to the dimension of the bounding rectangle).

By setting the **object property "ModuleWidth"** to a specific value, the resulting bar code width depends on the amount of encoded data and – on the module width (= narrowest bar/space width). Please note, that not the dimension of the bounding rectangle (the object dimensions) is changing, the bar code size within the bounding rectangle is changing.

Example: Code128 needs a minimum module width of 0,19 mm, i.e. we select 0.25mm as module width. The unit of the "ModuleWidth" is set in 1/1000 mm. In Visual Basic we use the following instruction: `obj.ModuleWidth = "250"`

The dimensions of the bar code object are defined by your settings in design mode of your application or by the properties width and height. Keep in mind to choose an object size that ensures that the bar code is not truncated (clipped), especially when using the bar code on a form. The dimension of the bounding rectangle must be set according to the largest data content that can occur. For printing: in order to examine the occurrence of clipping for each printing cycle, you can set the object property `MustFit` to true (VB syntax: `obj.MustFit = TRUE`). As soon as the bar code within the object becomes larger than the bounding rectangle, a standard exception is raised (Handling in VB: `On Error Goto ErrHandler`).

If you would like to **vary the external dimensions** of the barcode object according to the width of the bar code (with constant module width), you can determine the dimensions of the bar code using the property **"BCWidthHdc"**. This is done in the following VB example:

```
BCObj.ModuleWidth = "250"
DoEvents
Printer.ScaleMode = vbPixels
Xsize = BCObj.BCWidthHdc (Printer.hDC, 1, 1, eMUPixel)
BCObj.BCDraw Printer.hDC, 500, 500, Xsize, Printer.ScaleY(20,
vbMillimeters, vbPixels)
```

In the example a fixed height of 20 mm is used - the x-dimension (width) is adapted to the current bar code. The value pair 1, 1 (for X,Y) is irrelevant since we don't know the final size of the bar code yet. If necessary, the starting point of 500, 500 [Pixels] could be converted with the ScaleX and ScaleY methods into absolute values [mm] in order to be independent of the actual printer resolution.

**Image-Files:** This workaround by reading the BCWidth property could also be used when producing Image-Files (BMP, JPG...). But we recommend other ways to do this by using the property „CountModules“ (look at section [Resolution and Readability...](#)).

#### **7.1.13 I am changing the font resolution (large fonts, 120dpi) and get a clipped off bar code with SaveImage?**

This can happen, if you set the ModulWidth property to a specific value. For image saving, the screen device context is used. This changing resolution of the screen influences the module width calculations.

If you don't set the module width to a specific value, you will see that the bar code will fit into the x and y parameters of SaveImage.

If you need a specific module width for your application (e.g. 3 pixels), you can calculate the output size of the image using the number of horizontal graphical modules (CountModules property) and then multiply the value of CountModules with 3 (to get 1 module = 3 Pixels wide). With this method, the image size will adapt automatically depending on number of horizontal bar code modules.

For 2D symbologies divide the value of CountModules through the number of data rows to get the correct X dimension value. Contact our support for help with 2D symbologies.

#### **7.1.14 I am using PrintForm in VB and the barcode is not readable**

If bar code objects in a form are drawn with (low) screen resolution (and not with printer resolution) – the barcode may become unreadable.

To get a workaround the module width of the bar code has to be adapted to the resolution of the screen:

- This can be achieved by using the CountModules property (see below).
- Or you use the new TBarCode4 property OptResolution = true

##### Using the CountModules Property:

A bar code consists of so called “modules” – small elements that can be either a bar or a gap (empty space). If you set the width of the bar code (the number of Pixels on the screen) exactly to the number of modules in the bar code, the resolution problem can be avoided. Therefore a bar-module is represented by a black pixel and a gap-module is represented by a white pixel.

To set the width of an object to a value in Pixel, you have to set the VB Scale-Mode of the form to 3 (Pixel)!

Sample VB Code for Code 128 (or other bar codes without a space around like Code 30, 2of5 IL,...)

```
` set the width of the TBarcode object according to the CountModules property
BCObject.text = barcode_data
BCObject.width = BCObject.CountModules
```

### Sample VB Code for PDF417

```
`If you use a PDF417 as the bar code symbology use this code:
BCObject.text = barcode_data
BCObject.width = BCObject.CountModules / BCObject.CountRows
BCObject.height = BCObject.CountRows * 5

Form1.PrintForm
```

Please consider: in our example the module width adapts to 1 Pixel automatically. If you set the module width to another value (e.g. by using the ModulWidth property) the solution above doesn't work! In this workaround the bar code width also depends on the amount of data.

## 7.1.15 What is the maximum data capacity of PDF417?

We have found in the PDF417 specification the following limits:

Maximal data characters without error correction (PDF417\_ECLevel = 0):

Numerical data only:	2710 digits
Bytes:	1108 (also used for control characters to switch to lowercase letters)
Text characters:	1850 (only uppercase letters used [A..Z])

If you mix the character types (as shown above) the maximum data capacity cannot be predicted exactly (due to internal compression and character set switching - this is by design of PDF417).

If you use a combination of digits and text (lower & uppercase letters) the maximum data capacity would be about 1100 to 1200 characters - but this can vary due to your input data. If you want to encode large data amounts we recommend using multiple symbols (or to use only capital letters).

## 7.1.16 When using ESC-Sequences they are not encoded (and the scanners signals an error) - why?

Escape sequences are several characters starting with „\“ and enable encoding of a special character. They are translated into this special character when the bar code is created.

*Escape sequences are only translated if this translation is explicit activated.*

Otherwise you find the original character sequence (e.g. the two characters „\“ and „t“) in your data and not the special character you wanted to encode (e.g. TAB). Some scanners don't „like“ the backslash – so you might also become reading problems.

To activate the Translation of Escape sequences

- Within the property pages of the TBarCode you have to mark „Translate Esc-Sequences“.
- Or (for programmers): within the API of the ActiveX object there is the property „EscapeSequences“ that has to be set to „True“

More about that topic: [ESC Sequences](#)

### 7.1.17 How can I save the MaxiCode symbol with a higher image resolution?

You have to set the ModulWidth property to a higher value (as it is by default) – and increase the values for Xsize / Ysize at saving.

VB Sample for saving a MaxiCode symbol with a higher image resolution:

```

FlName = "Maxicode.bmp"
TBarCode.Text = BarCodeText
TBarCode.Barcode = eBC_MAXICODE

'Use nRatio to enlarge or reduce image size
nRatio = 2

TBarCode.ModulWidth = 1500 * nRatio
DoEvents
YSize = 208 * nRatio
XSize = 215 * nRatio

TBarCode.SaveImage App.Path & "\" & FlName, eIMBmp, XSize, YSize,
72, 72

'Hint: When printing we recommend to use the standardized size of
MaxiCode
'Height: ~24.4 mms; Width: ~25,2 mms

```

#### Background:

If you don't print the barcode by the BCDraw-Method but by using a BMP-bitmap, you have to increase the x/y-dimension of the image. BMP uses a fixed resolution of 72 dpi. Increasing the dimensions to get more Pixels results in a bigger MaxiCode symbol at printout. But you have to pay attention to the standardized size of MaxiCode. You have to scale (to "shrink") the BMP bitmap to the proper size at printout. If you save MaxiCode as JPG-File you are able to adapt the horizontal and vertical dpi-resolution of the file to get the proper size.

### 7.1.18 How to draw bar code to image or to printer device context

Below is a sample code which is taken from the DLL sample.

The ActiveX also supports the BCDraw method, for drawing to a device context.

#### **Draw to image:**



```
imgrect.Left      := Image1.ClientRect.Left;
imgrect.Right     := Image1.ClientRect.Right;
imgrect.Top       := Image1.ClientRect.Top;
imgrect.Bottom    := Image1.ClientRect.Bottom;
eCode:= Ax.BCDraw (pBC, Image1.Canvas.Handle, imgrect);
Image1.Refresh();
```

#### **Print out:**

```
Prntr              := Printer;
Prntr.BeginDoc;

pRect.Left         := trunc (Prntr.PageWidth / 3);
pRect.Top          := trunc (Prntr.PageHeight / 4);
pRect.Right        := trunc (Prntr.PageWidth - pRect.Left);
pRect.Bottom       := trunc (pRect.Top + (Prntr.PageHeight / 5));

// Draw bar code to the coordinates of pRect (in Pixel)
eCode := Ax.BCDraw (pBC, Prntr.Handle, pRect);

Prntr.Refresh();

Prntr.EndDoc;
```

### **7.1.19 I always get an error when calling SavelImage method?**

If SavelImage doesn't work, please check, whether you have installed VIC32.dll. Otherwise it can be that you don't have write permissions to the directory you want to write to (e.g. in web application).

### **7.1.20 When using SavelImage, my barcode reader can't scan the symbol / image!**

The bars and spaces in the symbol must have precise widths. By saving the symbol to a bitmap image, the widths are adapted to the matrix of the bitmap - depending on its resolution less or more deviations occur. The lower the resolution, the more differences in width of bars/spaces appear.

#### Suggested workarounds:

- Saving the bitmap with higher amount of pixels is a good method to avoid such problems. Save the image with twice or triple of normal width/height and then shrink it back for printing (if that's possible in your application). Please read also the chapter [Resolution and Readability](#).
- Also using the OptResolution Property of TBarCode4 (set it = true) can avoid such problems. It adapts the module width automatically to the next lower pixel resolution. That can lead to shrinking of the symbol. Make it wide enough to avoid too small bar codes.
- Set the width of the bar code corresponding to the number of modules in the symbol (CountModules property).

#### Background: module width and the bitmap resolution:

If you *save or print* the bar code using a resolution of 300 dpi, one pixel will be 0,003333 inch = 0,08466 mms wide. The module width should not go under 0.19 mms - so select a multiple of

0,08466 mms for the module width to get best results for scanners, e.g.  $3 * 0.08466 = 0,254$  mms for the module width. By selecting a multiple of the width of a pixel the tolerances in width for the bars and spaces are minimized.

- The module width can be set either with the ModulWidth property (ActiveX) or by BCSetModWidth (DLL) in Hi-Metric (1/1000 mms).
- Recommended: to set the Module Width indirect by the bar code size, you can use the [CountModules](#) method. If you set the bar code width to the number of modules (CountModules), you will get automatically a module width of 1 pixel. If you set the bar code width to 2 times of CountModules, you will get a module width of 2 pixels (and so on). This works for all bar codes with integer print ratios, e.g. Code128, Code39 (see [Reference Table](#)).

Code 11 / Code 2/5 Std: if you have [print ratios](#) that are not integer ratios, you must use a number of pixels that can represent all module widths within the print ratio (e.g. you have a print ratio of 1:1.5 → select 2 pixels as for the module width → 2 : 3 pixels = 1:1.5 print ratio = OK).

### 7.1.21 How can I use TBarCode within FoxPro?

Insert TBarCode into a Form:

- Select *[OLE Container]* from the Standard Toolbox
- After drawing the outlet of the OLE container, an "Insert object" dialog is opened. Select *TBarCode4* (Insert Control) and confirm with OK

If you want to print bar codes in reports or labels, you need a table with a specific column, in which the TBarCode ActiveX must be stored as OLE Control (data type *general*). This data field must be initialized (before use) with the data structure of a bar code object. For this purpose you need an instance of a TBarCode object as template (e.g. on a form, it may be invisible).

Example code for the initialization of the data fields with a bar code object:

```
FOR n = 1 TO 10
  INSERT INTO Table1 FROM MEMVAR
  APPEND GENERAL Table1.BC CLASS TBarCode4.TBarCode4
  WITH THISFORM.[Name of the TBarCode object instance]
    .CONTROLSOURCE = "Table1.BC"
    .REFRESH
    .barcode = 20 && Barcode-Type = Code128
    .TEXT = "000070000041"
    .printdatatext = .T.
    .BACKCOLOR = 16777215
    .FORECOLOR = 0 && 16711680 -> (blue)
    .REFRESH
  ENDWITH
NEXT
```

After this code has been executed, an OLE Picture/ActiveX Bound Control can be inserted into a Report or Label Form. In our sample, you have to specify "Table1.BC" as data source (field). Please

take a look at our FoxPro-Sample, which can be downloaded at <http://www.tec-it.com/Download> (Samples)!

### 7.1.22 How to speed up the generation and printing of QR-CODE for large numbers?

The following hints are also useful for other 2D symbologies (e.g. Data Matrix...)

You could:

- Set the QR-Code mask pattern to a constant value (setting could affect readability, make tests before you choose a value).
- Set the symbol size to a constant value (property "symbol version") if the symbol should have always the same size.
- Set the error correction level to "low" (if you don't have dirt, scratches or other possible surface detractions).
- Optimize database speed

Only relevant for programming:

- Each time, if a TBarCode property is changed by your program (e.g. text, barcode type, QR-code setting, . . . ) the bar code is internally recalculated - so if you set all properties each time you need a bar code, you need much more CPU time.
- Solution = set the configuration properties of the ActiveX Control only one time at startup of your program, and do only the changing of the text property for each bar code, so the bar code is only calculated one time.

### 7.1.23 What can I do to optimize PDF417 for sending through a desktop analogue FAX?

For dealing with resolution issues, you should download our Barcode Studio Software from <http://www.tec-it.com/download/>

- Fax works with 200 dpi, so enter 200 in the DPI input field.
- Then enter the text you want to encode and select PDF417
- Next make the symbol the same size as you use for printing.
- Now the "quality watch" shows you the tolerance / pixel aberration in per cent.

We know from bar code standards that up to 15% is OK and up to 28% (+/-2%) is decoding with most scanners. But more than 30% can make serious problems.

Now what you can do:

- Make the module width a multiple of 1 Pixel (but  $\geq 0.190\text{mm}$ ), e.g. make the module width =  $0.381\text{ mm} = 3\text{ Pixels}$ . You will see that the tolerance shows a maximum quality.
- Adapting the module width to the available pixel width (depending on dpi) is the best way to overcome resolution problems.

- For PDF417 make sure that the line or row height is not lower than 3 times the module width.

After you have set up these parameters with your bar code we suggest that you send test faxes and make test readings with your scanner in order to see if the reading quality increased.