TBarCode Library

Barcode Generation Library

Version 11

Developer Manual

9 January 2018

TEC-IT Datenverarbeitung GmbH
Hans-Wagner-Str. 6
A-4400 Steyr, Austria

t ++43 (0)7252 72720
f ++43 (0)7252 72720 77
office@tec-it.com
www.tec-it.com

1 Content

2 Disclaimer

The actual version of this product (document) is available as is. TEC-IT declines all warranties which go beyond applicable rights. The licensee (or reader) bears all risks that might take place during the use of the system (the documentation). TEC-IT and its contractual partners cannot be penalized for direct and indirect damages or losses (this includes non-restrictive, damages through loss of revenues, constriction in the exercise of business, loss of business information or any kind of commercial loss), which is caused by use or inability to use the product (documentation), although the possibility of such damage was pointed out by TEC-IT.

TEC-IT Datenverarbeitung GmbH
Hans-Wagner-Str. 6

A-4400 Austria
t.: +43 (0)7252 72720
f.: +43 (0)7252 72720 77
http://www.tec-it.com

3 Introduction

3.1 What is TBarCode SDK?

TBarCode SDK is a set of professional tools for the generation of barcodes. More than 100 different symbologies (linear barcodes, 2D barcodes and stacked barcode variants) can be printed or exported as graphics files. All industry formats are supported. The barcodes can be generated in the highest possible resolution and quality.

TBarCode SDK is available in several versions for different operating systems, applications and programming environments.

The following versions are included in the Windows setup:

TBarCode OCX	A Microsoft® ActiveX® compliant barcode control. It can be used with Microsoft® Office applications as well as by software developers.
TBarCode .NET	A .NET barcode library for software developers. It includes barcode controls for Windows Forms and ASP.NET 2.0.
TBarCode Library	A dynamically linked library (DLL) for software developers.

Additionally the following versions are available as separate downloads:

TBarCode/X	**TBarCode** software for Linux® and UNIX® platforms. **TBarCode/X** includes: ▪ command-line tools, ▪ filter scripts, and ▪ **TBarCode Library** for UNIX®.

3.2 What is TBarCode Library?

TBarCode Library is a software library for barcode generation. It is the core of all **TBarCode** products. The library is available for software developers to integrate barcode generation in custom applications. TEC-IT provides the **TBarCode Library** for a wide range of platforms:

- 32-bit Microsoft® Windows® platforms
- 64-bit Microsoft® Windows® platforms
- Microsoft® Windows® CE / Windows® Mobile platforms (available on request – please contact us with your platform details)
- Most Linux® and UNIX® platforms

The **TBarCode** setup for Windows® includes **TBarCode Library** as dynamically linked library (DLL). A static library is available on request.

TBarCode Library for Windows is often just called "**TBarCode DLL**.

3.3 Scope of this Document

This document explains how you can use the **TBarCode Library** in your own applications. The complete application programming interface (API) is described in the *TBarCode Library Developer Reference*.

3.4 Restrictions of the Demo Version

In the demo version the barcodes will be drawn with a demo-hint. That means that the word “Demo” or the phrase “www.tec-it.com” is drawn partially over the barcode (see Figure 6). The demo-hint does not influence the readability of the barcode in a negative way.

When barcodes are generated in PostScript® or PCL® format, an additional horizontal bar is drawn across the barcode. Like the other demo-hint this bar usually does not influence the readability of the barcode. Its sole purpose is to indicate that the barcodes were generated with a demo version of **TBarCode**.

- In special cases (e.g. very small or high-resolution barcodes) you may want to test the product without restrictions. To obtain a temporary license key contact sales@tec-it.com.
- For enabling the full-featured version (without the demo hints) you can obtain a license key from TEC-IT (http://www.tec-it.com/order/).
- For more information on licensing **TBarCode**, please refer to chapter 9, “How to License TBarCode”.

4 Installation

4.1 TBarCode for Windows

4.1.1 Requirements

The following operating systems are supported:

- Microsoft Windows 10
- Microsoft Windows 8
- Microsoft Windows 7
- Microsoft Windows Server 2012 and 2012 R2
- Microsoft Windows Server 2008 and 2008 R2

4.1.2 Download and Setup

Download **TBarCode SDK** from http://www.tec-it.com/Download ► *TBarCode SDK* and execute the setup application. Depending on your selection one or more of the following components are installed – along with documentations and sample applications:

- **TBarCode OCX**
  A barcode ActiveX control for Microsoft Office users and for software developers. User manual and API reference are included in the setup.
- **TBarCode Library**
  For software developers. Developer manual and API reference are included in the setup.
- **TBarCode .NET**
  For .NET developers. Developer manual and API reference are included in the setup.

► Administrative rights are required to install **TBarCode SDK.**

5 General

Please keep in mind that **TBarCode Library** is a software component. It is not an executable by its own. Read this document and check out the accompanying sample applications to learn how to embed **TBarCode** into your own application.

5.1 TBarCode Library

The dynamically linked library is named *TBarCode11.dll* and can be found in the directory *C:\Program Files\Common Files\TEC-IT\TBarCode\11.0.*

> ► Make sure that the **TBarCode Library** *(TBarCode11.dll)* is located in a path included in the *'PATH'* environment variable (or in the same directory as your application).

5.1.1 Additional Dependencies

TBarCode Library further uses the DLL *zlib1.dll*. This DLL file is optional: *zlib1.dll* is only required if compression of barcode data is applied.

If *zlib1.dll* is missing at runtime the **TBarCode Library** is still working, but data compression is disabled.

> ► Make sure that *zlib1.dll* is located in the same directory as *TBarCode11.dll*.

5.1.2 64-bit Systems

On 64-bit systems you **TEC-IT** recommends to install the 64-bit version of **TBarCode SDK**. The 64-bit setup contains both, 32-bit and 64- library. The name of the 64-bit library is *TBarCode11_x64.dll*. It is located in the directory *C:\Program Files\Common Files\TEC-IT\TBarCode\11.0*, whereas the 32-bit library can be found in *C:\Program Files (x86)\Common Files\TEC-IT\TBarCode\11.0*.

5.1.3 C/C++ Header Files

The DLL is delivered with C/C++ header files needed for embedding the DLL (the library) into your own application. These files are located in the folder *C:\Program Files\TEC-IT\TBarCode11\Include* (or in the folder you have chosen during setup).

Include the file *tbarcode.h* in your project in order to get full access to the DLL functions within C/C++:

```
#include "tbarcode.h"
```

Header files for miscellaneous programming environments (for example Delphi) are included in the appropriate source code samples. Several samples are installed with **TBarCode**; more samples are available on the TEC-IT web page http://www.tec-it.com.

5.1.4 Linking

TEC-IT provides the file *TBarCode11.lib*, which contains the entry points to the DLL interface. Link your application using this file. This file is located in the folder *C:\Program Files\TEC-IT\TBarCode11\Lib* (or in the folder you have chosen during setup).

The 64-bit version is called *TBarCode11_x64.lib*.

6 Using TBarCode Library

6.1 Important Functions

The basic function calls to produce a barcode are as follows (in the appropriate order).

- *BCLicenseMe()*
  This function licenses **TBarCode** and removes the demo restrictions. Licensing must be performed before you draw a barcode (e.g. after **TBarCode** has been loaded to memory).

- *BCAlloc()*
  This function sets up and initializes the internal barcode structure. You receive a handle that is used for all other function calls (*pBarCode*). This function must be called before any other function which expects a parameter **pBarCode**.

- *BCSetBCType()*
  Sets the type of the barcode (symbology); e. g. *Code39, Code128, UPC, EAN, 2OF5, ...*

- *BCSetText()*
  Sets the data to be encoded as barcode.

- *BCSetModWidth()* (optional)
  This function is used if an application requires a specific module width. Without this function the module width is computed automatically by **TBarCode**. It adapts to the barcode dimensions (specified via a bounding rectangle) and the current input data.

- More optional barcode settings
  Set the barcode properties according to your application; e.g. *BCSet_PDF417_RowHeight(), BCSetCDMethod(), BCSetBearerBarWidth(), BCSetRatio(), BCSetTextDist(), BCSetLogFont(), ...*

- *BCCheck()* (optional)
  This function checks if the data characters are valid for the selected barcode type. If invalid data was encountered it returns an error-code. If escape-sequences are used, they are not translated in this function. It must be called before *BCCalcCD()*.
  Note: This function call is optional; *BCCreate()* calls this function in any case automatically.

- *BCCalcCD()* (optional)
  This function computes the check-digit(s) for the given input data and the selected check-digit method. The check digits are added to the barcode data automatically. On demand you can retrieve the check digits with *BCGetCheckDigits()*. Please consider that symbology internal check digits (like *modulo 103* of Code-128) are not calculated with this function – they are always part of the created barcode.
  Note: This function call is optional; *BCCreate()* calls this function in any case automatically.

- *BCCreate()*
  This function prepares the barcode structure (pattern) to be drawn with *BCDraw()*. It returns *ErrOk* if everything is ok. If not, it returns an error code (of type *ERRCODE*) that specifies the error in more detail. After *BCCreate()* all parameters of the resulting barcode are available (e.g. number of modules, dimensions, check-digits, meta-description).

- Get Dimensions (optional)
  After *BCCreate()* you can call the methods *BCGetBarcodeHeight(), BCGetBarcodeWidth(), ...*

- *BCDraw()*
  This function draws the barcode into the given device context. The barcode dimensions are set through passing the coordinates of a bounding rectangle. No special mapping is performed.
  Note: Only available in **TBarCode Library for Windows**!

- *BCPostscriptToFile(), BCPCLToFile()*
  These function save the barcode in PostScript or PCL output format.

- *BCFree()*
  This function de-initializes the barcode 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 then DO NOT call subsequent *BCxxxx* functions (except of *BCFree()*). An error code <> 0 indicates an error condition - subsequent calls (except of *BCFree()*) may fail and produce unexpected results.

6.2 Calling Order

► Please note: Since TBarCode 8 the following calling order must be maintained to guarantee the correct conversion of the input data to the target character set:

1. First set all barcode properties (like barcode type, translation of escape sequences, etc.)
2. Then call *BCSetBCText()*
3. Finally call *BCCreate()*

6.3 ANSI and UNICODE

Since version 8 of the **TBarCode Library** each function that has parameters or returns values of type string (=character pointer) is implemented in 2 versions, one for ANSI strings and one for UNICODE strings. ANSI functions end with the suffix A whereas UNICODE functions end with W.

Preprocessor defines are provided to be compatible to ANSI and UNICODE builds. These defines do not have a suffix.

Example:

```
// ANSI function - pass an ANSI text
BCSetTextA(t_BarCode* pBarCode, LPCSTR szText, LONG nLen)

// UNICODE function - pass a UNICODE/wide character text
BCSetTextW(t_BarCode* pBarCode, LPCWSTR szText, LONG nLen)

// Compatibility Define - pass a UNICODE/wide character text
// text or an ANSI depending on whether  UNICODE is set or not
BCSetText (...)
```

To use the compatibility defines for the function names (like *BCSetText()*) while creating a UNICODE build, you have to define the preprocessor variable `_UNICODE` before including `tbarcode.h`:

```
#define _UNICODE
#include "tbarcode.h"
```

If `_UNICODE` is undefined, the compatibility defines refer to the ANSI functions.

7 C/C++ Sample Code

Below are the steps to create a barcode image in C/C++ (only for demonstrative purposes, not all variables declared).

> ► Also check out the fully functional samples provided with the setup – or available as separate download.

Include the header-file:

```
#include "tbarcode.h"
```

Sample code for barcode generation (excerpt):

```
// Initialize library
// nothing to do for Windows (except when using TBarCode as static library)

// License the product
BCLicenseMe("LicenseeName", eLicKindDeveloper, 1, "LicenseKey", eLicProd2D);

// Allocate memory and retrieve barcode handle (pointer)
t_BarCode* pBC;
BCAlloc(&pBC);

// Adjust symbology
BCSetBCType(pBC, eBC_Code128);

// Set barcode data
char* demo = "12345678";
BCSetText(pBC, demo, strlen(demo));

// Set font height for the human readable text
LOGFONTA* pLF = BCGetLogFont(pBC);
pLF->lfHeight = 14;

// Find out wrong characters (check if data can be encoded) (optional)
eCode = BCCheck(pBC)
if (eCode != ErrOk)
{
    // your error handling
}

// Calculate check-digits (optional)
BCCalcCD(pBC);

// Create barcode pattern (bars, spaces)
BCCreate(pBC);

// Set barcode size
// Below we use Hi-Metric [0.01 mm] for hDC
rect.left = 0;        // 0 mm
rect.bottom = 0;      // 0 mm
rect.right = 5000;    // 50 mm
rect.top = 3000;      // 30 mm

// Draw to device context
// Hi-Metric [0.01 mm] for hDC
SetMapMode(hPrinterDC, MM HIMETRIC);
OffsetRect(&rect, Xpos, Ypos);     // position
BCDraw(pBC, hPrinterDC, &rect);

// Save to Postscript file
// Unit is [0.001 mm] for Vector-EPS format
BCSaveImage(pBC, "barcode.eps", eIMPsVector, 50000, 30000, 0, 0);

// Save barcode image to buffer
// Unit is [Pixel] for JPG format
nWidthPx = 2 * BCGetCountModules (pBC);
BCSaveImageToBuffer(pBC, &lpszBuffer, eIMJpg, nWidthPx, 100, 96, 96);
```

```
// Release memory / free barcode structure
BCFree(pBC);

// Clean up
// nothing to do for Windows (except when using TBarCode as static library)
```

8 Custom Drawing Functions for Special Devices

8.1 Why Custom Drawing Functions?

TBarCode Library offers the possibility to implement custom drawing functions. This is useful whenever you control a device which is not supported by any standard-driver functionality. Good examples are laser marking devices, OS-400 specific printers, …

Custom drawing functions can be registered as so called call-back functions. When drawing a barcode the **TBarCode Library** will call the custom drawing functions instead of using the internal drawing routines.

► **IMPORTANT**: Custom drawing functions will only work if a valid **TBarCode** license is provided! Temporary license keys are available on request – please contact support@tec-it.com. Section 9, "How to License TBarCode", describes how to apply a license.

8.2 The General Concept

TBarCode computes a barcode using a so-called meta-description. This meta-description defines in a complete device independent way where bars and where spaces are to be drawn.

Such a meta-description consists of upper- and lowercase letters:

- Uppercase letters are placeholders for bars (or dots)
- Lowercase letters are placeholders for spaces
- The letter itself (A or B or C or …) defines the width of the bar (space) to be drawn.

8.3 Linear Barcodes & PDF417

For barcodes, which are using multiple widths for the bars (or spaces), multiple uppercase (or lowercase) letters are passed to the call-back function:

Uppercase letters = bars:

- A ... bar (actual width = 1 * module width X)
- B ... bar (actual width = 2 * module width X)
- C ... bar (actual width = 3 * module width X)
- D ... bar (actual width = 4 * module width X)
- E ... and so on

The factors for the module width depend on the current print-ratio. In this example the print-ratio for the bars is 1:2:3:4

Lowercase letters = spaces:

- a ... space (actual width = 1 * module width X)
- b ... space (actual width = 2 * module width X)
- c ... space (actual width = 3 * module width X)
- d ... space (actual width = 4 * module width X)
- e ... and so on

The factor for the module widths depend on the current print-ratio. In this example the print-ratio for the bars is 1:2:3:4

X represents the module width. All actual widths of bars or spaces are usually multiples of the module width.

Each barcode symbology uses a pre-defined print-ratio (and this ratio can be adjusted by the user). For example Code39 uses the following print-ratio: 1:3:1:3

- A ... 1 X
- B ... 3 X
- a ... 1 X
- b ... 3 X

It is possible to query the used print-ratio for a specific barcode symbology – please check out the relevant functions *BCGetRatioString, BCGetRatioHint, BCGetCountBars, and BCGetCountSpaces.*

8.4 2D Matrix Codes (Data Matrix, QR-Code, Aztec Code, etc.)

2D matrix codes like Data Matrix, QR-Code, and Aztec Code consist of several rows. The corresponding row patterns are transmitted to a user-defined callback function separately row by row. The callback function is responsible for drawing a single barcode row.

The row pattern consists of uppercase and lowercase letters. Uppercase letters serve as placeholders for black bars (or squares) – lowercase letters are placeholders for spaces (white squares):

Uppercase “A” - black dot/bar, Lowercase “a” - white dot/space

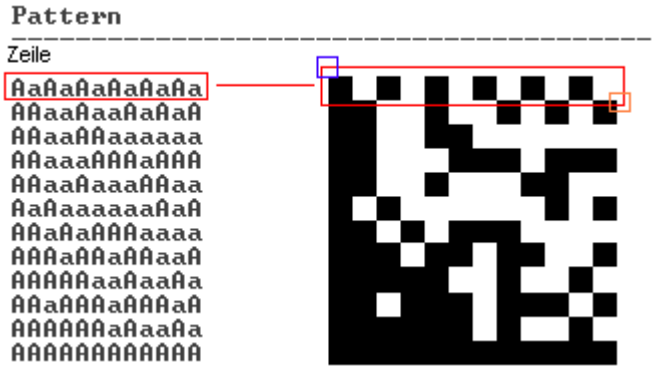

Figure 1: Custom Barcode Drawing

The example above shows Data Matrix, but QR-Code and Aztec Code work in the same manner.

8.4.1 About Drawing

The pattern itself contains no absolute sizes. The matrix dots (A and a) have the same width and height X. This is called the module width. By adjusting the module width to the size of the device dots (pixels) you can minimize the printing tolerances.

8.5 Postal Codes with Bars of Different Height

In contradiction to other linear barcodes many postal codes don’t use multiple widths for the bars but multiple heights. Instead of letters, digits are passed to the call-back function.

A single digit is a placeholder for a sequence of one bar and one space. All bars and spaces have the same width of 1X (=module width). The heights of the bars differ.

Depending on the barcode type the pattern string may contain two, three, or four different digits (=heights).

8.5.1 Barcode with 2 different heights

Barcodes with 2 different heights consist of long bars and short bars, both sharing the same base line, growing bottom up.

- 1 ... long bar
- 2 ... short bar

Figure 2: US Postal Code with 2 different heights

8.5.2 Barcodes with 3 different heights

Barcodes with 3 different heights consist of long bars and short bars either growing bottom up or to down, and short bars.

- 0 ... long bar
- 1 ... short bar, growing top down
- 2 ... short bar, growing bottom up

Figure 3: Pharmacy Code Two-Track with 3 different heights

8.5.3 Barcodes with 4 different heights

Barcodes with 4 different heights consist of long bars, medium sized bars either growing bottom up or to down, and short bars, drawn vertically centered.

- 0 ... long bar (**Full**)
- 1 ... medium sized bar, growing top down (**Ascender**)
- 2 ... medium sized bar, growing bottom up (**Descender**)
- 3 ... short bar (**Tracker**)

Figure 4: Australian Postal Code with 4 different heights

8.5.4 About Drawing

The pattern itself contains no absolute sizes. Following table gives you detailed hints how to convert a given pattern to a valid barcode.

Size and position of a bar is defined by the bar's height (in percent of a full height bar) and the bar's distance from the upper edge (also in percent of a full height bar).

Barcode Type(s)	Pattern Digit			
	0	1	2	3
US Postal, CEPNet, Planet	**Height:** 100% **Distance:** 0%	**Height:** 38.5% **Distance:** 61.5%	--	--
Pharmacode 2-Track	**Height:** 100% **Distance:** 0%	**Height:** 50% **Distance:** 50%	**Height:** 50% **Distance:** 0%	--
Australian Postal	**Height:** 100% **Distance:** 0%	**Height:** 62% **Distance:** 0%	**Height:** 62% **Distance:** 38%	**Height:** 24% **Distance:** 38%
Royal Mail 4 State, KIX	**Height:** 100% **Distance:** 0%	**Height:** 62,5% **Distance:** 0%	**Height:** 62,5% **Distance:** 37,5%	**Height:** 25% **Distance:** 37,5%
Intelligent Mail® Barcode, DAFT, Japanese Postal	**Height:** 100% **Distance:** 0%	**Height:** 66,7% **Distance:** 0%	**Height:** 66,7% **Distance:** 33,3%	**Height:** 33,3% **Distance:** 33,3%

Table 1: Drawing Barcodes with Multiple Heights

8.6 Control Patterns

Apart from letters and digits the pattern string may contain control characters. In the following you find a short overview.

8.6.1 Protruding Bars for EAN and UPC Codes

Following patterns specify changes of the bar length

- ASCII (254) ... Begin of section with long (=protruding) bars
- ASCII (255) ... End of section with long (=protruding) bars
- ASCII (253) ... Begin of an add-on section which last until the end of the code

The barcode types **EAN 8/13** and **UPC A/E** contain protruding bars on the begin, in the middle, and on the end of the barcode. With **ASCII (254)** the section with protruding bars starts, after **ASCII (255)** it ends. Bars of "normal" length follow. Protruding bars are extended on the bottom side by about the half height of the human readable text.

Add-on sections start with **ASCII (253)**. They continue until the end of the code. Add-on bars leave space for the text above the barcode and align at the bottom with the protruding bars.

Figure 5: EAN8 with 5 add-on digits

8.6.2 Increment and Decrement the Bar Width for EAN and UPC Codes

- ASCII (252) ... Begin bar width increment
- ASCII (251) ... End bar width increment
- ASCII (250) ... Begin bar width decrement
- ASCII (249) ... End bar width decrement

Based on the definition of EAN and UPC codes the width of some bars has to be increased whereas the width of other bars has to be decreased. This is done by the control patterns shown above.

Please examine the *EAN specification* for a detailed description.

9 How to License TBarCode

In order to enable the full-featured version, you need a valid license key from TEC-IT. A description of the available license-types as well as all necessary information for ordering can be found at http://www.tec-it.com/prices.

If you don't know the license type according to your application, please ask our sales team (sales@tec-it.com).

For placing an online order check out http://www.tec-it.com/order/.

► For testing the call-back API or other evaluation purposes you can request a time-limited license key from support@tec-it.com.

9.1 Demo Limitations

Whenever **TBarCode** is not licensed with a valid license key, an additional text “Demo” or an additional horizontal bar is drawn across the barcode. In addition all call-back functions (for custom barcode drawing functions) are disabled.

To remove the demo limitations call *BCLicenseMe()* with valid a license key. For example:

```
ERRCODE eCode = BCLicenseMe("John Smith", eLicKindSite, 1,
                            "01234567890ABCDEFGHIJKLMNOPQRSTU", eLicProd2D);
```

In Windows: *BCLicenseMe()* should be called as the first function of **TBarCode Library.**

In UNIX: *BCLicenseMe()* should be called directly after *BCInitLibrary()*.

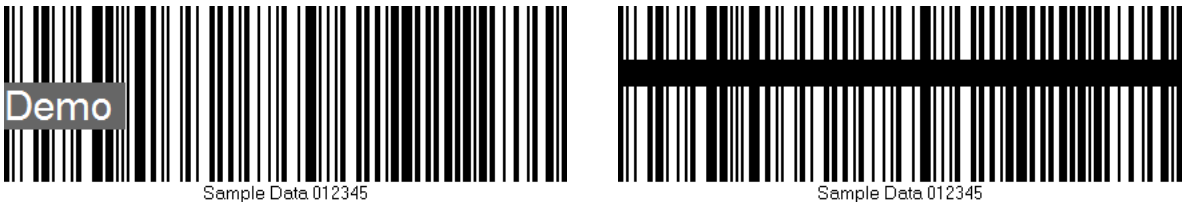

*Figure 6: Barcodes rendered **without** valid license*

Figure 7: Barcode rendered with valid license

10 Redistributing TBarCode

This chapter explains what is important when redistributing a custom application that uses the **TBarCode Library**.

> ► Please note that in most cases you need a developer license for re-distribution of **TBarCode Library** (except for in-house applications which are bound to one or more sites).

10.1 Dependencies

An application that uses **TBarCode SDK** requires the following files:

- **TBarCode11.dll**
  This is a native Win32 (x86) library that performs barcode generation. This file is mandatory for 32-Bit applications.
- **TBarCode11_x64.dll**
  This is a native Win64 (x64) library that performs barcode generation. This file is mandatory for 64-Bit applications.
- **zlib1.dll**
  This is a 3rd-party library that performs data compression. This file is optional. It is only required, when data compression in barcodes is enabled.

These files are located in the folder *C:\Program Files\Common Files\TEC-IT\TBarCode\11.0* (or in the folder you have chosen during setup).

TBarCode Library further requires the **Microsoft VC15 Common Runtime DLLs**. See next section how to distribute them with your application.

10.2 Redistribution

When redistributing a custom application the files described above need to be redistributed together with the application. The DLLs should be located in the same folder as the executable. Other files than those listed above must not be redistributed.

You may have to redistribute the **Visual C++ 2015 Update 3 runtime components** (MS CRT14.0 DLLs) with your application and ensure they are installed on the target computer.

- You can install these components with the *Microsoft Visual C++ 2015 Redistributable Update 3 RC Packages* available at
  - https://www.microsoft.com/en-US/download/details.aspx?id=52685

  The package installs all required runtime DLLs.

Please contact TEC-IT Support if you need help.

11 Contact and Support Information

TEC-IT Datenverarbeitung GmbH

Address: Hans-Wagnerstr. 6
AT-4400 Steyr
Austria/Europe
Phone: +43 / (0)7252 / 72 72 0
Fax: +43 / (0)7252 / 72 72 0 – 77
Email: office@tec-it.com
Web: http://www.tec-it.com
FAQ: http://www.tec-it.com/support/faq/tbarcode/Default.aspx