QS-Barcode Erkennung

(32- und 64-Bit Version)

Software Development Kit

für Microsoft Windows Betriebssysteme

Version 5.0.1

QS QualitySoft GmbH
Tempowerkring 21a
21079 Hamburg

Tel: +49 (0) 40 790 100 40
Fax: +49 (0) 40 790 100 44
info@qualitysoft.de

Neueste Infos im Internet:
www.qualitysoft.de

Inhaltsverzeichnis

Das QS-Barcode SDK verwendet die Open Source FreeImage.dll zum Lesen der Bilddateien.

Den Sourcecode und Details zur FreeImage Public License 1.0 finden Sie im Internet unter https://freeimage.sourceforge.io/ .

This software is using LIBTIFF
Copyright (C) Sam Leffler
Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

The software is provided "as-is" and without warranty of any kind, express, implied or otherwise, including without limitation, any warranty of merchantability or fitness for a particular purpose.

In no event shall Sam Leffler or Silicon Graphics be liable for any special, incidental, indirect or consequential damages of any kind, or any damages whatsoever resulting from loss of use, data or profits, whether or not advised of the possibility of damage, and on any theory of liability, arising out of or in connection with the use or performance of this software.

This software is based in part on the work of the Independent JPeg Group
LibJpeg Copyright (C) 1994-1997, Thomas G. Lane

Portions of this software are copyright © 2009 The FreeType Project (www.freetype.org). All rights reserved.

Microsoft, Microsoft .NET Framework sowie Microsoft Windows sind Warenzeichen oder eingetragene Warenzeichen der Microsoft Corporation.

Weitere in diesem Dokument erwähnte Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Marken und unterliegen als solche den gesetzlichen Bestimmungen.

1 Einleitung

QS-Barcode SDK ist eine leistungsfähige Software zur **Erkennung von Barcodes** aus digitalisierten Bildern und PDF-Dokumenten, die ab Version 4.9 in einer 32-Bit und einer 64-Bit Version vorliegt und ab Version 5.0 Multithreading-fähig ist. Das **QS-Barcode SDK** wurde mit Visual Studio 2013 erstellt.

TIPP	Wenn Sie Software für **Barcode-Druck** suchen, werfen Sie einen Blick auf unsere Partner-Seite http://www.qualitysoft.de/index.php?id=25

Seitdem 1998 die erste Version des **QS-Barcode SDK** veröffentlicht wurde, ist es kontinuierlich weiterentwickelt worden.
Inzwischen ist die Barcode-Erkennung weltweit auf weit über 50.000 Arbeitsplätzen im Einsatz.

Digitalisierte Bilder mit Barcodes entstehen zum Beispiel durch Scannen mit Dokumenten-Scannern, durch digitale Fotografie oder durch Fax-Empfang.
Es werden monochrome schwarz/weiß Bilder, Bilder in Graustufen sowie Farbbilder unterstützt. Alle gängigen Bilddatei-Formate wie Bitmap, TIFF, JEPG und viele weitere können verarbeitet werden. Auch die Lesung von Barcodes aus Adobe PDF Dokumenten funktioniert mit QS-Barcode. Für Performance-Anwendungen besteht auch die Möglichkeit, ohne Dateiablage direkt Bilddaten im Hauptspeicher zu verarbeiten.

QS-Barcode SDK ist keine eigenständige Software, sondern eine Programmbibliothek für Programmierer, die damit die Möglichkeit haben, eigene Software mit der Fähigkeit der Barcode-Erkennung auszustatten.
Dazu stehen mehrere Schnittstellen zur Verfügung (LIB, DLL, OCX).
Wählen Sie diejenige aus, die optimal zu Ihren Anforderungen passt.
QS-Barcode läuft unter Windows Betriebssystemen.

TIPP	Ganz ohne Programmierung kommen Sie aus, wenn Sie unser Produkt **QS-DocumentAssembler** für die Barcode-Erkennung nutzen. Mehr unter: https://qualitysoft.de/produkte/qs-documentassembler

Dieses Handbuch für das **QS-Barcode SDK** dokumentiert, wie die Programmbibliothek eingebunden wird, wie die Barcode-Erkennung parametrisiert wird und wie bei der Auslieferung und Installation auf Kundensystemen verfahren werden muss.

2 Liste der unterstützten Barcode Typen

Die gängigen linearen Barcodetypen **Codabar**, **Code 2/5 interleaved, Code 2/5 industry, Code 39, Code 39 extended**, **Code 32**, **EAN 8**, **EAN 13, UPC A, UPC E, Code 128**, **EAN128** und **Code 93** sind mit **QS-Barcode** lesbar.
Die POS-fähigen Barcodes der **GS1 DataBar** Familie, die in Zukunft die EAN und UPC Codes ersetzen werden, sind bereits heute mit QS-Barcode lesbar.
Zusätzlich wird eine Vielzahl von seltenen Typen unterstützt: **Code 2/5 IATA**, **Code 2/5 3 Matrix**, **Code 2/5 3 Datalogic**, **Code 2/5 BCD Matrix**, **Code 2/5 invertiert, Code 93 extended** und **Code 11**. Als Besonderheiten können (Standard Einspur-) **Pharmacodes** und **Patchcodes** erkannt werden, ebenso wie der Stacked-Barcode **Codablock F.**
Als **Optionen** werden die Erkennung der **zweidimensionalen Barcodes PDF417**, **DataMatrix** (ECC 200 sowie ECC000, ECC050, ECC080, ECC100 und ECC140), **QR Code** (Model 2) und **Aztec Code** angeboten.
Diese Barcodes enthalten sehr viel mehr Informationen als herkömmliche Barcodes, je nach Typ und Größe können bis zu **3000 Zeichen** pro Barcode verschlüsselt werden. Der gesamte Text dieser Seite würde in einem Barcode Platz finden. 2D Barcodes sind zudem durch integrierte Fehlerkorrektur toleranter gegenüber Datenfehlern, die beim Druck oder beim Scannen entstehen.

Der **DataMatrix (ECC 200)** wird von der Deutschen Post als "Stampit" Briefmarke verwendet. Es werden auch Binärdaten kodiert.

Der rechte Barcode enthält 15 Zeichen (Inhalt: 19 200375 J 000). Beide DataMatrix Barcodes sind in Originalgröße abgebildet.

Der **PDF417** besteht aus mehreren Zeilen. In ihm kann der volle Zeichensatz verschlüsselt werden, bis zu 1850 Zeichen. Er ist variabel in der Breite und Höhe. Der PDF417 wurde von der **KBV** zum Bedrucken von medizinischen Formularen ausgewählt.

Der **QR Code** ist in Japan entwickelt worden und wird im asiatischen Raum häufig genutzt, vor allem weil er für die Kodierung von Kanji-Zeichen optimiert ist. Aber auch Binärdaten und alphanumerische Daten können effizient verschlüsselt werden.
Auch der **QR Code** verfügt wie die anderen 2D-Barcodes über eine intelligente Fehlerkorrektur.

Der quadratische **Aztec Code** wurde 1995 von Welch Allyn entwickelt. Er hat **ein** Suchelement, das genau in der Mitte des Codes ist und aus mehreren ineinander befindlichen Quadraten besteht. Der Aztec Code benötigt keine Ruhezone (keinen weißen Rand) wie die anderen Codes. Größen von minimal 15x15 bis maximal 151x151 Elementen sind möglich. Auch der Aztec Code hat eine intelligente Fehlerkorrektur mit vielen Redundanzstufen.

Der Aztec Code wird z.B. von der Deutschen Bahn für online-Tickets verwendet.

3 Welche Schnittstelle einsetzen?

QS-Barcode bietet verschiedene Schnittstellen für verschiedene Bedürfnisse und Anwendungsumgebungen. Dieser Abschnitt soll Ihnen helfen, zu entscheiden, welche Schnittstelle für Ihr Projekt passt.

Datei oder Hauptspeicher als Datenquelle?

Die meisten unserer Kunden nutzen die **Datei-Schnittstellen** und überlassen QS-Barcode damit das Laden der Bilddaten.
Als Datei-Schnittstellen stehen zur Verfügung:

f_ocx	Eine ActiveX Komponente
f_dll	Eine Windows API Dynamic Link Library (DLL)

Welche dieser beiden Schnittstellen sie wählen, hängt von Ihren persönlichen Vorlieben und davon ob, welche Einbindung Ihre Entwicklungsumgebung besser unterstützt.

Aus Performance-Gründen, wenn Sie nur auf Bildausschnitten arbeiten oder eigene Bildvorverarbeitung durchführen wollen, können Sie die **Hauptspeicher-Schnittstellen** nutzen.
Als Hauptspeicher-Schnittstellen stehen zur Verfügung:

h_dll	Eine Windows API Dynamic Link Library (DLL) Übergabe der Bilddaten als Windows Handle auf eine Device Independent Bitmap (DIB)
p_lib	Eine Library zur statischen Bindung in C/C++ Projekte Übergabe ist ein Zeiger auf die monochromen Bilddaten.

Achtung: Die Schnittstelle **p_lib** unterstützt nur schwarz/weiß-Bildinformationen. Alle anderen Schnittstellen (**f_ocx, f_dll** und **h_dll)** unterstützen Farb-, Graustufen- und schwarz/weiß-Bilder.

Ausgelieferte Versionen der Libs/DLLs

Alle QS Barcode DLLs und LIBs sind in vier verschiedenen Versionen (MD, MDd, MT, MTd) im QS-Barcode SDK enthalten.

1. Multi-threaded (MT)
2. Multi-threaded Debug (MTd)
3. Multi-threaded DLL (MD)
4. Multi-threaded DLL Debug (MDd)

Sie unterscheiden sich durch die unterschiedlichen eingebundenen System Runtime Libraries und liegen jeweils in der Debug- und Release-Version vor.

Die Versionen MT und MTd bedeuten, dass die libcmt.lib statisch gelinkt wurde.

Die Versionen MD und MDd bedeuten, dass die msvcrt.lib dynamisch gelinkt wurden. Dies hat zur Folge, dass die msvcr120.dll sowie die MSVCP120.DLL separat ausgeliefert werden müssen. Diese Dateien befinden sich im "Microsoft Visual C++ 2013 Redistributable"-Package.
Wenn Ihnen diese Unterscheidung nichts sagt, so wählen Sie die Version "MT".

Programmbeispiele

Viele Programmbeispiele werden bei der Installation des QS-Barcode SDK mit installiert. Orientieren Sie sich an diesen Beispielen, und die Nutzung der Barcode-Ernennung wird Ihnen leicht fallen.
Es liegen Beispiele für Visual Basic 6, Visual Basic for Applications, C, Java, Delphi und C# vor.

Für alle Schnittstellen gilt: **QS-Barcode SDK** läuft unter den Microsoft Betriebssystemen Windows Server 2003, Windows Server 2008, Windows Server 2012, Windows Vista, Windows 7 und Windows 8.x

.

4 Lizenzdateien

QS-Barcode SDK benutzt ein Lizenzierungsverfahren mit Lizenzdateien. In der Auslieferung des QS-Barcode SDK sind zwei Lizenzdateien namens QSBC.LIC enthalten.
Eine dieser Lizenzdateien QSBC.LIC ist für den Arbeitsplatz bestimmt, auf dem Sie das SDK installiert haben und auf dem die Entwicklung der Software erfolgt, die QS-Barcode einbindet.
Die andere Lizenzdatei QSBC.LIC ist für den ersten Anwendungs-Arbeitsplatz bestimmt, auf dem Sie Ihre Software installieren. Für jeden weiteren Anwendungs-Arbeitsplatz ist der Erwerb einer **Runtime-Lizenz** bei uns notwendig (in der QS-Barcode SDK ist also **eine** Runtime-Lizenz bereits enthalten).

Auf dem Entwickler-Arbeitsplatz bei 32-Bit Betriebssystemen kopieren Sie während des Setup-Prozesses die QSBC.LIC Datei manuell in das Windows-System Verzeichnis (normalerweise C:\WINDOWS\SYSTEM32).

Wenn Sie unter 64-Bit Betriebssystemen entwickeln und das 64-Bit QS Barcode SDK einsetzen, kopieren Sie die QSBC.LIC ebenfalls in das Windows-System Verzeichnis. Wenn Sie jedoch das 32-Bit QS Barcode einsetzen, kopieren Sie die Datei in das „SYSWOW64“-Verzeichnis.

Bei der Installation auf Anwendungs-Arbeitsplätzen müssen (und dürfen) Sie eine Lizenzdatei QSBC.LIC mit der Anwendung, in die Sie QS-Barcode eingebunden haben, ausliefern.

Die Datei wird im Verzeichnis der QS-Barcode nutzenden Anwendung gesucht. Bei nicht vorhandener oder fehlerhafter Lizenzdatei schaltet QS-Barcode bei linearen Barcodes in den **DEMO-Modus** (mit Ergebnis-Verfälschungen). Bei 2D-Barcodes erscheint eine Meldung, dass der Barcodetyp nicht unterstützt wird.

Es werden folgende Lizenztypen unterschieden

Bezeichner	**Bedeutung**	**Wert (HEX)**
BC_LIC_DEMO	Demo-Modus	0x00000001
BC_LIC_LINEAR	Modus für lineare Barcodes	0x00000002
BC_LIC_PDF417	Modus für PDF417 Barcodes	0x00000004
BC_LIC_DATAM	Modus für DataMatrix Barcodes	0x00000008
BC_LIC_QRCODE	Modus für QR Code Barcodes	0x00000010
BC_LIC_AZTEC	Modus für Aztec Barcodes	0x00000020

Kombinationen durch OR sind möglich, der Wert 0x00000007 besagt z.B. BC_LIC_DEMO OR BC_LIC_LINEAR OR BC_LIC_PDF, also lineare und PDF417 Barcodes werden gelesen, aber im Demo-Modus.

Um den aktuellen Lizenzmodus abzufragen kann die Funktion QSLicense() genutzt werden. Wenn sich die Lizenz nicht im Systempfad befindet, können Sie auch QSLicense1(char *szLicenseFile) mit direkter Übergabe der Lizenzdatei benutzen.

5 Über Barcodes

5.1 Übersicht

Der häufigste Einsatz für die Barcodeerkennung ist die schnelle und sichere Identifikation und Indizierung von Dokumenten für Datenerfassung (OCR, ICR, etc.) und Archivierung. Das Scannen in diesem Umfeld wird in der Regel mit einer Auflösung von 200-300 dpi durchgeführt. Um bei dieser geringen Auflösung eine fehlerfreie Erkennung der Barcodes zu erreichen, müssen die Barcodes sauber gedruckt sein und dürfen nicht zu dicht sein. Empfohlen werden bei 200 dpi maximal 2 Zeichen/cm, bei 300 dpi 3 Zeichen/cm. Also sollte z.B. ein 8-stelliger Barcode min. 4 cm breit sein, um bei 200 dpi sicher erkannt zu werden.
Neben der verwendeten Scannerauflösung und der Barcodebreite haben weitere Faktoren großen Einfluss auf eine zuverlässige Erkennung:

- Barcodetyp
- Scanner Einstellungen
- Höhe des Barcodes
- Druck- und Papierqualität
- Schräglage (besonders bei Etiketten)
- Ruhezone um den Barcode herum

Bei Barcodes ist nicht nur die Anzahl der Striche und Lücken, sondern auch deren relative Breite wichtig. Alles, was die Balken verändert, verursacht eine fehlerhafte Erkennung der Barcodesymbole. Zum Beispiel auch die Kontrasteinstellung, die die Breite der Balken ändert.

QS-Barcode bietet eine Reihe von Parametern, um die Qualität der Erkennung zu sichern, sofern die Grundvoraussetzungen gegeben sind.
Es sollte vor dem Routineeinsatz auf jeden Fall ein größerer Test mit den geplanten Barcodes unter produktionsnahen Bedingungen durchgeführt werden.
Senden Sie Ihre Fragen und Testbilder an: support@qualitysoft.de.
Wir helfen Ihnen gern!

5.2 Barcodetypen

Im Laufe der Zeit und für verschiedene Einsatzgebiete sind unterschiedliche Barcodetypen entstanden. Die gebräuchlichsten linearen Barcodetypen sehen Sie hier:

Code 39: 123456

Code 2 aus 5 interleaved: 123456

Code 128: ABC987

1 234567 890128 EAN 13

Einige Eigenschaften von gebräuchlichen Typen:

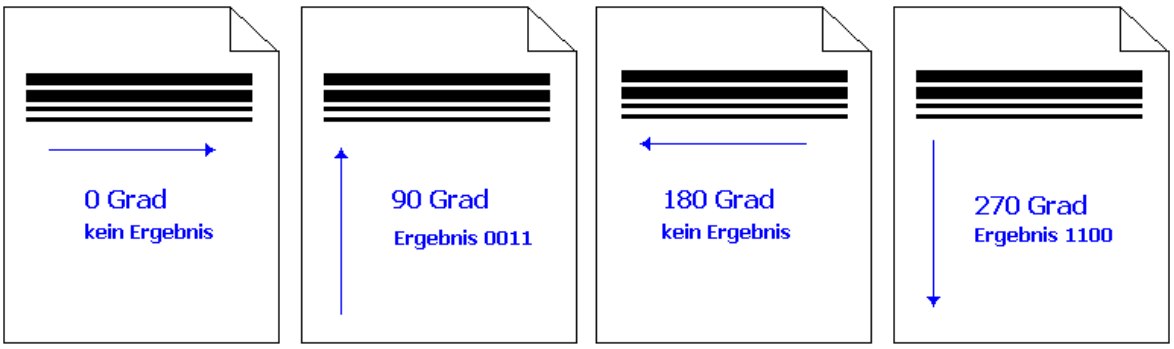
Typ	Zeichensatz	Länge
Code 39	0-9 A-Z - $ % + /	variabel
Code 39 extended	kompletter ASCII-Zeichensatz	variabel
Code 2/5 interleaved	0-9	variabel (gerade Anzahl)
Code 2/5 Industrie	0-9	variabel
Code 93	0-9 A-Z - $ % + / 4 Sonderz.	variabel
Codabar	0-9 - $: + / .	variabel
Code 128	kompletter ASCII-Zeichensatz	variabel
EAN 8 / EAN 13	0-9	8 / 13
UPC A / UPC E	0-9	12 / 6
EAN128	kompletter ASCII-Zeichensatz	variabel

QS-Barcode liest neben den obigen auch noch eine Reihe anderer Typen, von denen die meisten nur in besonderen Bereichen (aus historischen Gründen) eingesetzt werden. Die aktuelle Liste der möglichen Typen sehen Sie in der Beschreibung der Testapplikation (s. unten).
Wenn Bedarf an der Lesung weiterer, nicht aufgeführter Typen besteht, sprechen Sie uns bitte an.

Als Sonderfälle werden der **"Patchcode"** und der **"Pharmacode"** gelesen.
Diese beiden Codes haben ganz spezielle Einsatzgebiete, sie können nicht universell eingesetzt werden. Sie haben keine Start- und Endekennungen, die bei den normalen Typen nicht nur die Lokalisierung des Barcodes erleichtern, sondern dadurch, dass jeder Barcodetyp eindeutige Start- und Endemuster hat, auch die automatische Typerkennung ermöglichen.

Die Erkennung von Patchcodes und Pharmacodes muss jeweils durch einen gesonderten Aufruf erfolgen, es ist nicht möglich, in einem Aufruf der Barcodeerkennung z.B. einen Patchcode und einen EAN13 Handelsbarcode zu erkennen.

Der **Patchcode** kann von einigen Scannern zur Steuerung und Trennung von Scanjobs verwendet werden. Er zeichnet sich durch seine sehr langen Striche und sehr geringe Anzahl (immer 4) von Strichen aus.



Der erkannte Patchcode wird wie üblich im Barcodeergebnis zurückgegeben.

Der **Pharmacode** wird bei der Produktion und Verpackung von Arzneimitteln zur Kennzeichnung und Kontrolle verwendet.

Hinweis: Bei dem Begriff Pharmacode gibt es eine Reihe von Bedeutungen. Der oben beschriebene (Standard Einspur-) Pharmacode darf nicht mit der Pharma-Zentralnummer (PZN) verwechselt werden, die in Deutschland zentral für die Identifikation zugelassener Arzneimittel vergeben wird. Für die PZN wird als Barcodetyp Code 39 verwendet, in Italien wird für den "Pharmacode" der Typ Code 32 verwendet.

666

111111

Beispiel: Pharmacode

Der Pharmacode enthält zwischen 4 und 16 Strichen und ermöglicht die Darstellung einer Zahl von 1 bis 131070. Der Code sollte immer in einem eingeschränkten Feld gesucht werden, da sonst sehr viele andere Muster auch als "Pharmacode" erkannt werden. Der Zweispur-Pharmacode und der 2D-Pharmacode sind auf Anfrage verfügbar.

5.3 Übersicht über die 2D Barcode Optionen

Um mehr Daten in die Barcodes zu schreiben, wurden 2D (zweidimensionale) Barcodes entwickelt. In der Anfangsphase wurde von verschiedenen Firmen eine große Anzahl von unterschiedlichen Typen entwickelt, von denen viele nur für spezielle Anwendungsfälle eingesetzt werden (UPS, etc.). 2D-Barcodes werden, je nach Inhalt, in sehr unterschiedlichen Größen eingesetzt und können auch binäre Daten enthalten, so dass die Daten leicht auch verschlüsselt werden können (z.B. biometrische Daten auf Ausweisen).

Inzwischen haben sich der PDF417 und der Data Matrix (meist ECC 200) durchgesetzt, vor allem im asiatischen Raum spielt auch der QR Code eine wichtige Rolle. Diese Codes sind international standardisiert.

QR Code **PDF 417** **Data Matrix** **Aztec Code**

Mit den Optionen **QR Code, PDF417, Data Matrix** und **Aztec Code** kann **QS-Barcode** um die Lesung dieser Typen erweitert werden.

Die Einbindung der Library in eigene Entwicklungsumgebungen gestaltet sich größtenteils wie auch bei der Erkennung von linearen Barcodes.

Mit den Optionen werden zudem zusätzliche Beispielprogramme inkl. Quellcode mitgeliefert, die den Einsatz und die Erkennung der 2D-Barcodes demonstrieren. In der Testapplikation und auf unserer Website finden Sie interessante Anwendungsbeispiele zum Einsatz von 2D Barcodes.

Beachten Sie: Wenn Sie lineare Barcodes und 2D Barcodes lesen wollen, müssen die sie Erkennungsroutine zweimal ausführen, es ist ein Durchgang für die Erkennung linearer Barcodes notwendig und ein getrennter Durchgang für die Erkennung des 2D Barcodes Ihres Typs. Dies ist so notwendig, weil bei der Erkennung von DataMatrix, QR Code, Aztec Code und PDF417 die Parameter der Barcode-Erkennung andere Bedeutungen haben und daher auch andere Werte zugewiesen werden müssen.

6 Das Programm bcTester

Mit bcTester können Sie Ihre Barcodes testen und einfach mit verschiedenen Einstellungen experimentieren.

Unter http://www.bctester.de/fileadmin/ContentFiles/de/download/bctester_de.zip finden Sie stets die neueste Version.

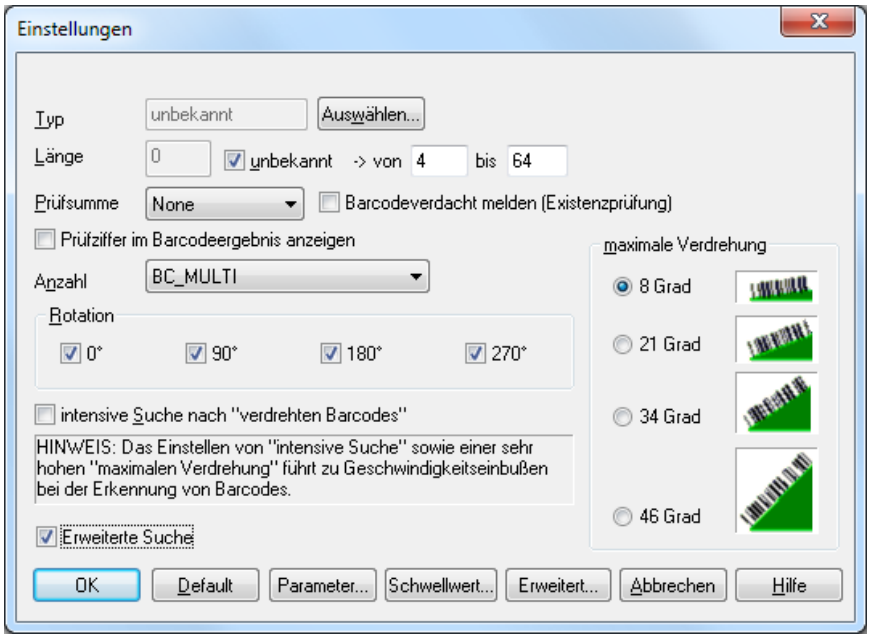
6.1 Barcodes Testen

Dazu gehen Sie wie folgt vor:

1. Öffnen Sie ein Bild mit dem zu testenden Barcode (Drag and Drop wird unterstützt).
2. Ziehen Sie mit der Maus ein Rechteck um den Barcode. Rechts und links sollte mindestens 20 Pixel Platz bis zum Barcode sein (Ruhezone). Wenn Sie kein Rechteck ziehen, wird auf dem gesamten Bild gesucht.
3. Stellen Sie die Barcodeparameter über <Barcode> <Einstellungen...> (STRG-N) ein.
4. Starten Sie mit <Barcode> <Erkennen> (STRG-L) die Erkennung. Die Ergebnisse werden angezeigt.

6.2 Barcode Einstellungen

In diesem Dialog finden Sie die Barcode-Einstellungen von bcTester.
Die an der Oberfläche dargestellten Parameter entsprechen denen des QS-Barcode SDK, zur die Bezeichnungen und die Darstellung weicht teilweise voneinander ab. Daher erfolgt hier eine genaue Zuordnung auf die zugeordneten SDK-Begriffe, die im Kapitel 11 Parameter detailliert beschrieben werden.



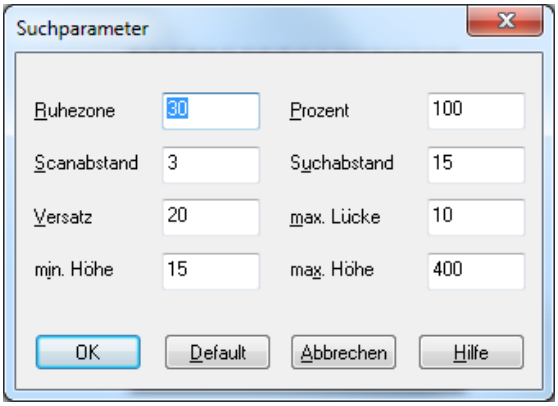
Mit der Schaltfläche "Auswählen" können Sie einen oder mehrere Typen auswählen.

Hinter der Schaltfläche "Erweitert" verbergen sich selten benötigte Einstellungen wie Ruhezonen und Scanabstände.

Die "Default-Werte" sind für Barcodes von üblichen Form und Größe eine gute Wahl.

Einstellungen in bcTester	**"Barcode-Parameter" im SDK**
Typ	iBC_Type
Länge	iBC_Length
[Länge] unbekannt	iBC_Length = 0
[Länge] von	iLaengeVon
[Länge] bis	iLaengeBis
Prüfsumme	iBC_Checksum
Barcodeverdacht melden (Existenzprüfung)	iBC_Checksum (Wert: BC_EXISTENCE)
Prüfziffer im Barcodeergebnis anzeigen	iBC_Checksum (Wert BC_REPORT_CHECKSUM)
Anzahl	iBC_ReadMultiple
Rotation	iBC_Orientation (Werte: BC_0, BC_90, BC_180, BC_270, oder-Verknüpfung möglich)
Maximale Verdrehung	iBC_Orientation (Werte: BC_SKEW_LIGHT, BC_SKEW_MEDIUM, BC_SKEW_HEAVY, oder-Verknüpfung möglich)
intensive Suche	iBC_Orientation (Wert: BC_SKEW_DENSE_SEARCH, oder-Verknüpfung möglich) Bei DataMatrix gilt als Wert: DM_INTENSIVE_SEARCH
Erweiterte Suche	AdvancedSearch

Im Dialog "Suchparameter " finden Sie speziellere Einstellungen.

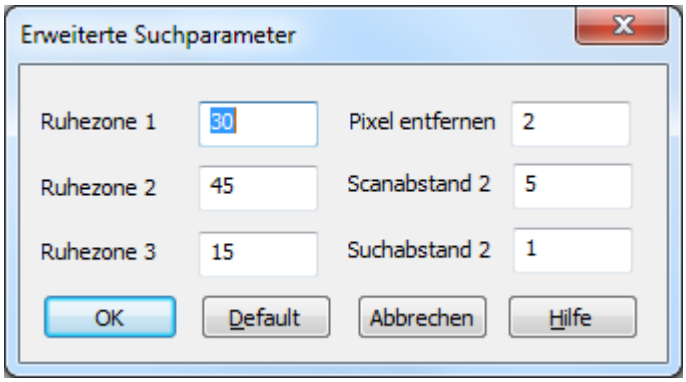


Die "Default-Werte" sind für Barcodes von üblichen Form und Größe eine gute Wahl.

In Sonderfällen und wenn die Qualität der gescannten Barcodes schlecht ist, kann durch Anpassung dieser Parameter in der Regel die Lesung ermöglicht und die Lesequalität und -sicherheit erhöht werden.

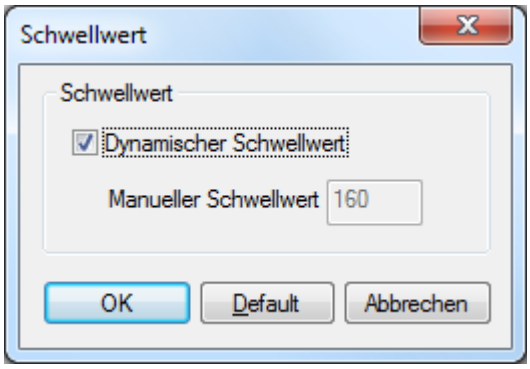
Einstellungen in bcTester	**"Barcode-Parameter" im SDK**
Ruhezone	iBC_LightMargin
Prozent	iBC_Percent
Scanabstand	iBC_ScanDistBarcode
Suchabstand	iBC_ScanDistance
Versatz	iBC_Tolerance
max. Lücke	iBC_MaxNoVal
min. Höhe	iBC_MinHeight
max. Höhe	iBC_MaxHeight

Im Dialog "Erweiterte Suchparameter" finden Sie die Einstellungen für die Erweiterte Suche.



Einstellungen in bcTester	**"Barcode-Parameter" im SDK**
Ruhezone 1	iBC_Lightmargin1
Ruhezone 2	iBC_Lightmargin2
Ruhezone 3	iBC_Lightmargin3
Pixel entfernen	iBC_RemovePixel
Scanabstand 2	iBC_ScanDistBarcode2
Suchabstand 2	iBC_ScanDistance2

Im Dialog "Schwellwert" finden Sie die Einstellungen für den Schwellwert.

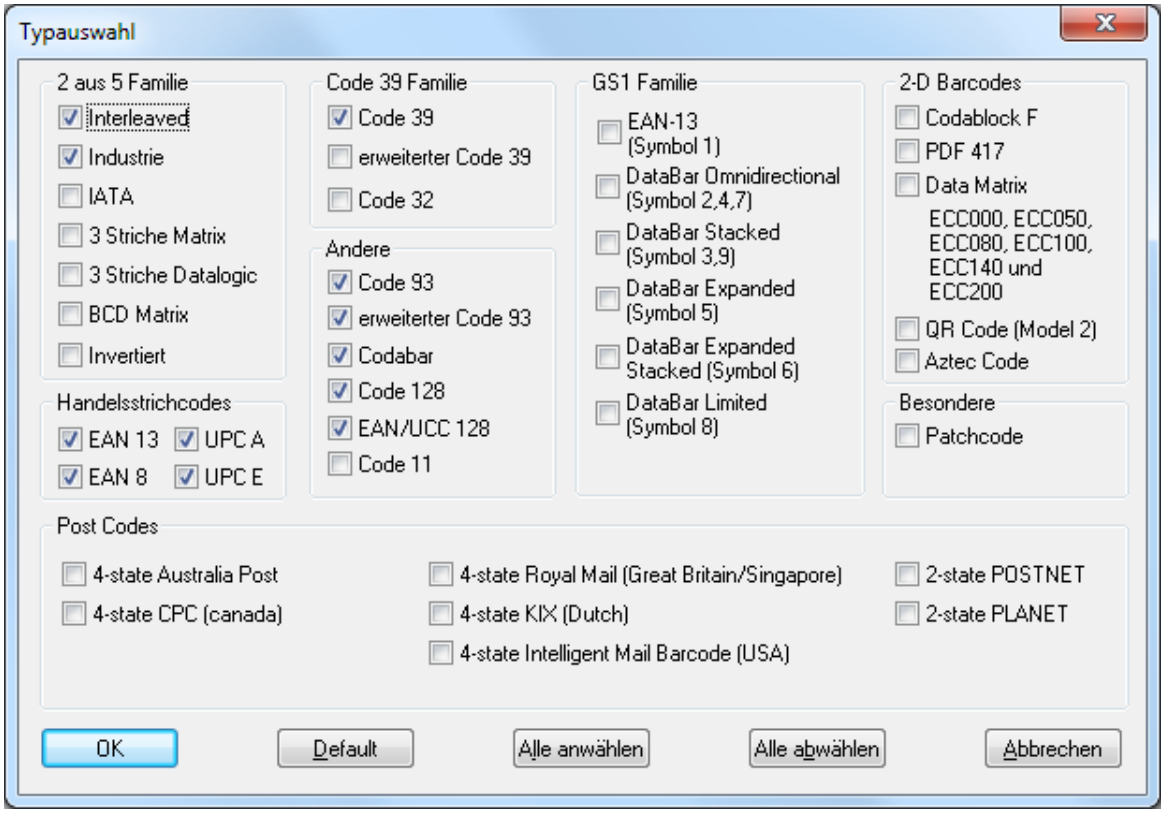


Einstellungen in bcTester	"Barcode-Parameter" im SDK
Dynamischer Schwellwert	DynamicThreshold
Manueller Schwellwert	iBC_Threshold

Beachten Sie: Bei 2D Barcodes stehen diese Dialoge nicht zur Verfügung bzw. nicht alle Parameter sind relevant. Siehe dazu auch Kapitel "14.2 Besondere Einstellungen beim Data Matrix Code" und die darauffolgenden Kapitel.

TIPP	Das Dokument bcTipps.pdf, das bei der QS-Barcode SDK Installation ebenfalls installiert wird, enthält viel wertvolle Hinweise zur optimalen Einstellung von Barcode Parametern.

In der Typauswahl erfolgt die Selektion der zu erkennenden Barcodes. Im SDK werden entsprechend die gewünschten Typkonstanten gesetzt.

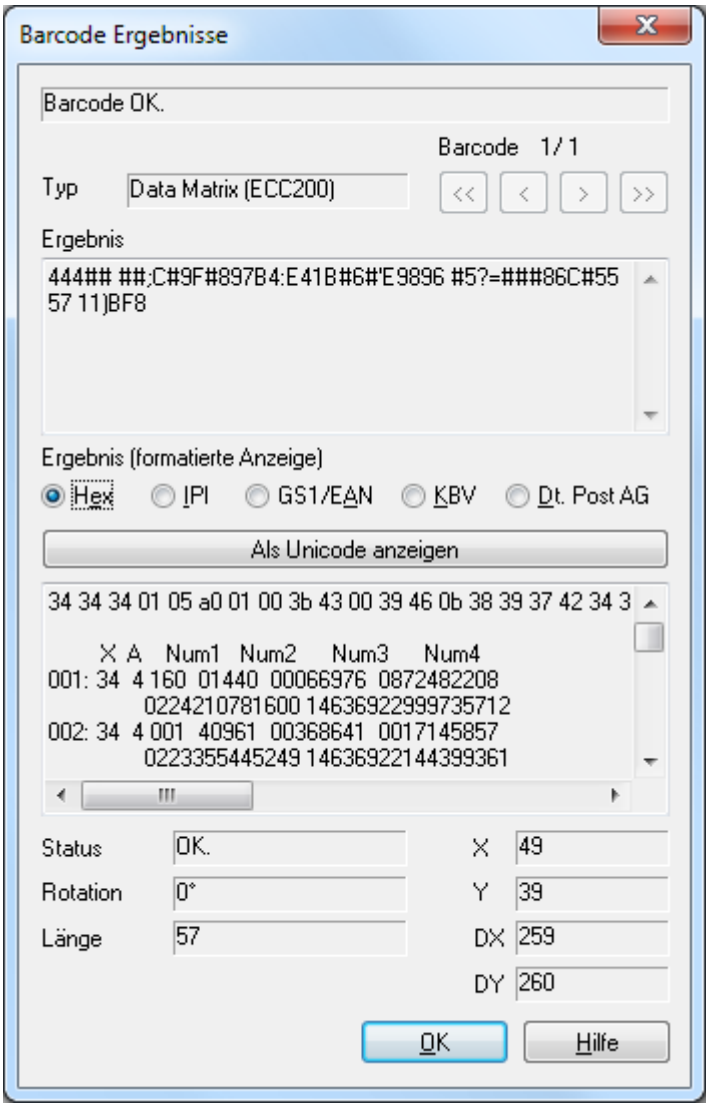


BC_INTERLEAVED25 BC_INDUSTRIE25 BC_25_IATA BC_25_3MATRIX BC_25_3DATALOGIC BC_25_BCDMATRIX BC_25_INVERTIERT BC_EAN13 BC_UPC_A BC_EAN8 BC_UPC_E	BC_CODE39 BC_CODE39EXT BC_CODE32 BC_CODE93 BC_CODE93EXT BC_CODABAR BC_CODE128 BC_EAN128 BC_CODE11	BC_GS1_EAN13 BC_GS1_DATABAR BC_GS1_DATABAR_STACKED BC_GS1_DATABAR_EXP BC_GS1_DATABAR_EXP_STACKED BC_GS1_DATABAR_LIMITED	BC_CODABLOCK BC_PDF417 BC_DATAMATRIX BC_QR_CODE BC_AZTEC BC_PATCHCODE

BC_POSTNET_AUSTRALIAN BC_POSTNET_CPC	BC_POSTNET_ROYAL BC_POSTNET_KIX BC_POSTNET_IMB	BC_POSTNET_ZIP BC_POSTNET_PLANET

Tabelle: Konstanten zur Typauswahl

6.3 Barcode Ergebnisse



In nebenstehendem Fenster werden die Ergebnisse angezeigt.

Mit den Schaltflächen "<<", "<",">", ">>" kann zwischen den einzelnen Ergebnissen umgeschaltet werden.
Neben Typ und Wert werden auch einige weitere Informationen zur Lage, Länge und Rotation angezeigt.
In der unteren Listbox werden die Ergebnisse in Hex angezeigt. Dies ist bei 2D Barcodes nützlich, die auch nicht druckbare Zeichen enthalten können.
Umschalten ermöglicht für Spezialfälle sogar die feldbezogene Dateninterpretation für normierte Anwendungsfälle.

Wird ein Barcode nicht gefunden, überprüfen Sie die Einstellungen.

6.4 Barcode Analyse

Die Barcodeanalyse führt eine Abfolge von Barcodeerkennungen in schneller Folge automatisch durch. Dabei werden Schritt für Schritt einzelne Parameter variiert, bis ein Barcode gefunden wurde.

Wir haben in diese Analyse unsere Erfahrungen einfließen lassen, was die häufigste Ursache einer Nicht-Erkennung angeht, die Variation der Parameter (z.B. der Ruhezone) spiegelt dies wieder.

7 Library mit Pointer (p_lib)

7.1 Umfang der p_lib Schnittstelle

ACHTUNG	Bei Sprung auf QS-Barcode SDK Version 4.6 wurden alle Schnittstellen überarbeitet. Alle Programmbeispiele und Beschreibungen basieren auf den neuen Versionen der Schnittstellen. In Kapitel "16.6 Update-Hinweise" erfahren Sie mehr zu der alten und neuen Version der Schnittstellen.

Der p_lib Schnittstelle werden ein Bild oder ein Bildausschnitt sowie ein Parameterblock übergeben. Der Parameterblock enthält einige Informationen über das übergebene Bild sowie Steuerparameter für die Barcodeerkennung.
Als Rückgabe erhält man einen Rückgabewert, der den generellen Status nach der Barcode-Erkennung meldet. Weiterhin werden eine oder mehrere Ergebnis-Strukturen zurückgegeben, für jeden erkannten Barcode eine Struktur.
Im Falle eines Fehlers zeigt der Rückgabewert diesen detailliert an. In den Ergebnis-Strukturen werden zudem Statusinformationen zu der Erkennung des jeweiligen Barcodes angezeigt.
Alle Funktionen sind Multithreading-fähig.

Die **p_lib** Schnittstelle bietet folgende Funktionen:

```
int QSBarcode2( BarcodeParam2 *pBarcodeParam2 );
```

Diese Funktion führt die Barcodeerkennung durch und ist somit die Kernfunktion der p_lib Schnittstelle.

```
int QSBarcode3( BarcodeParam2 *pBarcodeParam2, char *szIniFile );
```

Diese Funktion führt ebenfalls die Barcodeerkennung durch. Im Gegensatz zu " `QSBarcode2`", bei der die "qsbc.ini" für Bildverbesserungen gesucht wird, kann hier explizit eine "qsbc.ini" übergeben werden, die nur für diesen Aufruf gültig ist.

```
int QSBarcode4( BarcodeParam2 *pBarcodeParam2, char *szIniFile, char
*szLicenseFile );
```

Dieser Funktion der Barcodeerkennung kann eine "qsbc.ini" für die Bildverbesserungen sowie eine Lizenzdatei (QSBC.LIC) direkt übergeben werden. Der Pfad auf die Lizenzdatei muss voll qualifiziert übergeben werden. Wenn keine „qsbc.ini“ übergeben wird, wird dieser Parameter ignoriert.

```
int QSBarcode5( BarcodeParam2 *pBarcodeParam2, char *szIniFile, char
*szLicenseFile, AdvancedMode *pBC_AdvancedMode);
```

Dieser Funktion der Barcodeerkennung kann AdvancedMode Struktur zur erweiterten Suche übergeben werden (siehe Kapitel 12, Erweiterte Suche).

```
int QSBarcode6( BarcodeParam2 *pBarcodeParam2, char *szIniFile, char
*szLicenseFile, AdvancedMode *pBC_AdvancedMode, char *szImaging);
```

Dieser Funktion kann zusätzlich ein String übergeben werden, der einzelne Bildbearbeitungsbefehle enthält, die über die Graphlib vor der Barcodeerkennung durchgeführt werden sollen. (siehe Kapitel xx, Bildbearbeitung).

```
int QSLicense( void );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Lizenz von QS-Barcode SDK.

```
int QSLicense1( char *szLicenseFile );
```

Diese Hilfsfunktion ermittelt die Lizenz von QS-Barcode SDK bei direkter Übergabe der Lizenzdatei.

```
void QSVersion( char * szVersion );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Version von QS-Barcode SDK

Zur Wahrung der Abwärtskompatibilität wird weiterhin die alte Funktion zur Barcodeerkennung angeboten:

```
int QSBarcode( BarcodeParam *pBarcodeParam );
```

Die Aufrufparameter, das Verhalten und die Nutzung der Funktionen wird im Folgenden erklärt.

7.2 Typen und Strukturen der p_lib Schnittstelle

Informationen zu den einzelnen Variablen der Strukturen finden Sie im Kapitel "11 11 Parameter".

```
typedef struct BarcodeParam2_tag
{
    char              szBC_Version [30];
    PBarcodeData2     pBC_BarcodeData2;
    LPSTR             lpstrBC_Image;
    int               iBC_Width;
    int               iBC_Height;
    int               iBC_StartX;
    int               iBC_StartY;
    int               iBC_SizeX;
    int               iBC_SizeY;
    RotInfo           BC_RotInfo;

    void FAR          *pBC_Memory;
    long              lBC_MemorySize;
```

```
    BarcodeResult2    *pbrBC_Result2;
    int               iBC_ResultCount;
    char              szBC_SubstitutionString [10];
    char              szBC_SubstitutionChar;

    int               iBC_Debug;
    HFILE             hBC_ErrorFile;
    HANDLE            hBC_Reserve;
} BarcodeParam2;

typedef struct BarcodeData2_tag
{
    int               iBC_Type;
    int               iBC_Type2;
    int               iBC_Length;
    int               iBC_Checksum;
    int               iBC_Orientation;
    int               iBC_ReadMultiple;
    int               iBC_LightMargin;
    int               iBC_ScanDistance;
    int               iBC_Percent;
    int               iBC_ScanDistBarcode;
    int               iBC_MaxHeight;
    int               iBC_MinHeight;
    int               iBC_MaxNoVal;
    int               iBC_Tolerance;
    int               iLaengeVon;
    int               iLaengeBis;
    PBarcodeData2     pBC_NextBarcodeData2;
} BarcodeData2;


typedef struct BarcodeResult2_tag
{
    int               iBC_Type;
    int               iBC_Type2;
    int               iBC_Status;
    char              szBC_Barcode[64];
    int               iBC_StartX;
    int               iBC_StartY;
    int               iBC_SizeX;
    int               iBC_SizeY;
    int               iBC_Orientation;
    TwoDimResult      BC_TwoDimRes;
} BarcodeResult2;


typedef struct tag_RotInfo
{
    double            dBC_cos;
    double            dBC_sin;
    int               iBC_BMoffsetX;
    int               iBC_BMoffsetY;
    BOOL              fBC_bRotated;
    int               iBC_offsetX;
    int               iBC_offsetY;
    double            dBC_XKorr;
```

```
    double              dBC_YKorr;
} RotInfo;


typedef struct TwoDimResult_tag
{
    HANDLE              hBC_TwoDimRes;
    int                 iBC_TwoDimLen;
    int                 iBC_TwoDimRows;
    int                 iBC_TwoDimCols;
    int                 iBC_PdfECL;
} TwoDimResult;


typedef struct AdvancedMode_tag
{
    int                 AdvancedSearch;
    int                 DynamicThreshold;
    int                 iBC_Threshold;
    int                 iBC_RemovePixel;
    int                 iBC_LightMargin1;
    int                 iBC_LightMargin2;
    int                 iBC_LightMargin3;
    int                 iBC_ScanDistance2;
    int                 iBC_ScanDistBarcode2;
} AdvancedMode;
```

7.3 Funktionen der p_lib Schnittstelle

7.3.1 Funktion: QSBarcode2

Überblick

`QSBarcode2` führt die Barcode-Erkennung auf einem schwarz/weiß Bild durch und liefert Informationen über die erkannten Barcodes.

```
int QSBarcode2( BarcodeParam2 *pBarcodeParam2 );
```

Parameter

`*pBarcodeParam2` [in/out] Dieser Zeiger auf eine `BarcodeParam2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `pBC_Memory` gespeichert werden können.

oder es wird ein Fehlercode zurückgegeben:

```
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

`pBarcodeParam2->pbrBC_Result2` Dieser Zeiger zeigt, wenn mindestens ein Barcode erkannt wurde, auf die erste `BarcodeResult2` Struktur mit den Ergebnis-Informationen. Über `pBC_NextBarcodeData2` kann jeweils auf das nächste Barcode-Ergebnis zugegriffen werden.

Erläuterungen

Empfohlenes Vorgehen zur Befüllung von `pBarcodeParam2` vor dem Aufruf von `QSBarcode2`:

1. Die komplette Struktur mit `0` initialisieren.
2. `BC_RotInfo.dBC_cos` muss auf `1.0` gesetzt werden, ansonsten können Sie die Struktur `BC_RotInfo` aber getrost ignorieren.
3. Die Variablen der `pBC_BarcodeData2` Struktur müssen entweder mit den Default-Werten für Barcode-Erkennung gefüllt werden oder individuell konfiguriert werden je nachdem was für Barcodes erkannt werden sollen. Beachten Sie dazu die Erläuterungen zu den einzelnen Parametern weiter hinten in dieser Dokumentation und auch die Programmbeispiele.

4. `pBC_Memory` muss einen Zeiger auf einen allokierten Speicherbereich enthalten, `lBC_MemorySize` muss die Größe dieses Speicherbereichs in Byte beinhalten. `pBC_Memory` wird von `QSBarcode2` genutzt, um dort die `BarcodeResult2` Strukturen mit den Ergebnis-Informationen abzulegen. Daher muss die Größe des Speicherbereichs auch ein Vielfaches der Größe von `BarcodeResult2` sein.
5. `lpstrBC_Image` muss einen Zeiger auf Bildinformationen enthalten. Für diese Bildinformationen gelten folgende Formatierungsverpflichtungen:
   - Pro Pixel ein Bit, weiß=1 und schwarz=0
   - Erstes Bit ist das Bildpixel ganz oben links, dann wird nach rechts gelaufen
   - Jede Bildzeile ist auf 32 Bit normiert, d.h. eine Bildzeile ist ein Vielfaches von 32 Bit breit
6. `iBC_Width` und `iBC_Height` müssen passend zu `lpstrBC_Image` gefüllt sein (Pixel-Angaben)

Um nach der Funktionsausführung die Barcode-Ergebnisse auszulesen, greifen Sie auf die Struktur `pbrBC_Result2` und dann solange auf die nächste Struktur in der verketteten Liste unter `pBC_NextBarcodeData2` zu, bis dieser Zeiger `NULL` ist.

Für **2-dimensionale Barcodes** gibt es einige Besonderheiten zu beachten:
Der Barcode-Inhalt ist hier nicht in `szBC_Barcode[64]` zu finden, diese Variable ist für 2D Barcodes nicht gefüllt. Der Grund hierfür ist, dass 2D Barcodes auch deutlich längere Ergebnisse haben können und das zudem alle möglichen Binärdaten enthalten sein können, auch das 0-Byte, welches in einem C-String als String-Ende interpretiert werden würde.
Bei 2D Barcodes sind die Variablen in der erweiterten Struktur `BC_TwoDimRes` gefüllt, `hBC_TwoDimRes` enthält dabei einen Windows Handle auf einen Speicherbereich, in dem dann das komplette Ergebnis der Länge `iBC_TwoDimLen` steht.
Details zum Auslesen des Windows Handle entnehmen Sie bitte den Programmbeispielen.

7.3.2 Funktion: QSBarcode3

Überblick

`QSBarcode3` führt die Barcode-Erkennung auf einem schwarz/weiß Bild durch und liefert Informationen über die erkannten Barcodes. Zusätzlich kann dieser Funktion eine "qsbc.ini" übergeben werden.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSBarcode3( BarcodeParam2 *pBarcodeParam2, char *szInifile );
```

Parameter

`*pBarcodeParam2` [in/out] Dieser Zeiger auf eine `BarcodeParam2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der

"qsbc.ini".

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `pBC_Memory` gespeichert werden können.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

`pBarcodeParam2->pbrBC_Result2` Dieser Zeiger zeigt, wenn mindestens ein Barcode erkannt wurde, auf die erste `BarcodeResult2` Struktur mit den Ergebnis-Informationen. Über `pBC_NextBarcodeData2` kann jeweils auf das nächste Barcode-Ergebnis zugegriffen werden.

Erläuterungen

siehe 7.3.1 Funktion: QSBarcode2

7.3.3 Funktion: QSBarcode4

Überblick

`QSBarcode4` führt die Barcode-Erkennung auf einem schwarz/weiß Bild durch und liefert Informationen über die erkannten Barcodes. Zusätzlich kann dieser Funktion eine "qsbc.ini" und die Lizenzdatei "qsbc.lic" übergeben werden.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSBarcode4( BarcodeParam2 *pBarcodeParam2, char *szInifile char
*szLicenseFile);
```

Parameter

`*pBarcodeParam2` [in/out] Dieser Zeiger auf eine `BarcodeParam2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `pBC_Memory` gespeichert werden können.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

`pBarcodeParam2->pbrBC_Result2` Dieser Zeiger zeigt, wenn mindestens ein Barcode erkannt wurde, auf die erste `BarcodeResult2` Struktur mit den Ergebnis-Informationen. Über `pBC_NextBarcodeData2` kann jeweils auf das nächste Barcode-Ergebnis zugegriffen werden.

Erläuterungen

siehe 7.3.1 Funktion: QSBarcode2

7.3.4 Funktion: QSBarcode5

Überblick

`QSBarcode5` führt die Barcode-Erkennung auf einem schwarz/weiß Bild durch und liefert Informationen über die erkannten Barcodes Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini") sowie eine AdvancedMode-Struktur zur erweiterten Barcodeerkennung übergeben werden.

```
int QSBarcode5( BarcodeParam2 *pBarcodeParam2, char *szInifile char
*szLicenseFile, AdvancedMode *pBC_AdvancedMode);
```

Parameter

`*pBarcodeParam2` [in/out] Dieser Zeiger auf eine `BarcodeParam2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `pBC_Memory` gespeichert werden können.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

`pBarcodeParam2->pbrBC_Result2` Dieser Zeiger zeigt, wenn mindestens ein Barcode erkannt wurde, auf die erste `BarcodeResult2` Struktur mit den Ergebnis-Informationen. Über `pBC_NextBarcodeData2` kann jeweils auf das nächste Barcode-Ergebnis zugegriffen werden.

Erläuterungen

siehe 7.3.1 Funktion: QSBarcode2

7.3.5 Funktion: QSBarcode6

Überblick

`QSBarcode6` führt die Barcode-Erkennung auf einem schwarz/weiß Bild durch und liefert Informationen über die erkannten Barcodes Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini") sowie eine AdvancedMode-Struktur zur erweiterten Barcodeerkennung übergeben werden. Zusätzlich kann ein String übergeben werden der einzelne Bildbearbeitungsbefehle enthält, die vor der Barcodeerkennung durchgeführt werden sollen.

```
int QSBarcode6( BarcodeParam2 *pBarcodeParam2, char *szInifile char
*szLicenseFile, AdvancedMode *pBC_AdvancedMode, char *szImaging);
```

Parameter

`*pBarcodeParam2` [in/out] Dieser Zeiger auf eine `BarcodeParam2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten

Parametern gefüllt werden.

`szImaging` [in] Ein null-terminierter String mit durch ein Semikolon getrennte Bildbearbeitungsbefehle.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `pBC_Memory` gespeichert werden können.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
BC_TOO_MANY_THREADS
```

`pBarcodeParam2->pbrBC_Result2` Dieser Zeiger zeigt, wenn mindestens ein Barcode erkannt wurde, auf die erste `BarcodeResult2` Struktur mit den Ergebnis-Informationen. Über `pBC_NextBarcodeData2` kann jeweils auf das nächste Barcode-Ergebnis zugegriffen werden.

Erläuterungen

siehe 7.3.1 Funktion: QSBarcode2

7.3.6 Funktion: QSLicense

Überblick

`QSLicense` ermittelt die auf dem Arbeitsplatz aktuell installierte Lizenz.

```
int QSLicense( void );
```

Parameter

keine

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt an, welche Barcode-Typen lizensiert sind und erkannt werden können. Dieser Wert ist eine Kombination aus den folgenden Konstanten:

```
BC_LIC_DEMO
BC_LIC_LINEAR
BC_LIC_PDF417
BC_LIC_DATAM
BC_LIC_QRCODE
BC_LIC_AZTEC
```

Wenn Sie die Demo-Version des SDK zur Evaluation installiert haben,

ist der Rückgabewert

```
BC_LIC_DEMO + BC_LIC_LINEAR + BC_LIC_PDF417 +
BC_LIC_DATAM + BC_LIC_QRCODE + BC_LIC_AZTEC
```

Derselbe Wert wird zurückgegeben, wenn keine Lizenzdatei gefunden wurde.
Haben Sie ein QS-Barcode SDK Variante LD erworben und die Lizenzdatei ist richtig installiert, ist der Rückgabewert

```
BC_LIC_LINEAR + BC_LIC_DATAM
```

7.3.7 Funktion: QSLicense1

Überblick

`QSLicense1` ermittelt die Lizenz der übergebenen Lizenzdatei.

```
int QSLicense1( char *szLicenseFile );
```

Parameter

`szLicenseFile` [in] Voll qualifizierter Pfad auf die Lizenzdatei.

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt an, welche Barcode-Typen lizensiert sind und erkannt werden können. Dieser Wert ist eine Kombination aus den folgenden Konstanten:

```
BC_LIC_DEMO
BC_LIC_LINEAR
BC_LIC_PDF417
BC_LIC_DATAM
BC_LIC_QRCODE
BC_LIC_AZTEC
```

Wenn Sie die Demo-Version des SDK zur Evaluation installiert haben, ist der Rückgabewert

```
BC_LIC_DEMO + BC_LIC_LINEAR + BC_LIC_PDF417 +
BC_LIC_DATAM + BC_LIC_QRCODE + BC_LIC_AZTEC
```

Derselbe Wert wird zurückgegeben, wenn keine Lizenzdatei gefunden wurde.
Haben Sie ein QS-Barcode SDK Variante LD erworben und die Lizenzdatei ist richtig installiert, ist der Rückgabewert

```
BC_LIC_LINEAR + BC_LIC_DATAM
```

7.3.8 Funktion: QSVersion

Überblick

`QSVersion` ermittelt die auf dem Arbeitsplatz aktuell installierte Version.

```
void QSVersion( char * szVersion );
```

Parameter

`szVersion` [out] wird mit der Version gefüllt und sollte mindestens 11 Zeichen Platz bieten.

Rückgabewerte

`szVersion` wird mit der Version im Format `<Major Version>.<Minor Version>.<Step>.<Build>` zurück, also z.B. `"5.00.0178"`
Die aktuelle Version ist zum Beispiel hilfreich, wenn Sie mit dem QualitySoft Support in Verbindung treten.

7.4 Einbindungsbeispiel für die p_lib Schnittstelle

Dieses (nicht kompilierbare!) Beispiel zeigt die Ansteuerung und den Aufruf der p_lib Hauptfunktion. Kompilierbare Beispieldateien siehe "C-Beispielprogramm".

```
LPSTR         lpBits;     //zeigt auf die Bits der Pixel
Int           nHeight;    //enthält die Höhe des Bildes
Int           nWidth;     //enthält die Breite des Bildes
BarcodeData2  barData2;   //Barcodedaten
BarcodeParam2 barParam2;  //Barcodeparameter
void FAR      *lpResultMem = 0; //Zeiger auf das Ergebnis
int           iNumResults = 10; //Maximale Anzahl Ergebnisse

// Benötigten Speicher bereitstellen
lpResultMem = malloc(sizeof(BarcodeResult2) * iNumResults);
// Zunächst Strukturen leeren
memset(&barParam2, 0, sizeof(BarcodeParam2));
memset(&barData2, 0, sizeof(BarcodeData2));
// die BarcodeData2 - Struktur mit den geeigneten Werten füllen ...
barData2.iBC_Type = BC_INTERLEAVED25 | BC_CODE39;//diese Typen suchen
barData2.iBC_Type2 = BC_NONE;
barData2.iBC_Length = 0;        //Länge unbekannt
barData2.iLaengeVon = 4;        //irgendwo zwischen 4
barData2.iLaengeBis = 12;       //und 12
barData2.iBC_Checksum = 0;
barData2.iBC_Orientation = BC_0 | BC_90 | BC_180 | BC_270;
barData2.iBC_LightMargin = 30;
barData2.iBC_ScanDistance = 15;
barData2.iBC_ScanDistBarcode = 3;
barData2.iBC_ReadMultiple = 1;
barData2.iBC_Percent = 100;
barData2.iBC_MaxNoVal = 10;
barData2.iBC_Tolerance = 20;
barData2.iBC_MinHeight = 15;
barData2.iBC_MaxHeight = 400;
barData2.pBC_NextBarcodeData2 = NULL;
// die BarcodeParam2 - Struktur mit den richtigen Werten füllen ...
barParam2.lpstrBC_Image = lpBits;           //Die Bits der Pixel
barParam2.pBC_BarcodeData2 = &barData2;     //BarcodeEinstellungen
barParam2.iBC_Width = nWidth;               //Breite des Bildes
barParam2.iBC_Height = nHeight;             //Höhe des Bildes
barParam2.BC_RotInfo.dBC_cos = 1.0;   //Wichtig: 1.0!
                                      //Rest der Struktur kann 0 bleiben
barParam2.pBC_Memory = lpResultMem;   //Ergebnisse hierhin
barParam2.lBC_MemorySize = sizeof(BarcodeResult2) * iNumResults;
barParam2.iBC_Debug = 0;
// nun können wir die Suche starten ...
iReturn = QSBarcode2(&barParam2);
//Ergebnisse ausgeben
for(nResult=0;nResult < barParam2.iBC_ResultCount;nResult++)
{
     printf("%i. Ergebnis: %s\n", nResult +1,
           barParam2.pbrBC_Result2[nResult].szBC_Barcode);
}
free(lpResultMem);
lpResultMem = NULL;
```

Tipp!	**Beispielprogramme** Komplette Beispielprogramme für viele Programmiersprachen sind dem SDK beigefügt. Beachten Sie die Dokumentation qsbarsdk_samples.pdf.

7.5 Auslieferungsdateien für die p_lib Schnittstelle

Um **QS-Barcode SDK** über die **p_lib** Schnittstelle zu integrieren, müssen Sie die `QSBarLib.lib` in Ihre Anwendung einbinden und `QSBarLib.h` nutzen, um auf die Definitionen und Konstanten von **QS-Barcode SDK** zuzugreifen.

Eine Installation der "Microsoft Visual C++ 2013 Redistributable " kann eventuell notwendig sein. Dieses Package erhalten Sie in der aktuellen Version auf der Microsoft Internetseite.

☞ Außerdem müssen Sie die **Lizenzdatei** (`QSBC.lic`) im Verzeichnis der Anwendung ausliefern.

☞ Für jeden Arbeitsplatz, auf dem eine Instanz einer die `QSBarLib.lib` enthaltenden Anwendung läuft, benötigen Sie eine **QS-Barcode Runtime-Lizenz**.

Mit der Runtime-Lizenz müssen Sie die folgenden Dateien mit Ihrer Applikation mitliefern:

- **qsbLog.dll**
- **GraphLib.DLL**
- **Graph_ger.DLL**
- **qsImgImp.dll**

8 DLL mit Handle (h_dll)

8.1 Umfang der h_dll Schnittstelle

ACHTUNG	Bei Sprung auf QS-Barcode SDK Version 4.6 wurden alle Schnittstellen überarbeitet. Alle Programmbeispiele und Beschreibungen basieren auf den neuen Versionen der Schnittstellen. In Kapitel "16.6 Update-Hinweise" erfahren Sie mehr zu der alten und neuen Version der Schnittstellen.

Die **h_dll** Schnittstelle liest Barcodes aus dem mittels Windows Handle übergebenen Bild (oder -ausschnitt) aus.
Alle Funktionen sind Multithreading-fähig. Die **h_dll** Schnittstelle bietet folgende Funktionen:

```
int QSReadBarcode2(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults);
```

Durch den Aufruf von `QSReadBarcode2` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`.

```
int QSReadBarcode3(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile);
```

Durch den Aufruf von `QSReadBarcode3` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`.

```
int QSReadBarcode4(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile, char
                   *szLicenseFile);
```

Durch den Aufruf von `QSReadBarcode4` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung sowie eine Lizenzdatei "qsbc.lic" übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt.

```
int QSReadBarcode5(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile, char
```

```
                    *szLicenseFile, AdvancedMode *pBC_AdvancedMode);
```

Durch den Aufruf von `QSReadBarcode5` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung, eine Lizenzdatei "qsbc.lic" sowie eine AdvancedMode Struktur zur erweiterten Suche übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt. Bei Übergabe eines Leerstrings für die Lizenzdatei "qsbc.lic" wird nach dieser Datei im System in den Suchpfaden gesucht.

```
int QSReadBarcode6(HANDLE hDIB, Barcode2 *pBarcode2,
                    int iNumResults, char *szIniFile,
                    char *szLicenseFile, AdvancedMode *pBC_AdvancedMode,
                    char *szImaging);
```

Durch den Aufruf von `QSReadBarcode6` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung, eine Lizenzdatei "qsbc.lic", eine AdvancedMode Struktur zur erweiterten Suche sowie ein String mit Bildbearbeitungsbefehlen übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt. Bei Übergabe eines Leerstrings für die Lizenzdatei "qsbc.lic" wird nach dieser Datei im System in den Suchpfaden gesucht.

```
int QSGetNextBarResult2(BarcodeResult2 *pBarcodeResult2);
```

Um die Barcodeergebnisse zu bekommen wird in einer Schleife die Funktion `QSGetNextBarResult2` aufgerufen. Solange diese Funktion `BC_OK` zurückgibt, sind noch Barcodes vorhanden. Für jeden erkannten Barcode wird eine `BarcodeResult2`-Struktur angelegt.

```
int QSFreeBarResult2(void);
```

Am Schluss der Erkennung **muss** die Funktion `QSFreeResult2` aufgerufen werden, um den intern benutzten Speicher wieder freizugeben.

```
int QSLicense( void );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Lizenz von QS-Barcode SDK

```
int QSLicense1( char *szLicenseFile );
```

Diese Hilfsfunktion ermittelt die Lizenz der übergebenen Lizenzdatei von QS-Barcode SDK

```
void QSVersion( char * szVersion );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Version von QS-Barcode SDK

Zur Wahrung der Abwärtskompatibilität werden weiterhin die alten Funktionen zur Barcodeerkennung angeboten:

```
int QSReadBarcode(HANDLE hDIB, Barcode *pBarcode,
                  int iNumResults);

int QSGetNextBarResult(BarcodeResult *pBarcodeResult);

int QSFreeBarResult(void);
```

Die Aufrufparameter, das Verhalten und die Nutzung der Funktionen werden im Folgenden erklärt.

8.2 Typen und Strukturen der h_dll Schnittstelle

Informationen zu den einzelnen Variablen der Strukturen finden Sie im Kapitel "Parameter".

```
typedef struct Barcode2_tag
{
    int iBC_Type;
    int iBC_Type2;
    int iBC_Length;
    int iBC_Checksum;
    int iBC_Orientation;
    int iBC_ReadMultiple;
    int iBC_LightMargin;
    int iBC_ScanDistance;
    int iBC_Percent;
    int iBC_ScanDistBarcode;
    int iBC_MaxHeight;
    int iBC_MinHeight;
    int iBC_MaxNoVal
    int iBC_Tolerance
    int iBC_StartX;
    int iBC_StartY;
    int iBC_SizeX;
    int iBC_SizeY;
    int iLaengeVon
    int iLaengeBis;
} Barcode2;
```

Die Strukturen `BarcodeResult2, TwoDimResult` und `AdvancedMode` entsprechen denen der **p_lib** und wurden schon im Kapitel "7.2 Typen und Strukturen der p_lib Schnittstelle" beschrieben.

8.3 Funktionen der h_dll Schnittstelle

8.3.1 Funktion: QSReadBarcode2

Überblick

Durch den Aufruf von `QSReadBarcode2` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`.

```
int QSReadBarcode2(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults);
```

Parameter

`hDIB` [in] Ein Windows Handle auf eine Device Independent Bitmap im Windows Standard-Format.

`pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Nutzen Sie die Funktion `QSGetNextBarResult2` um nach der Funktionsausführung die Barcode-Ergebnisse auszulesen.
Die Ergebnisse der Barcode-Erkennung müssen vor einem erneuten Aufruf der Funktion `QSReadBarcode2` ausgelesen werden, ansonsten gehen sie verloren.

Empfohlenes Vorgehen zur Befüllung von `pBarcode2` vor dem Aufruf von `QSReadBarcode2`:

1. Die komplette Struktur mit `0` initialisieren.
2. Die Variablen der Struktur müssen entweder mit den Default-Werten für Barcode-Erkennung gefüllt werden oder individuell konfiguriert werden, je nachdem was für Barcodes erkannt werden sollen. Beachten Sie dazu die Erläuterungen zu den einzelnen Parametern weiter hinten in dieser Dokumentation und auch die Programmbeispiele.

8.3.2 Funktion: QSReadBarcode3

Überblick

Durch den Aufruf von `QSReadBarcode3` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSReadBarcode3(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile);
```

Parameter

`hDIB` [in] Ein Windows Handle auf eine Device Independent Bitmap im Windows Standard-Format.

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini".

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

`BC_QSBCINI_NOT_EXISTS`
`BC_RESOLUTION_LOW`
`BC_WRONG_PARAMETERS`
`BC_LOW_MEMORY`

```
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

8.3.3 Funktion: QSReadBarcode4

Überblick

Durch den Aufruf von `QSReadBarcode4` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine "qsbc.ini" und die Lizenzdatei "qsbc.lic" übergeben werden.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSReadBarcode4(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile);
```

Parameter

`hDIB` [in] Ein Windows Handle auf eine Device Independent Bitmap im Windows Standard-Format.

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

`BC_QSBCINI_NOT_EXISTS`

```
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

8.3.4 Funktion: QSReadBarcode5

Überblick

Durch den Aufruf von `QSReadBarcode5` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini") sowie eine AdvancedMode-Struktur zur erweiterten Barcodeerkennung übergeben werden.

```
int QSReadBarcode5(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile, AdvancedMode *pBC_AdvancedMode);
```

Parameter

`hDIB` [in] Ein Windows Handle auf eine Device Independent Bitmap im Windows Standard-Format.

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt

`BC_MORE_BARCODES` Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

`BC_QSBCINI_NOT_EXISTS`
`BC_RESOLUTION_LOW`
`BC_WRONG_PARAMETERS`
`BC_LOW_MEMORY`
`BC_NO_MEMORY`
`BC_WRONG_VERSION`
`BC_INVALID_BMP`
`BC_NO_SUCH_PAGE`

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

8.3.5 Funktion: QSReadBarcode6

Überblick

Durch den Aufruf von `QSReadBarcode6` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini") eine AdvancedMode-Struktur zur erweiterten Barcodeerkennung sowie ein String mit Bildbearbeitungsbefehlen übergeben werden.

```
int QSReadBarcode6(HANDLE hDIB, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile, AdvancedMode *pBC_AdvancedMode,
                   char *szImaging);
```

Parameter

`hDIB` [in] Ein Windows Handle auf eine Device Independent Bitmap im Windows Standard-Format.

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szImaging` [in] Ein null-terminierter String mit durch ein Semikolon getrennte Bildbearbeitungsbefehle.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
BC_TOO_MANY_THREADS
```

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

8.3.6 Funktion: QSGetNextBarResult2

Überblick

Um die Barcodeergebnisse zu bekommen, die durch den Aufruf von `QSReadBarcode2` ermittelt wurden, rufen Sie die Funktion `QSGetNextBarResult2` auf. Solange diese Funktion `BC_OK` zurückgibt, sind noch Barcodes vorhanden. Jeder neue Aufruf liefert das jeweils nächste Ergebnis.

```
int QSGetNextBarResult2(BarcodeResult2 *pBarcodeResult2);
```

Parameter

`pBarcodeResult2` [in/out] Zeiger auf eine `BarcodeResult2` Struktur. Hier werden die Informationen über das Barcode-Ergebnis von `QSGetNextBarResult2` hineingeschrieben.

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein weiterer Barcode erkannt
`BC_NO_BARCODE`	Keine weiteren Barcodes

Erläuterungen

Für **2-dimensionale Barcodes** gibt es einige Besonderheiten zu beachten:
Der Barcode-Inhalt ist hier nicht in `szBC_Barcode[64]` zu finden, diese Variable ist für 2D Barcodes nicht gefüllt. Der Grund hierfür ist, dass 2D Barcodes auch deutlich längere Ergebnisse haben können und das zudem alle möglichen Binärdaten

enthalten sein können, auch das 0-Byte, welches in einem C-String als String-Ende interpretiert werden würde.
Bei 2D Barcodes sind die Variablen in der erweiterten Struktur `BC_TwoDimRes` gefüllt, `hBC_TwoDimRes` enthält dabei einen Windows Handle auf einen Speicherbereich, in dem dann das komplette Ergebnis der Länge `iBC_TwoDimLen` steht.
Details zum Auslesen des Windows Handle entnehmen Sie bitte den Programmbeispielen.

8.3.7 Funktion: QSFreeBarResult2

Überblick

Am Schluss der Erkennung **muss** die Funktion `QSFreeResult2` aufgerufen werden, um den intern benutzten Speicher wieder freizugeben.

```
int QSFreeBarResult2(void);
```

Parameter

keine

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt immer den folgenden Wert zurück:

`BC_OK`	Freigabe erfolgreich

8.3.8 Funktion: QSLicense

Siehe Kapitel "7.3.6 Funktion: QSLicense"

8.3.9 Funktion: QSLicense1

Siehe Kapitel "7.3.7 Funktion: QSLicense1"

8.3.10 Funktion: QSVersion

Siehe Kapitel "7.3.8 Funktion: QSVersion"

8.4 Einbindungsbeispiel für die h_dll Schnittstelle

Dieses (nicht kompilierbare!) Beispiel zeigt die Einbindung der h_dll Schnittstelle in den Code. Kompilierbare Beispieldateien siehe "C-Beispielprogramm".

```
int             iReturn,nResult,iNumResults;
Barcode2        pBarcode2;
BarcodeResult2  pBarcodeResult2;
//hDIB = Handle auf die Bits des Bildes
if(hDIB==NULL)
    return 0;
// Zunächst Struktur leeren
memset(&pBarcode2, 0, sizeof(Barcode2));
memset(&pBarcodeResult2, 0, sizeof(BarcodeResult2));
// Die Barcode2-Struktur mit geeigneten Werten füllen.
pBarcode2.iBC_Type = BC_INTERLEAVED25 | BC_CODE39;
pBarcode2.iBC_Type2 = BC_NONE;
pBarcode2.iBC_Length = 0;
pBarcode2.iBC_Checksum = 0;
pBarcode2.iBC_Orientation = BC_0 | BC_90 | BC_180 | BC_270;
pBarcode2.iBC_ReadMultiple = 1;
pBarcode2.iBC_LightMargin = 18;
pBarcode2.iBC_ScanDistance = 15;
pBarcode2.iBC_Percent = 100;
pBarcode2.iBC_ScanDistBarcode = 3;
pBarcode2.iBC_MaxHeight = 400;
pBarcode2.iBC_MinHeight = 15;
pBarcode2.iBC_MaxNoVal = 10;
pBarcode2.iBC_Tolerance = 10;
pBarcode2.iBC_StartX = 0;
pBarcode2.iBC_StartY = 0;
pBarcode2.iBC_SizeX = 0;
pBarcode2.iBC_SizeY = 0;
pBarcode2.iLaengeVon = 4;
pBarcode2.iLaengeBis = 12;
iNumResults=10;
// jetzt können wir die Suche beginnen
iReturn = QSReadBarcode2(hDIB, &pBarcode2, iNumResults2);
//Ergebnisse ausgeben
for(nResult=0;nResult < iNumResults;nResult++)
{
    iReturn = QSGetNextBarResult2(&pBarcodeResult2);
    if(iReturn!=BC_NO_BARCODE)
        printf("%i. Ergebnis: %s\n",
               nResult+1, pBarcodeResult2.szBC_Barcode);
    else
        break;
}
// Nicht vergessen, die Ergebnisse freizugeben!
GlobalUnlock(hDIB);
GlobalFree(hDIB);
iReturn = QSFreeBarResult2();
```

Tipp!	**Beispielprogramme** Komplette Beispielprogramme für viele Programmiersprachen sind dem SDK beigefügt. Beachten Sie die Dokumentation qsbarsdk_samples.pdf.

8.5 Auslieferungsdateien für die h_dll Schnittstelle

Mit der Runtime-Lizenz müssen Sie die folgenden Dateien mit Ihrer Applikation mitliefern:

- **QSBarDll_H.dll**
- **GraphLib.DLL**
- **Graph_ger.DLL**
- **QSBImage.DLL**
- **FreeImage.dll**
- **qsImgImp.dll**
- **qsblog.dll**

Eine Installation der "Microsoft Visual C++ 2013 Redistributable" kann eventuell notwendig sein. Dieses Package erhalten Sie in der aktuellen Version auf der Microsoft Internetseite.

Diese DLLs müssen entweder im Pfad der Applikation oder in einem anderen Pfad sein, in dem das System DLLs findet.

Außerdem müssen Sie die **Lizenzdatei** (`QSBC.lic`) im Verzeichnis der Anwendung ausliefern.

Für jeden Arbeitsplatz, auf dem eine Instanz Ihrer Anwendung läuft, benötigen Sie eine **QS-Barcode Runtime-Lizenz**.

9 DLL mit Datei (f_dll)

9.1 Umfang der f_dll Schnittstelle

ACHTUNG	Bei Sprung auf QS-Barcode SDK Version 4.6 wurden alle Schnittstellen überarbeitet. Alle Programmbeispiele und Beschreibungen basieren auf den neuen Versionen der Schnittstellen. In Kapitel "16.6 Update-Hinweise" erfahren Sie mehr zu der alten und neuen Version der Schnittstellen.

Die **f_dll** Schnittstelle liest Barcodes aus einer Bilddatei oder aus einem Adobe PDF Dokument.
Alle Funktionen sind Multithreading-fähig. Die **f_dll** Schnittstelle bietet folgende Funktionen:

```
int QSReadBarcode2(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults);
```

Durch den Aufruf von `QSReadBarcode2` werden die Barcodes aus der angegebenen Datei ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`.

```
int QSReadBarcode3(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile);
```

Durch den Aufruf von `QSReadBarcode3` werden die Barcodes aus der angegebenen Datei ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`.

```
int QSReadBarcode4(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile, char
                   *szLicenseFile);
```

Durch den Aufruf von `QSReadBarcode4` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung sowie eine Lizenzdatei "qsbc.lic" übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt.

```
int QSReadBarcode5(char *szImageName, Barcode2 *pBarcode2,
int iNumResults, char *szIniFile, char *szLicenseFile, AdvancedMode
*pBC_AdvancedMode);
```

Durch den Aufruf von `QSReadBarcode5` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung, eine Lizenzdatei "qsbc.lic" sowie eine AdvancedMode Struktur zur erweiterten Suche übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt. Bei Übergabe eines Leerstrings für die Lizenzdatei "qsbc.lic" wird nach dieser Datei im System in den Suchpfaden gesucht.

```
int QSReadBarcode6(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile, char
                   *szLicenseFile, AdvancedMode *pBC_AdvancedMode,
                   char *szImaging);
```

Durch den Aufruf von `QSReadBarcode6` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Es werden hierbei maximal `iNumResults` Ergebnisse gespeichert. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung, eine Lizenzdatei "qsbc.lic" , eine AdvancedMode Struktur zur erweiterten Suche sowie ein String mit Befehlen zur Bildbearbeitung übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`. Wenn Sie einen Leerstring für die "qsbc.ini" übergeben, wird dieser Parameter nicht berücksichtigt. Bei Übergabe eines Leerstrings für die Lizenzdatei "qsbc.lic" wird nach dieser Datei im System in den Suchpfaden gesucht.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

```
int QSGetNextBarResult2(BarcodeResult2 *pBarcodeResult2);
```

Um die Barcodeergebnisse zu bekommen wird in einer Schleife die Funktion `QSGetNextBarResult2` aufgerufen. Solange diese Funktion `BC_OK` zurückgibt, sind noch Barcodes vorhanden. Für jeden erkannten Barcode wird eine `BarcodeResult2`-Struktur angelegt.

```
int QSFreeBarResult2(void);
```

Am Schluss der Erkennung **muss** die Funktion `QSFreeResult2` aufgerufen werden, um den intern benutzten Speicher wieder freizugeben.

```
int QSLicense( void );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Lizenz von QS-Barcode SDK

```
int QSLicense1( char *szLicenseFile );
```

Diese Hilfsfunktion ermittelt die Lizenz von QS-Barcode SDK bei direkter Übergabe der Lizenzdatei.

```
void QSVersion( char * szVersion );
```

Diese Hilfsfunktion ermittelt die auf dem Arbeitsplatz aktuell installierte Version von QS-Barcode SDK

Zur Wahrung der Abwärtskompatibilität werden weiterhin die alten Funktionen zur Barcodeerkennung angeboten:

```
int QSReadBarcode(char *szImageName, Barcode *pBarcode,
                  int iNumResults);

int QSGetNextBarResult(BarcodeResult *pBarcodeResult);

int QSFreeBarResult(void);
```

Die Aufrufparameter, das Verhalten und die Nutzung der Funktionen wird im Folgenden erklärt.

9.2 Typen und Strukturen der f_dll Schnittstelle

Informationen zu den einzelnen Variablen der Strukturen finden Sie im Kapitel "11 Parameter".

Die Struktur **`Barcode2_tag`** entspricht der der **h_dll** und wurde schon im Kapitel "8.2 Typen und Strukturen der h_dll Schnittstelle" beschrieben.

Die Strukturen `BarcodeResult2`,`TwoDimResult` und `AdvancedMode` entsprechen denen der **p_lib** und wurden schon im Kapitel "7.2 Typen und Strukturen der p_lib Schnittstelle" beschrieben.

9.3 Funktionen der f_dll Schnittstelle

9.3.1 Funktion: QSReadBarcode2

Überblick

Durch den Aufruf von `QSReadBarcode2` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`.

```
int QSReadBarcode2(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults);
```

Parameter

`szImageName` [in] Ein null-terminierter String auf den Dateinamen einer Bilddatei oder eines Adobe PDF-Dokuments. Bei mehrseiten Dateien werden normalerweise nur Barcodes auf der ersten Seite gelesen, Sie können aber auch gezielt andere Seiten ansteuern.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

`pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Nutzen Sie die Funktion `QSGetNextBarResult2` um nach der Funktionsausführung die Barcode-Ergebnisse auszulesen.
Die Ergebnisse der Barcode-Erkennung müssen vor einem erneuten Aufruf der Funktion `QSReadBarcode2` ausgelesen werden, ansonsten gehen sie verloren.

Empfohlenes Vorgehen zur Befüllung von `pBarcode2` vor dem Aufruf von `QSReadBarcode2`:

1. Die komplette Struktur mit `0` initialisieren.
2. Die Variablen der Struktur müssen entweder mit den Detault-Werten für Barcode-Erkennung gefüllt werden oder individuell konfiguriert werden je nachdem was für Barcodes erkannt werden sollen. Beachten Sie dazu die Erläuterungen zu den einzelnen Parametern weiter hinten in dieser Dokumentation und auch die Programmbeispiele.

9.3.2 Funktion: QSReadBarcode3

Überblick

Durch den Aufruf von `QSReadBarcode3` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine separate "qsbc.ini" zur Bildoptimierung übergeben werden. Wenn die angegebene "qsbc.ini" nicht existiert, so ist der Rückgabewert `BC_QSBCINI_NOT_EXISTS`.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSReadBarcode3(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile);
```

Parameter

`szImageName` [in] Ein null-terminierter String auf den Dateinamen einer Bilddatei oder eines Adobe PDF-Dokuments. Bei mehrseiten Dateien werden normalerweise nur Barcodes auf der ersten Seite gelesen, Sie können aber auch gezielt andere Seiten ansteuern.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini".

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Siehe 9.3.1 Funktion: QSReadBarcode2

9.3.3 Funktion: QSReadBarcode4

Überblick

Durch den Aufruf von `QSReadBarcode4` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine "qsbc.ini" und die Lizenzdatei "qsbc.lic" übergeben werden.
(Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini")

```
int QSReadBarcode4(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile);
```

Parameter

`szImageName` [in] Ein null-terminierter String auf den Dateinamen einer Bilddatei oder eines Adobe PDF-Dokuments. Bei mehrseiten Dateien werden normalerweise nur Barcodes auf der ersten Seite gelesen, Sie können aber auch gezielt andere Seiten ansteuern.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Siehe 9.3.1 Funktion: QSReadBarcode2

9.3.4 Funktion: QSReadBarcode5

Überblick

Durch den Aufruf von `QSReadBarcode5` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini") sowie eine Advanced-Struktur zur erweiterten Barcodeerkennung übergeben werden.

```
int QSReadBarcode5(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile, AdvancedMode *pBC_AdvancedMode);
```

Parameter

`szImageName` [in] Ein null-terminierter String auf den Dateinamen einer Bilddatei oder eines Adobe PDF-Dokuments. Bei mehrseiten Dateien werden normalerweise nur Barcodes auf der ersten Seite gelesen, Sie können aber auch gezielt andere Seiten ansteuern.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

```
BC_QSBCINI_NOT_EXISTS
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
```

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

9.3.5 Funktion: QSReadBarcode6

Überblick

Durch den Aufruf von `QSReadBarcode6` werden die Barcodes aus dem angegebenen Bild ausgelesen und in einer internen Struktur abgelegt. Verlief die Barcodeerkennung fehlerfrei und es wurden Barcodes gefunden, so ist der Rückgabewert der Funktion `BC_OK`. Wurden keine Barcodes gefunden, so ist der Rückgabewert `BC_NO_BARCODE`. Zusätzlich kann dieser Funktion eine Lizenzdatei "qsbc.lic", eine "qsbc.ini", (Erläuterungen siehe Kapitel 16.4 Bildverbesserung über die "qsbc.ini"), eine Advanced-Struktur zur erweiterten Barcodeerkennung sowie ein String mit Befehlen zur Bildbearbeitung übergeben werden.

```
int QSReadBarcode6(char *szImageName, Barcode2 *pBarcode2,
                   int iNumResults, char *szIniFile char
                   *szLicenseFile, AdvancedMode *pBC_AdvancedMode,
                   char *szImaging);
```

Parameter

`szImageName` [in] Ein null-terminierter String auf den Dateinamen einer Bilddatei oder eines Adobe PDF-Dokuments. Bei mehrseiten Dateien werden normalerweise nur Barcodes auf der ersten Seite gelesen, Sie können aber auch gezielt andere Seiten ansteuern.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

`*pBarcode2` [in] Dieser Zeiger auf eine `Barcode2` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`iNumResults` [in] Es werden maximal `iNumResults` Ergebnisse in der internen Struktur für Barcode-Ergebnisse gespeichert, auch wenn mehr Barcodes auf dem Bild vorhanden sind.
Wichtig: Wählen Sie für `iNumResults` einen Wert deutlich über der maximalen Barcode-Anzahl, die Sie erwarten. Vor allem: Wählen Sie einen Wert deutlich >1 auch wenn Sie wissen, dass auf den Bildern nur ein Barcode vorhanden ist.

`szIniFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.ini". Dieser Parameter kann als Leerstring übergeben werden.

`szLicenseFile` [in] Ein null-terminierter String mit dem voll qualifizierten Dateinamen der "qsbc.lic".

`*pBC_AdvancedMode` [in/out] Dieser Zeiger auf eine `AdvancedMode` Struktur muss vor dem Aufruf der Funktion initialisiert und mit den gewünschten Parametern gefüllt werden.

`szImaging` [in] Ein null-terminierter String mit durch ein Semikolon getrennte Bildbearbeitungsbefehle.

Rückgabewerte

`int` Die Funktion gibt entweder einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein Barcode erkannt
`BC_NO_BARCODE`	Keine Barcodes erkannt
`BC_MORE_BARCODES`	Es wurden mehr Barcodes erkannt als in `iNumResults` angegeben.

oder es wird ein Fehlercode zurückgegeben:

`BC_QSBCINI_NOT_EXISTS`

```
BC_RESOLUTION_LOW
BC_WRONG_PARAMETERS
BC_LOW_MEMORY
BC_NO_MEMORY
BC_WRONG_VERSION
BC_INVALID_BMP
BC_NO_SUCH_PAGE
BC_TOO_MANY_THREADS
```

Erläuterungen

Siehe 8.3.1 Funktion: QSReadBarcode2

9.3.6 Funktion: QSGetNextBarResult2

Überblick

Um die Barcodeergebnisse zu bekommen, die durch den Aufruf von `QSReadBarcode2` ermittelt wurden, rufen Sie die Funktion `QSGetNextBarResult2` auf. Solange diese Funktion `BC_OK` zurückgibt, sind noch Barcodes vorhanden. Jeder neue Aufruf liefert das jeweils nächste Ergebnis.

```
int QSGetNextBarResult2(BarcodeResult2 *pBarcodeResult2);
```

Parameter

`pBarcodeResult2` [in/out] Zeiger auf eine `BarcodeResult2` Struktur. Hier werden die Informationen über das Barcode-Ergebnis von `QSGetNextBarResult2` hineingeschrieben.

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt einen der folgenden Werte zurück:

`BC_OK`	Es wurde mindestens ein weiterer Barcode erkannt
`BC_NO_BARCODE`	Keine weiteren Barcodes

Erläuterungen

Für **2-dimensionale Barcodes** gibt es einige Besonderheiten zu beachten:
Der Barcode-Inhalt ist hier nicht in `szBC_Barcode[64]` zu finden, diese Variable ist für 2D Barcodes nicht gefüllt. Der Grund hierfür ist, dass 2D Barcodes auch deutlich längere Ergebnisse haben können und das zudem alle möglichen Binärdaten enthalten sein können, auch das 0-Byte, welches in einem C-String als String-Ende interpretiert werden würde.
Bei 2D Barcodes sind die Variablen in der erweiterten Struktur `BC_TwoDimRes` gefüllt, `hBC_TwoDimRes` enthält dabei einen Windows Handle auf einen Speicherbereich, in dem dann das komplette Ergebnis der Länge `iBC_TwoDimLen` steht.
Details zum Auslesen des Windows Handle entnehmen Sie bitte den Programmbeispielen.

9.3.7 Funktion: QSFreeBarResult2

Überblick

Am Schluss der Erkennung **muss** die Funktion `QSFreeResult2` aufgerufen werden, um den intern benutzten Speicher wieder freizugeben.

```
int QSFreeBarResult2(void);
```

Parameter

keine

Rückgabewerte

`int` Der Rückgabewert der Funktion gibt immer den folgenden Wert zurück:

`BC_OK`	Freigabe erfolgreich

9.3.8 Funktion: QSLicense

Siehe Kapitel "7.3.6 Funktion: QSLicense"

9.3.9 Funktion: QSLicense1

Siehe Kapitel "7.3.7 Funktion: QSLicense1"

9.3.10 Funktion: QSVersion

Siehe Kapitel "7.3.8 Funktion: QSVersion"

9.4 Einbindungsbeispiel für die f_dll Schnittstelle

Dieses (so nicht kompilierbare und gekürzte) Beispiel stammt aus dem Visual Basic Beispielprogramm.

Funktionen deklarieren:

```
Declare Function QSReadBarcode2 Lib "QSBarDll_F.Dll" _
      (ByVal szImageName As String, pBarcode2 As WU_Barcode2, _
       ByVal iNumResults As Long) As Long
Declare Function QSGetNextBarResult2 Lib "QSBarDll_F.Dll" _
      (pBarcodeResult2 As WU_BarcodeResult2) As Long
Declare Function QSFreeBarResult2 Lib "QSBarDll_F.Dll" () As Long
```

Strukturen und Konstanten (Visual Basic Notation der QSBarLib.h)

```
'TwoDim_Result
Type WU_TwoDimResult
    hBC_TwoDimRes As Long
'...
    iBC_PdfECL As Long
End Type

'  Type WU_Barcode2
Type WU_Barcode2
    iBC_Type As Long
    iBC_Type2 As Long
    iBC_Length As Long
'...
    iLaengeBis As Long
End Type
' Type WU_BarcodeResult2
Type WU_BarcodeResult2
    iBC_Type As Long
    iBC_Type2 As Long
    iBC_Status As Long
'...
    sBC_TwoDimRes As WU_TwoDimResult
End Type

'*** Defaultvalues
Global Const BC_LIGHTMARGIN = 30
Global Const BC_SCANDISTANCE = 15
'...

'*** Barcodetypes
Global Const BC_NONE = 0
Global Const BC_INTERLEAVED25 = 1
'...
Global Const BC_PATCHCODE = &H01000000

'*** Checksumtypes
Global Const BC_CHECKNONE = 0
Global Const BC_MOD10 = 1
'...
```

```
'*** Orientations
Global Const BC_0 = 1
Global Const BC_90 = 2
'...

'*** Errorcodes
Global Const BC_OK = 0
Global Const BC_NO_BARCODE = 1
'...
Global Const BC_INVALID_BMP = &H10
Global Const BC_ERROR = -1
```

Aufrufen der DLL-Funktion (aufs Wesentliche beschränkt)

```
Function TestBARCODERead(BMPName As String, newName As String,
                         BCBarcode2 As WU_Barcode2) As String
Dim iNumResults As Integer
Dim BCResult2 As WU_BarcodeResult2
Dim BCResultNr As Byte
Dim a As Integer

    'Werte vor dem Funktionsaufruf definieren
    BCBarcode2.iBC_ReadMultiple = 1
    BCBarcode2.iBC_LightMargin = 24
    '...
    BCResult2.iBC_SizeY = 0
    BCResult2.iBC_Orientation = 0

    iNumResults=10
    'Aufruf mit Bildname, Barcode-Parametern und den gewünschten
    'Ergebnissen.
    QSReadBarcode2 BMPName, BCBarcode2, iNumResults
    'Ergebnis abfragen (in diesem Beispiel nur ein einziges)
    QSGetNextBarResult2 BCResult2

    'Wenn kein Barcoderesult gelesen wurde
    If QSTrim(BCResult2.szBC_Barcode) = "" Then
        TestBARCODERead = 1
        newName = ""
    Else 'sonst Ergebnis in newName zurückliefern
        TestBARCODERead = 0
        newName = QSTrim(BCResult2.szBC_Barcode)
    End If
    'WICHTIG: Ergebnisse freigeben.
    rc = QSFreeBarResult2()

End Function
```

Tipp!	**Beispielprogramme** Komplette Beispielprogramme für viele Programmiersprachen sind dem SDK beigefügt. Beachten Sie die Dokumentation qsbarsdk_samples.pdf.

9.5 Auslieferungsdateien für die f_dll Schnittstelle

Mit der Runtime-Lizenz müssen Sie die folgenden Dateien mitliefern:

- **QSBARDLL_F.DLL**
- **GraphLib.DLL**
- **Graph_ger.DLL**
- **QSBImage.DLL**
- **FreeImage.dll**
- **qsblog.dll**

Wenn Sie auch Adobe PDF Dokumente verarbeiten wollen:

- **pstilldll.dll / pstilldll64.dll**

Eine Installation der "Microsoft Visual C++ 2013 Redistributable" kann eventuell notwendig sein. Dieses Package erhalten Sie in der aktuellen Version auf der Microsoft Internetseite.

Diese DLLs müssen entweder im Pfad der Applikation oder in einem anderen Pfad sein, in dem das System DLLs findet.
Um ganz sicher zu sein dass die Dateien gefunden werden, sollten Sie das Windows Systemverzeichnis wählen.

Außerdem müssen Sie die **Lizenzdatei** (`QSBC.lic`) im Verzeichnis der Anwendung ausliefern.

Für jeden Arbeitsplatz, auf dem eine Instanz Ihrer Anwendung läuft, benötigen Sie eine **QS-Barcode Runtime-Lizenz**.

10 ActiveX mit Datei (f_ocx)

10.1 Beschreibung

ACHTUNG	Bei Sprung auf QS-Barcode SDK Version 4.6 wurden alle Schnittstellen überarbeitet. Alle Programmbeispiele und Beschreibungen basieren auf den neuen Versionen der Schnittstellen. In Kapitel "16.6 Update-Hinweise" erfahren Sie mehr zu der alten und neuen Version der Schnittstellen.

In einem Parameterblock wird dem OCX der Name der Bilddatei übergeben sowie gegebenenfalls ein Bildausschnitt spezifiziert, wenn die Barcode-Erkennung nur in einem Teilbereich des Bildes durchgeführt werden soll. Weiterhin werden Parameter eingestellt, welche die Art der zu suchenden Barcodes konkretisieren.
Die Methode **`getBarcodeResult2`** stößt die Barcodeerkennung an und füllt falls vorhanden direkt die Result-Parameter des OCX mit den Daten zum ersten Barcode-Ergebnis. Als Rückgabe erhält man einen Ergebniswert der besagt, ob ein Barcode gefunden wurde oder nicht.

TIPP	**Unterstützung für mehrseitige Dateien** Gezielt können Seiten in mehrseitigen Dateien angewählt werden. Details siehe Anhang 16.2 MultiPage-Unterstützung **Adobe PDF Dokumente** Beachten Sie Anhang 16.3 Adobe PDF Dokumente – Spezielle Einstellungen

10.2 Einbindung

Auf dem Entwicklungs-Arbeitsplatz, auf dem das QS-Barcode SDK installiert wurde, ist das QS-Barcode OCX bereits installiert und einsatzbereit.
Für die Arbeitsplätze, auf denen Sie Ihre Zielanwendung installieren, wird eine eigene Installationsroutine für das QS-Barcode OCX mitgeliefert (f_ocx\x86\setup).
Nach der Installation des QS-Barcode ActiveX Komponente mit diesem Setup-Programm ist das Control registriert und kann wie gewohnt verwendet werden.

Beachten Sie, dass das Setup nicht nur das OCX registriert, sondern zusätzlich auch noch die benötigte VisualBasic Runtime Umgebung installiert sowie benötigte DLL-Dateien kopiert:
`QSBARDLL_F.DLL, GraphLib.DLL, Graph_ger.DLL, QSBImage.DLL, FreeImage.dll, pstilldll.dll, qsblog.dll`

Sie sollten das OCX und die benötigten DLLs im Systemverzeichnis installieren, so gibt es keine Probleme beim Auffinden von Komponenten.

Die Registrierung des ActiveX Controls kann bei Problemen auch manuell über den Befehl `"regsvr32.exe qsbcocx.ocx"` ausgeführt werden.
Beachten Sie aber, dass dieses Verfahren nicht für PCs ohne installierte Visual Basic Runtime Umgebung geeignet ist.

10.3 Methoden

Der **Aufruf** erfolgt über die Funktion

```
rcBC = QSBCOCX1.getBarcodeResult2()
```

Diese Methode kann solange aufgerufen werden bis keine Ergebnisse mehr vorhanden sind.
Nach der Auswertung der Ergebnis-Parameter ist unbedingt die Methode

```
rcBC = QSBCOCX1.freeBarcodeResult2()
```

für die Freigabe des intern im OCX verwendeten Speichers aufzurufen.

Sprechende Barcodetyp-Namen und Fehlermeldungen erhalten Sie über die folgenden Methoden:

```
QSBCOCX1.getTypeName(QSBCOCX1.ResultType)
QSBCOCX1.getOrientationName(QSBCOCX1.ResultOrientation)
QSBCOCX1.getErrorName(rcBC).
```

Die Eigenschaft `License` zeigt Ihnen an als Kombination der Konstanten `BC_LIC_DEMO, BC_LIC_LINEAR, BC_LIC_PDF417, BC_LIC_DATAM, BC_LIC_AZTEC` welche **Lizenzversion** (Lineare Barcodes (L), DataMatrix (D) und PDF417 (P), QS Code (Q), Aztec (A), gegebenenfalls Demo) Sie verwenden. Die Funktion `getLicenseName` liefert Ihnen die Lizenzbezeichnung als Text.

Besonderes

Mit der Methode `fillResultBinary()` wird ein spezielles Problem behandelt: Je nach Systemumgebung wird die als ASCII-String deklarierte Eigenschaft `ResultBarcode` in einen Unicode-String gewandelt, wobei nicht druckbare Zeichen und Zeichen außerhalb des 7-bit Raums Codepage-abhängig umgesetzt werden. Bei 2D-Barcodes können aber sehr wohl binäre Ergebnisdaten vorliegen, die unverfälscht ausgelesen werden sollen.
`fillResultBinary()` kann dazu wie folgt genutzt werden (Visual Basic Beispiel).

```
Dim b() As Byte
[..]
rcBC = QSBCOCX1.freeBarcodeResult2()
[..]
ReDim b(QSBCOCX1.ResultLength)
Call QSBCOCX1.fillResultBinary(b, QSBCOCX1.ResultLength)
[..]
```

In dem Byte-Array `b` befinden sich nach dem Aufruf die Daten-Bytes des Barcode. Rückgabewert der Methode ist <0 für Fehler und >=0 für eine korrekte Ausführung, wobei der Zahlenwert der Anzahl der gefüllten Bytes des Arrays entspricht.

10.4 Aufruf-Eigenschaften (Properties)

Das Control stellt **Eigenschaften** zur Spezifizierung des gesuchten Barcodes zur Verfügung. Diese müssen vor dem Aufruf der Auswertung gesetzt werden. Eine genaue Spezifizierung von Typ und Länge führt zur schnelleren und sicheren Erkennung des Barcodes.

Beispielcode für das Setzen der Parameter im Code:

```
QSBCOCX1.PictureName = "c:\Qsbcocx\quelle\BarTest.tif"
QSBCOCX1.iNumResults = 10
QSBCOCX1.BarcodeType = BC_CODE39
QSBCOCX1.BarcodeType2 = BC_NONE
QSBCOCX1.BarcodeChecksum = BC_CHECKNONE
QSBCOCX1.BarcodeOrientation = BC_ALL_ORI
QSBCOCX1.BarcodeLength = 0
QSBCOCX1.BarcodeLength_from = 0
QSBCOCX1.BarcodeLength_to = 0

QSBCOCX1.LightMargin = BC_DEFAULTVALUES_TYPES.BC_LIGHTMARGIN
QSBCOCX1.ReadMultiple = BC_DEFAULTVALUES_TYPES.BC_READMULTIPLE
QSBCOCX1.ScanDistance = BC_DEFAULTVALUES_TYPES.BC_SCANDISTANCE
QSBCOCX1.ScanDistBarcode = BC_DEFAULTVALUES_TYPES.BC_SCANDISTBAR
QSBCOCX1.MaxHeight = BC_DEFAULTVALUES_TYPES.BC_MAXHEIGHT
QSBCOCX1.MinHeight = BC_DEFAULTVALUES_TYPES.BC_MINHEIGHT
QSBCOCX1.MaxNoVal = BC_DEFAULTVALUES_TYPES.BC_MAXNOVAL
QSBCOCX1.Tolerance = BC_DEFAULTVALUES_TYPES.BC_TOLERANCE
QSBCOCX1.Percent = BC_DEFAULTVALUES_TYPES.BC_PERCENT

QSBCOCX1.DynamicThreshold = 1

QSBCOCX1.StartX = 0
QSBCOCX1.StartY = 0
QSBCOCX1.SizeX = 0
QSBCOCX1.SizeY = 0
```

Barcode-Properties im Detail:
Die Eigenschaftennamen weichen etwas von denen der Library und DLL ab. Die Eigenschaften sind im Kapitel "11.1 Barcode-Parameter" genau beschrieben.

Name der Eigenschaft	Entspricht
PictureName	Name der auszuwertenden Bilddatei (*.tif) inkl. Pfadname
iNumResults	Anzahl der maximalen Ergebnisse
BarcodeType	iBC_Type
BarcodeType2	iBC_Type2
BarcodeChecksum	iBC_Checksum
BarcodeOrientation	iBC_Orientation
BarcodeLength	iBC_Length
BarcodeLength_from	iLaengeVon
BarcodeLength_to	iLaengeBis
LightMargin	iBC_LightMargin
ReadMultiple	iBC_ReadMultiple
ScanDistance	iBC_ScanDistance
ScanDistBarcode	iBC_ScanDistBarcode
MinHeight / MaxHeight	iBC_MinHeight / iBC_MaxHeight
Percent	iBC_Percent
MaxNoVal	iBC_MaxNoVal
Tolerance	iBC_Tolerance
StartX	iBC_StartX
StartY	iBC_StartY
SizeX	iBC_SizeX
SizeY	iBC_SizeY
AdvancedSearch	AdvancedSearch
DynamicThreshold	DynamicThreshold
Threshold	iBC_Threshold
RemovePixel	iBC_RemovePixel
LightMargin1	iBC_LightMargin1
LightMargin2	iBC_LightMargin2
LightMargin3	iBC_LightMargin3
ScanDistance2	iBC_ScanDistance2
ScanDistBarcode2	iBC_ScanDistBarcode2

10.5 Ergebnis-Eigenschaften

Weitere **Eigenschaften** bilden den Ergebnis-Typ des oder der Barcodes.
Die Initialisierung der Result-Werte muss nicht erfolgen, sie werden automatisch mit den Ergebnissen gefüllt.
Auch hier weichen die Eigenschaftennamen von denen der Library und DLL etwas ab. Die Eigenschaften sind im Kapitel "11.3 Rückgabe-Werte" genau beschrieben.

Barcode-Ergebnis-Eigenschaften im Detail:

Name der Eigenschaft	Entspricht
ResultType	iBC_Type

ResultType2	iBC_Type2
ResultOrientation	iBC_Orientation
ResultBarcode	szBC_Barcode (bei 2d Barcodes hBC_TwoDimRes)
ResultStatus	iBC_Status
ResultStartX	iBC_StartX
ResultStartY	iBC_StartY
ResultSizeX	iBC_SizeX
ResultSizeY	iBC_SizeY
Result2dCols	iBC_TwoDimCols (nur bei 2d Codes gefüllt)
Result2dRows	iBC_TwoDimRows (nur bei 2d Codes gefüllt)
ResultPdfECL	iBC_PdfECL (nur bei PDF417 gefüllt)
ResultLength	Länge von ResultBarcode (bei 2d Barcodes iBC_TwoDimLen)

Besonderes

Wie auch die Methode `fillResultBinary()` dienen die zwei nur im OCX vorhandenen Eigenschaften `ResultBarcode_Base64` und `ResultLength_Base64` dazu, eine alternative Darstellung der Barcode Ergebnisdaten zu ermöglichen.
Die eine Eigenschaft enthält den Inhalt des gelesenen Barcodes codiert im Base64 Format (siehe z.B. Wikipedia dazu), die andere Eigenschaft gibt die dazu passende Länge an.

10.6 Kurze Beispielanwendung

```
Public Sub ReadBCBtn_Click()
' short demonstration for using QSBCOCX
Dim rcBC As BC_ERROR_TYPES
Dim BCResultNr As Long
  BCResultNr = 0
  'some settings for Barcode
  QSBCOCX1.PictureName = "c:\Qsbcocx\quelle\BarTest.tif"
  QSBCOCX1.BarcodeType = BC_ALL
  QSBCOCX1.BarcodeType2 = BC_NONE
  QSBCOCX1.BarcodeChecksum = BC_CHECKNONE
  QSBCOCX1.BarcodeOrientation = BC_ALL_ORI
  QSBCOCX1.BarcodeLength = 0
  QSBCOCX1.BarcodeLength_from = 0
  QSBCOCX1.BarcodeLength_to = 0
  QSBCOCX1.iNumResults = 10
  'get the first result
  rcBC = QSBCOCX1.getBarcodeResult2
  While rcBC = BC_OK
    BCResultNr = BCResultNr + 1
    With frmTest.ListErgebnisse
      .AddItem BCResultNr & ". Result: "
      .AddItem "Type = " & QSBCOCX1.getTypeName(QSBCOCX1.ResultType)
      .AddItem "Type2 = " & _
               QSBCOCX1.getTypeName2(QSBCOCX1.ResultType2)
      .AddItem "Ergebnis = " & QSBCOCX1.ResultBarcode
      .AddItem "Orientation = " & QSBCOCX1.ResultOrientation
      .AddItem "Status = " & QSBCOCX1.ResultStatus
      .AddItem "StartX = " & QSBCOCX1.ResultStartX
```

```
      .AddItem "StartY = " & QSBCOCX1.ResultStartY
      .AddItem "SizeX = " & QSBCOCX1.ResultSizeX
      .AddItem "SizeY = " & QSBCOCX1.ResultSizeY
      .AddItem ""
    End With
    'get the following results
    rcBC = QSBCOCX1.getBarcodeResult2
  Wend
  MsgBox "Results stopped with " & QSBCOCX1.getErrorName(rcBC)
  'free the memory
  rcBC = QSBCOCX1.freeBarcodeResult2
  MsgBox "Free stopped with " & QSBCOCX1.getErrorName(rcBC)
End Sub
```

Tipp!	**Beispielprogramme** Komplette Beispielprogramme für viele Programmiersprachen sind dem SDK beigefügt. Beachten Sie die Dokumentation qsbarsdk_samples.pdf.

10.7 Auslieferungsdateien

Mit der Runtime-Lizenz müssen Sie die folgenden Dateien mitliefern:

- **QSBCOCX.OCX**
- **QSBARDLL_F.DLL**
- **GraphLib.DLL**
- **Graph_ger.DLL**
- **QSBImage.DLL**
- **FreeImage.dll**
- **qsbLog.dll**

Wenn Sie auch Adobe PDF Dokumente verarbeiten wollen:

- **pstilldll.dll**

Eine Installation der "Microsoft Visual C++ 2013 Redistributable" kann eventuell notwendig sein. Dieses Package erhalten Sie in der aktuellsten Version auf der Microsoft Internetseite.

☞ Die DLLs müssen entweder im Pfad der Applikation oder in einem anderen Pfad sein, in dem das System DLLs findet. Am sichersten ist das Windows System 32 Verzeichnis.

☞ Außerdem müssen Sie die **Lizenzdatei** (`QSBC.lic`) im Verzeichnis der **`QSBCOCX.OCX`** platzieren.

☞ Die Komponente `QSBCOCX.OCX` müssen Sie **registrieren**, z.B. mit dem Befehl `regsvr32.exe qsbcocx.ocx`. Besser: Nutzen Sie das mitgelieferte Setup im Verzeichnis `F_ocx\x86\setup`, dass sie auch mit ausliefern dürfen.

☞ Für jeden Arbeitsplatz, auf dem eine Instanz Ihrer Anwendung läuft, benötigen Sie eine **QS-Barcode Runtime-Lizenz**.

11 Parameter

Hier finden Sie eine Übersicht über die verfügbaren Parameter und Rückgabewerte. Je nach Version sind evtl. nicht alle Parameter verfügbar oder befinden sich in unterschiedlichen Parameterstrukturen, siehe hierzu jeweils das entsprechende Kapitel zu der verwendeten Version. Die Parameter sind gegliedert nach Barcode-Parametern (Kapitel "11.1 Barcode-Parameter"), Advanced Mode (Kapitel "11.2 Erweiterterte ") und Rückgabewerten (Kapitel "11.3 Rückgabe-Werte"), innerhalb der Kapitel sind die Parameter **alphabetisch sortiert**.

11.1 Barcode-Parameter

11.1.1 BC_RotInfo

Ist eine Struktur mit Rotationsinformationen.
(p_lib: BarcodeParam2_tag)
Diese Struktur ist nur aus Kompatibilitätsgründen zur Software **QS-Beleg** vorhanden. Im QS-Barcode Kontext sollte die komplette Struktur mit 0 initialisiert werden, außer `dBC_cos` , das unbedingt auf 1.0 gesetzt sein muss!

11.1.2 hBC_ErrorFile

Achtung! Dieser Parameter ist seit Version 4.6 nicht mehr in Benutzung.
Initialisieren Sie den Wert mit NULL.
(p_lib: BarcodeParam2_tag)

11.1.3 iBC_Checksum

Hier wird festgelegt, ob und mit welcher Prüfsumme gearbeitet wird.
Erlaubt sind: `BC_NONE`, `BC_MOD10`, `BC_MOD10_EXT, BC_MOD43, BC_EXISTENCE, BC_REPORT_CHECKSUM, BC_CODE11_C, BC_CODE11_C_K`.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeChecksum)

Die mit diesem Parameter aktivierten Prüfsummen sind optionale Prüfsummen. Einige Barcodes (z.B. Code 128) verfügen bereits per Definition immer über eine Prüfsumme. Solche Prüfsummen müssen Sie hier nicht spezifizieren. Bei eingestellter Prüfsumme `BC_MOD10, BC_MOD10EXT, BC_CODE11_C` oder `BC_MOD43` wird das letzte Datenzeichen des Barcodes (bei `BC_CODE11_C_K` die letzten zwei) als Prüfziffer interpretiert.
Barcodes mit fehlerhafter Prüfziffer werden komplett überlesen, der Funktions - Returnwert ist also `BC_NO_BARCODE`. Barcodes mit korrekter Prüfziffer werden berichtet, wobei die Prüfziffer im Ergebnis standardmäßig nicht auftaucht.
Dies können Sie ändern, indem Sie iBC_Checksum logisch verknüpfen mit der Konstanten `BC_REPORT_CHECKSUM`, also z.B. C-Syntax.
iBC_Cecksum = `BC_MOD10 | BC_REPORT_CHECKSUM`;
Achtung: Wenn Sie vor der Erkennung iBC_Length exakt angeben und eine Prüfsumme iBC_Checksum definieren, geben Sie die Länge inkl. der Prüfziffer an. Das Ergebnis ist dann um einen kürzer, wenn `BC_REPORT_CHECKSUM` nicht genutzt wird.

Ein Sonderfall ist der Parameter `BC_EXISTENCE`, der mit den anderen Parametern geOdert werden kann. Hierdurch wird die Prüfung auf "Barcode-Verdacht" aktiviert. Zur Steuerung wird bei linearen Barcodes der Parameter iBC_Percent verwendet (Details s. dort).

Bei 2D Barcodes werden bei aktivierter BC_EXISTENCE auch Barcodes berichtet, bei denen die Error Correction die Fehler nicht korrigieren konnte und so keine Dateninhalte geliefert werden können.

Die Prüfziffern werden üblicherweise mit einzelnen Barcodetypen in Verbindung gebracht.

BC_MOD10	2 aus 5 Barcodetypen
BC_MOD43	Code 39
BC_CODE11_C, BC_CODE11_C_K	Code 11

(BC_MOD10_EXT entspricht BC_MOD10, allerdings ohne dass am Ende die Differenz zu 10 gebildet wird, sondern die Prüfziffer ist direkt das Modulo-Ergebnis)

Eine Oder-Verknüpfung von zwei Prüfziffern ist nicht möglich, wenn unterschiedliche Barcode-Typen jeweils mit einer anderen Prüfziffer erkannt werden sollen, ist es notwendig, zwei Erkennungen nacheinander durchzuführen.

Bei der Data Matrix Erkennung hat der Parameter iBC_Checksum eine andere Bedeutung: Hier kann gesteuert werden, ob Data Matrix Barcodes unterschiedlicher ECC Arten gesucht werden sollen. Mögliche Parameter sind:
DM_ECC_000, DM_ECC_050, DM_ECC_080 DM_ECC_100, DM_ECC_140, DM_ECC_200, DM_ECC_ALL
Diese Werte können wie üblich oder-verknüpft werden. Ein Wert von Null für iBC_Checksum bei Data Matrix Suche entspricht DM_ECC_ALL.
Die weiter oben angesprochene Option BC_EXISTENCE steht auch bei DataMatrix zur Verfügung.

11.1.4 iBC_Debug

Achtung! Dieser Parameter ist seit Version 4.6 nicht mehr in Benutzung.
Initialisieren Sie den Wert mit 0.
(p_lib: BarcodeParam2_tag)

11.1.5 iBC_Height

Höhe des Bildes in Pixel.
(p_lib: BarcodeParam2_tag)

11.1.6 iBC_Length

Hier wird die Barcodelänge der Ergebnisse in Zeichen festgelegt. Es kann auch 0 übergeben werden. Dann wird das Intervall von iLaengeVon bis iLaengeBis benutzt. Sind diese Parameter auch auf 0 gesetzt, wird die Länge automatisch ermittelt, wobei als Standardwerte 4 und 64 verwendet werden.
Der maximale Wert ist 64. Sollte der Barcode weniger als 4 Zeichen enthalten, muss iLaengeVon explizit gesetzt werden.

Achtung: Wenn Sie vor der Erkennung iBC_Length exakt angeben und eine Prüfsumme iBC_Checksum definieren, geben Sie die Länge inkl. der Prüfziffer an. Das Ergebnis ist dann um einen kürzer, wenn `BC_REPORT_CHECKSUM` nicht genutzt wird.

Für alle 2D Barcodes sowie für PostNet Codes gilt: iBC_Length, iLaengeVon und iLaengeBis spielen keine Rolle, die Länge der Barcode Dateninhalte ergibt sich aus dem Barcode heraus.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeLength)

11.1.7 iBC_LightMargin

Dies entspricht der Ruhezone um den Barcode.
Defaultwert ist 30 Pixel oder `BC_LIGHTMARGIN`.
Die Ruhezone sollte nicht zu klein gewählt werden, damit die "Lücken" im Barcode nicht fälschlicherweise als Ruhezone interpretiert werden.
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: LightMargin)

11.1.8 iBC_MaxHeight

Hier wird die maximale Höhe des Barcodes spezifiziert, der gesucht werden soll. Default Wert für lineare Barcodes ist 400 Pixel oder `BC_MAXHEIGHT` für die maximale Höhe. Bei aktivierter Existenzsuche (s. iBC_Checksum) wird dieser Parameter ebenfalls genutzt, um zu hohe/lange Linien beim Bericht über Barcode-Existenzen auszuklammern.
Auch bei den 2D Barcodes DataMatrix und PDF417 ist dieser Parameter relevant, die Default-Werte weichen hier von dem Default für lineare Barcodes ab.
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: MaxHeight)

11.1.9 iBC_MaxNoVal

Erlaubte Größe der Lücke innerhalb eines Barcodes. Wird eine größere Anzahl Zeilen lang kein Wert erkannt, wird der Barcode als abgeschlossen angesehen.
Defaultwert: 10 Pixel
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: MaxNoVal)

11.1.10 iBC_MinHeight

Hier wird die minimale Höhe des Barcodes spezifiziert, der gesucht werden soll. Default Wert für lineare Barcodes ist 15 Pixel oder `BC_MINHEIGHT` für die minimale Höhe. Bei aktivierter Existenzsuche (s. iBC_Checksum) wird dieser Parameter ebenfalls genutzt, um zu kurze Linien beim Bericht über Barcode-Existenzen auszuklammern.
Auch bei den 2D Barcodes DataMatrix und PDF417 ist dieser Parameter relevant, die Default-Werte weichen hier von dem Default für lineare Barcodes ab.
Für Patchcodes gelten ebenfalls besondere Einstellungen, siehe Kapitel "14 Besondere Einstellungen".

Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte. Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: MinHeight)

11.1.11 iBC_Orientation

Die Ausrichtung des Barcodes wird hier festgelegt. Die Orientation wird über Konstanten angegeben. Erlaubt sind: `BC_0`, `BC_90`, `BC_180`, `BC_270`. Die Orientierungen dürfen auch mit ODER verknüpft werden.
Weicht ein Barcode von den vier Richtungen zu stark ab, kann er nicht erkannt werden. Dann kann man durch die Konstanten `BC_SKEW_LIGHT` (13°), `BC_SKEW_MEDIUM` (26°) und `BC_SKEW_HEAVY` (39°) zusätzliche Scandurchgänge eingeschaltet werden, die mit ODER verknüpft werden können.
Hat man sehr niedrige Barcodes, kann man zusätzlich noch `BC_SKEW_DENSE_SEARCH` einschalten. Damit werden jeweils zwischen den obigen Winkeln noch Zwischenwinkel zusätzlich untersucht.

Hinweis: `BC_SKEW_HEAVY` bewirkt nicht, dass die beiden anderen Winkel auch gescannt werden. Dazu müssen alle drei Konstanten mit ODER verknüpft werden!
Achtung: Je mehr Winkel untersucht werden, desto länger dauert natürlich die Suche. Geben Sie nur die Winkel an, bei denen wirklich Barcodes auftreten. Probieren Sie erst die vier Grundrichtungen und schalten Sie die zusätzlichen Winkel nur ein, wenn sie wirklich benötigt werden.

Alles oben Genannte gilt ausschließlich bei linearen Barcodes. Bei 2D Barcodes hat der Parameter keine Auswirkung. Dazu gibt es zwei Ausnahmen:

- Bei PDF417 Barcodes hat das Setzen der 4 Winkel `BC_0`, `BC_90`, `BC_180`, `BC_270` sehr wohl eine Auswirkung, die weiteren Parameter zur Verdrehung aber nicht.
- Bei DataMatrix werden immer alle Barcodes beliebiger Lage erkannt, dies kann nicht beeinflusst werden. Es kann aber über `iBC_Orientation=DM_INTENSIVE_SEARCH` die "intensive Suche" aktiviert werden. `iBC_Orientation=0` deaktiviert die "intensive Suche". Mehr dazu im Kapitel "Besondere Einstellungen beim DataMatrix"

(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeOrientation)

11.1.12 iBC_Percent

Für sehr schlechte Barcodes kann durch iBC_Checksum mit dem Wert `BC_EXISTENCE` eine "Verdachtsmeldung" aktiviert werden. Die Steuerung der Empfindlichkeit erfolgt bei linearen Barcodes über `iBC_Percent`. Hier kann angegeben werden, wie viel Prozent der Strichanzahl eines Barcodes erkannt werden muss, um anzuzeigen, dass es vermutlich einen gibt. Die Existenzsuche meldet einen Verdacht auf Barcodes, auch wenn diese nicht gelesen werden können, weil z.B. zu dunkel gescannt wurde und Lücken "zugelaufen" sind.
An einem Beispiel wird das Verfahren erläutert. Wenn der verwendete Barcodetyp z.B. Code 39 ist und die Länge 8 Zeichen ist, dann hat der erzeugte Barcode 50 Striche. Werden nur 44 gefunden, so entspricht dies 88 %. Bei einem kleineren Wert von `iBC_Percent` würde `BC_Exists` gemeldet werden. (Wenn der Barcode gelesen werden kann, wird natürlich `BC_OK` gemeldet). In `iBC_Status` (bzw. im Feld `Status` bei der OCX) der Resultstruktur erhalten Sie dann Auskunft, ob einer gefunden wurde oder nicht. `BC_EXISTS`, wenn einer gefunden wurde und `BC_NOT_EXIST` wenn

nicht. Eine Existenzprüfung ist selten sinnvoll für unbekannten Typ oder unbekannte Länge. Hier immer 100 einstellen.
Hinweis: Mit den Parametern `iBC_MinHeight` und `iBC_MaxHeight` wird der Längenbereich für die zu berücksichtigenden Striche eingestellt.
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: Percent)

11.1.13 iBC_ReadMultiple

Die folgenden Parameter decken die unterschiedlichsten Anwendungsfälle zur Erkennung mehrerer Barcodes auf einem Image ab. In der Regel wird mit `BC_MULTI` gearbeitet. Es wird gesteuert, ob nach einem oder mehreren Barcodes gesucht werden soll. Die möglichen Werte sind als defines im Abschnitt MultiRead aufgeführt:
`BC_ONE`: Es wird nur ein Wert zurückgeliefert. Der Wert muss über das ausgewertete Feld eindeutig sein. Wenn mehr als ein Barcode erkannt wird, wird **gar keiner** berichtet.
`BC_ONE_BREAK`: Es wird nur ein Wert zurückgeliefert. Nachdem ein Barcode gefunden wurde, wird die weitere Suche unterbrochen. Es erfolgt keine Prüfung, ob der gemeldete Barcode der einzige vorhandene ist. Es wird auch nicht überprüft, ob der Barcode ausreichend hoch ist (iBC_MinHeight). Diese Option ist sehr schnell, sollte aber nur genutzt werden, wenn Länge und Typ des Barcodes bekannt sind, die Länge > 6 beträgt und am besten nur in Kombination mit Prüfsummen.
`BC_MULTI`: Es wird nach mehreren Barcodes gesucht. Es wird jeder erkannte Wert einmal zurückgeliefert.
`BC_MULTI_ONE`: Es wird nach mehreren Barcodes gesucht. Es wird nur ein Wert pro Barcode zurückgeliefert. Der Wert muss über dem Barcode eindeutig sein.
`BC_MULTI_BESTGUESS`: Es wird nach mehreren Barcodes gesucht. Es wird nur ein Wert pro Barcode zurückgeliefert. Es werden diejenigen Werte berichtet, die über dem Barcode eindeutig sind oder die die meisten Vorkommen auf dem Barcode haben.
`BC_MULTI_MULTI`: Es wird nach mehreren Barcodes gesucht. Es werden alle erkannten Werte berichtet, auch mehrfach die gleichen Barcodes, wenn z.B. mehrere Etiketten gleichen Inhalts vorhanden sind und die Anzahl ermittelt werden soll.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: ReadMultiple)

Achtung: Wir empfehlen dringend, immer `BC_MULTI` oder `BC_MULTI_MULTI` zu nutzen. Auch dann, wenn nur ein Barcode gelesen werden soll. Dies hat sich als optimaler Schalter in (fast) allen Situationen herausgestellt.

Bei 2D Barcodes spielt dieser Parameter keine Rolle. Es werden immer alle 2D Barcodes des gewählten Typs gesucht und erkannt.

11.1.14 iBC_ResultCount

Gibt an, wie viele Ergebnis-Strukturen in `brBC_Result` stehen.
(p_lib: BarcodeParam2_tag)

11.1.15 iBC_ScanDistance

Dies entspricht dem Suchabstand, d.h., die Abtastung des Bildes in y-Richtung erfolgt mit Schritten in Höhe des Suchabstandes. Defaultwert ist 15 Pixel oder `BC_SCANDISTANCE`.
Bei schlecht gedruckten Barcodes sollte dieser Wert verringert werden. Dies hat allerdings eine Verlangsamung zur Folge.
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
Hinweis: An der Oberfläche in den Dialogen, z.B. bei der Barcode Demo, wird dieser Parameter oft auch als "Suchabstand" bezeichnet.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: ScanDistance)

11.1.16 iBC_ScanDistBarcode

Dies entspricht der Abtastung des Barcodes in y-Richtung. Defaultwert ist 3 Pixel oder `BC_SCANDISTBAR`.
Bei schlecht gedruckten Barcodes sollte dieser Wert verringert werden. Dies hat allerdings eine Verlangsamung zur Folge.
Hinweis: An der Oberfläche in den Dialogen, z.B. bei der Barcode Demo, wird dieser Parameter manchmal nur als "Scanabstand" bezeichnet.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: ScanDistBarcode)

11.1.17 iBC_SizeX

Breite des Suchfeldes für den Barcode. Kann auch 0 sein. Wenn iBC_SizeX, iBC_SizeY, iBC_StartX und iBC_StartY alle 0 sind, wird das ganze Bild genommen.
(p_lib: BarcodeParam2_tag, f_dll: Barcode2_tag, f_ocx: SizeX)

11.1.18 iBC_SizeY

Höhe des Suchfeldes für den Barcode. Kann auch 0 sein. Wenn iBC_SizeX, iBC_SizeY, iBC_StartX und iBC_StartY alle 0 sind, wird das ganze Bild genommen.
(p_lib: BarcodeParam2_tag, f_dll: Barcode2_tag, f_ocx: SizeY)

11.1.19 iBC_StartX

x-Koordinate des Suchfeldes für den Barcode. Kann auch 0 sein. Wenn iBC_SizeX, iBC_SizeY, iBC_StartX und iBC_StartY alle 0 sind, wird das ganze Bild genommen.
(p_lib: BarcodeParam2_tag, f_dll: Barcode2_tag, f_ocx: StartX)

11.1.20 iBC_StartY

y-Koordinate des Suchfeldes für den Barcode. Kann auch 0 sein. Wenn iBC_SizeX, iBC_SizeY, iBC_StartX und iBC_StartY alle 0 sind, wird das ganze Bild genommen.
(p_lib: BarcodeParam2_tag, f_dll: Barcode2_tag, f_ocx: StartY)

11.1.21 iBC_Tolerance

Maximale Schieflage des Barcodes (Linientoleranz), Defaultwert : 10
Bei PostNet Codes hat dieser Parameter eine abweichende Bedeutung und andere Standardwerte.Alles weitere dazu in Kapitel "12.6 Besondere Einstellungen bei PostNet Codes".
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: Tolerance)

11.1.22 iBC_Type

Hier wird der Barcodetyp festgelegt, nach dem gesucht werden soll. Die Konstanten umfassen: `BC_CODABAR, BC_CODE128, BC_CODE39, BC_CODE39EXT, BC_CODE93, BC_EAN128, BC_DATAMATRIX, BC_EAN8, BC_EAN13, BC_INDUSTRIE25, BC_INTERLEAVED25, BC_PDF417, BC_UPCA, BC_UPCE, BC_CODABLOCK, BC_25_IATA, BC_25_3MATRIX, BC_25_3DATALOGIC, BC_25_BCDMATRIX, BC_25_INVERTIERT, BC_CODE32, BC_PHARMA, BC_CODE93EXT, BC_PATCHCODE, BC_QR_CODE, BC_AZTEC.`

Die Typen dürfen auch mit ODER verknüpft werden, um mehrere Typen zugleich zu suchen.

Ausnahmen:

- Patchcode
- Pharmacode
- Codablock F
- 2D Barcodes DataMatrix, PDF417, QR Code, Aztec Code
  (iBC_Type wählt den 2D Barcode aus, iBC_Type2 muss `BC_NONE` sein)
- PostNet Codes
  (iBC_Type muss `BC_NONE` sein, iBC_Type2 wählt den PostNet Code aus)

Diese Typen müssen immer allein gesetzt werden, Kombinationen sind nicht möglich. Sollen mehrere dieser Typen erkannt werden, können bei Bedarf mehrere Aufrufe der Library nacheinander programmiert werden.

(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeType)

11.1.23 iBC_Type2

Dieser Parameter ist seit Version QS-Barcode 4.6 neu hinzugekommen!

Hier werden weitere Barcode-Typen festgelegt, nach denen gesucht werden soll.

Die Konstanten umfassen: `BC_GS1_EAN13, BC_GS1_DATABAR, BC_GS1_DATABAR_STACKED, BC_GS1_DATABAR_EXP, BC_GS1_DATABAR_EXP_STACKED, BC_GS1_DATABAR_LIMITED, BC_POSTNET_IMB, BC_POSTNET_ROYAL, BC_POSTNET_KIX, BC_POSTNET_AUSTRALIAN, BC_POSTNET_CPC, BC_POSTNET_ZIP, BC_POSTNET_PLANET.`

Die Typen der `GS1_DATABAR` dürfen auch mit ODER verknüpft werden, um mehrere Typen zugleich zu suchen.

Die Typen der POSTNET-Familie müssen immer alleine gewählt werden. Weder können mehrere POSTNET Typen gleichzeitig gesetzt werden noch dürfen andere Bacode-Typen mit ODER dazu verknüpft werden.

Bei Wahl eines PostNet Codes muss außerdem iBC_Type = `BC_NONE` gesetzt werden.

Wenn iBC_Type mit einem der folgenden Typen belegt wird, muss der Inhalt von iBC_Type2 `BC_NONE` sein:

- Patchcode
- Pharmacode
- Codablock F
- 2D Barcodes DataMatrix, PDF417, Aztec Code

(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeType2)

Wenn Sie für den QR Code explizit nach ECI-Codes suchen, geben Sie bei iBC_Type2 `BC_QR_CODE_ECI` an, sonst ebenfalls `BC_NONE`. (Siehe auch Kapitel 16.5 QR Code Extended Channel Interpretation (ECI) Mode)

11.1.24 iBC_Width
Breite des Bildes in Pixel.
(p_lib: BarcodeParam2_tag)

11.1.25 iLaengeBis
Größte mögliche Länge des Ergebnisses unbekannter Barcodelänge in Zeichen. Dieser Wert kann auch 0 sein. Sind sowohl iLaengeVon, iLaengeBis, und iBC_Length auf 0 gesetzt, wird die Länge automatisch ermittelt.
Für alle 2D Barcodes und für PostNet Codes gilt: iBC_Length, iLaengeVon und iLaengeBis spielen keine Rolle, die Länge der Barcode Dateninhalte ergibt sich aus dem Barcode heraus.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeLength_to)

11.1.26 iLaengeVon
Kleinste mögliche Länge des Ergebnisses bei unbekannter Barcodelänge in Zeichen.
Dieser Wert kann auch 0 sein. Sind sowohl iLaengeVon, iLaengeBis, und iBC_Length auf 0 gesetzt, wird die Länge automatisch ermittelt.
Für alle 2D Barcodes unf für PostNet Codes gilt: iBC_Length, iLaengeVon und iLaengeBis spielen keine Rolle, die Länge der Barcode Dateninhalte ergibt sich aus dem Barcode heraus.
(p_lib: BarcodeData2_tag, f_dll: Barcode2_tag, f_ocx: BarcodeLength_from)

11.1.27 iBC_MemorySize
Hier wird vom Aufrufer angegeben, wie viel Speicher er in `pBC_Memory` allokiert hat.
(p_lib: BarcodeParam2_tag)

11.1.28 lpstrBC_Image
Dies ist ein Zeiger auf die Bits des Bildes. Das Bild muss monochrom sein (monochrom, weil der Parameter relevant für die Schnittstelle p_lib ist, beim Einsatz anderer Schnittstellen sind auch Graustufen- oder Farbbilder möglich). Die Darstellung muss pro Pixel ein Bit betragen. Die Bits mit der obersten Bildzeile beginnen, d.h. (0,0) ist oben links, schwarze Pixel sind 0-Bits, weiße Pixel sind 1-Bits.
WICHTIG: Die Bildzeilen *müssen* auf 32 Bit normiert sein (32-bit Alignment). Wenn Ihr Bild also 310 Pixel breit ist, muss jede Bildzeile aus 320 Bits = 80 Byte besetehen. Die letzten 10 Bit, die außerhalb des Bildbereichs liegen, sollten mit 1 (=weiß) gefüllt sein. Beim Setzen von `iBC_Width` können Sie entweder die genaue Bildbreite angeben (um beim Beispiel zu bleiben: 310) oder Sie geben die normierte Breite an (320).
(p_lib: BarcodeParam2_tag)

11.1.29 pBC_BarcodeData
Dies ist der Zeiger auf die `BarcodeData`-Strukturen, die für jeden einzelnen Typ, der gesucht werden soll, aufgebaut werden müssen.
(p_lib: BarcodeParam2_tag)

11.1.30 pBC_Memory
In diesen vom Aufrufer allokierten Speicherbereich werden die Ergebnisse geschrieben. Ausgelesen werden die Ergebnisse über den `brBC_Result`-Zeiger. Die Anzahl der Ergebnisse, die im Speicherbereich stehen, wird in `iBC_ResultCount` abgelegt.

(p_lib: BarcodeParam2_tag)

11.1.31 pBC_NextBarcodeData2

Dies ist die Verknüpfung zum nächsten `BarcodeData2_tag`-Block. Für jeden Typ der erkannt werden soll, muss eine eigene Struktur aufgebaut werden.
Prinzipiell ergeben sich zwei Möglichkeiten, wenn nach mehreren Barcodes in einem Feld gesucht werden soll: Entweder, es wird ein `BarcodeData2_tag`-Block gefüllt, in dem die unterschiedlichen Barcodetypen ODER verknüpft werden und die Länge entsprechend den zu erwartenden Ergebnissen flexibel eingestellt wird (z.B. von 4 bis 11), oder aber es wird für jeden Barcode-Typ ein `BarcodeData2_tag`-Block erzeugt, was den Vorteil hat, dass man dann für jeden Typ eine exakte Länge angeben kann, sofern diese bekannt ist. Diese Unterscheidung lässt sich natürlich von der Länge auch auf andere Parameter wie z.B. die Orientierung übertragen.
(p_lib: BarcodeData2_tag)

11.1.32 pbrBC_Result

Ist ein Zeiger auf die Ergebnis-Strukturen.
(p_lib: BarcodeParam2_tag)

11.1.33 szBC_SubstitutionChar

Wie unter "szBC_SubstitutionString" beschrieben ist dies das Substitutionszeichen, welches für nicht erkannte Zeichen des Barcodes eingesetzt wird.
(p_lib: BarcodeParam2_tag)

11.1.34 szBC_SubstitutionString

Ist der String ungleich Null, so wird der entsprechende String als Ergebnis zurückgegeben, wenn der Barcode nicht richtig erkannt wurde (bei QS-Beleg z.B. "\bf" für Barcodefehler). Ist der String Null, so werden die nicht erkannten Zeichen des Barcodes mit dem unter `szBC_SubstitutionChar` angegebenen Zeichen zurückgegeben. Default ist hier '?'. Beispiel: 12??567.
(p_lib: BarcodeParam2_tag)

11.1.35 szBC_Version

Ignorieren Sie diesen Parameter. Er wird nicht mehr verwendet! Die aktuelle Versionsnummer ermitteln Sie stattdessen über die Funktion QSVersion().
(p_lib: BarcodeParam2_tag)

11.2 Erweiterterte Suche

11.2.1 AdvancedSearch

Hier wird die erweiterte Suche ein- und ausgeschaltet. 0=aus, 1=ein.
Die erweiterte Suche bietet die Möglichkeit, einige der folgenden Parameter während einer Barcodesuche zu variieren. So lassen sich während eines Suchlaufes Schmutzpixel verschiedener Breite entfernen sowie bis zu drei verschiedene Ruhezonen benutzen. Falls kein Barcode gefunden wurde, kann automatisch ein weiter Suchlauf gestartet werden, der das Bild mit einem engeren Suchabstand abtastet. Für die erneute Abtastung des Bildes werden **iBC_ScanDistance2** und **iBC_ScanDistBar2** herangezogen.
Defaultwert ist 1 oder `BC_ADVANCED_SEARCH`.

Achtung: Die "Erweiterte Suche" kann nur bei linearen Barcodes verwendet werden. Siehe auch Kapitel 12.
(f_ocx: AdvancedSearch)

11.2.2 DynamicThreshold

Schalten Sie hier die dynmische Schwellwertermittlung ein. Hierbei wird der Schwellwert aus dem im Bild bzw. dem übergebenem Ausschnitt vorkommenden Helligkeitswerten ermittelt. Dies entspricht der „automatischen Helligkeitseinstellung“, wie sie in vielen Kopiergeräten zu finden ist. Es wird für die Ermittlung des "dynamischen Schwellwerts" das ganze Bild verwendet und nicht nur der Barcodebereich. 0=aus, 1=ein.
Defaultwert ist 1 oder `BC_DYNAMIC_THRESHOLD`.
Siehe auch Kapitel 12.2 Dynamischer Schwellwert (DynamicThreshold).
(f_ocx: DynamicThreshold)

11.2.3 iBC_Threshold

Wenn Sie keine dynamische Schwellwertermittlung benötigen, können Sie den gewünschten Schwellwert explizit angeben. Der Wert für den Schwellwert liegt dabei im Wertebereich von 0-255. Unser Standard-Schwellwert liegt bei 160. Je niedriger der Schwellwert ist, desto eher wird ein graues Pixel in ein weißes Pixel umgewandelt und desto heller wird das Bild insgesamt.
Defaultwert ist 160 oder `BC_THRESHOLD`.
(f_ocx: Threshold)

11.2.4 iBC_RemovePixel

Entfernung von weißen und schwarzen "Schmutzpixeln", die häufig durch qualitativ schlechtere Scanner hervorgerufen werden. Der Wert gibt die Pixelbreite an, die innerhalb einer Zeile entfernt wird. Der gültige Wertebereich beträgt 0-2.
Defaultwert ist 2 oder `BC_REMOVE_PIXEL`.
(f_ocx: RemovePixel)

11.2.5 iBC_LightMargin1/iBC_LightMargin2/iBC_LightMargin3

In der erweiterten Suche können insgesamt bis zu drei Ruhezonen angegeben werden, die zur Barcodesuche herangezogen werden. Siehe 11.1.7 iBC_LightMargin
Defaultwert ist 30/45/15 oder
`BC_LIGHTMARGIN1/BC_LIGHTMARGIN2/BC_LIGHTMARGIN3/`
(f_ocx: LightMargin1/ LightMargin2/ LightMargin3)

11.2.6 iBC_ScanDistance2

In der erweiterten Suche besteht die Möglichkeit, einen zweiten Suchlauf durchzuführen, wenn im ersten Durchlauf kein Barcode gefunden wurde. Der zweite Suchlauf benutzt den hier definierten Suchabstand. Wird kein zweiter Suchlauf benötigt, ist dieser Wert auf 0 zu setzen. Siehe auch 11.1.15 iBC_ScanDistance
Defaultwert ist 5 oder `BC_SCANDISTANCE2`.
(f_ocx: ScanDistance2)

11.2.7 iBC_ScanDistBar2

Der zweite Suchlauf benutzt den hier definierten Wert zur Abtastung des Barcodes. Wird kein zweiter Suchlauf gewünscht, ist dieser Wert auf 0 zu setzen. Siehe auch 11.1.16 iBC_ScanDistBarcode
Defaultwert ist 1 oder `BC_SCANDISTBAR2`.
(f_ocx: ScanDistBar2)

11.3 Rückgabe-Werte

11.3.1 dBC_cos

Dies ist der cos-Wert des Rotationswinkels. Wenn keine Rotation erfolgen soll, so sind alle Parameter auf 0 zu setzen. Alleine der cos-Wert ist auf 1.0 zu setzen.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.2 dBC_sin

Dies ist der sin-Wert des Rotationswinkels.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.3 dBC_XKorr

Dies ist der x-Korrekturfaktor. Er ergibt sich aus Ist-Breite durch Soll-Breite des Beleges.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.4 dBC_YKorr

Dies ist der y-Korrekturfaktor. Er ergibt sich aus Ist-Höhe durch Soll-Höhe des Beleges.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.5 fBC_bRotated

Besagt, dass der Beleg schon rotiert wurde. Wenn Sie diesen Parameter auf `TRUE` setzen, sollten Sie auch alle anderen Parameter auf 0 setzen, bis auf cos, dessen Wert 1.0 betragen muss.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.6 hBC_TwoDimRes

Über diesen Handle wird das Ergebnis eines 2D-Barcodes zurückgegeben. Bei einem 2D-Baroce ist `szBC_Barcode` immer NULL.
(p_lib: TwoDimResult_tag, f_dll: TwoDimResult_tag, f_ocx: ResultBarcode)

11.3.7 iBC_BMoffsetX
Dies ist der x-Wert des Rotationspunktes in Pixel.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.8 iBC_BMoffsetY
Dies ist der y-Wert des Rotationspunktes in Pixel.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.9 iBC_TwoDimCols
Anzahl der erkannten Spalten im Barcode.
(p_lib: TwoDimResult_tag, f_dll: TwoDimResult_tag, f_ocx: Result2dCols)

11.3.10 iBC_PdfECL
Fehlerkorrektur-Level. Nur bei PDF417 Barcodes von Bedeutung.
(p_lib: TwoDimResult_tag, f_dll: TwoDimResult_tag, f_ocx: ResultPdfECL)

11.3.11 iBC_offsetX
Dies gibt die x-Verschiebung des Bildes in Pixel an.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.12 iBC_offsetY
Dies gibt die y-Verschiebung des Bildes in Pixel an.
Dieser Parameter dient im Wesentlichen der Kompatibilität zu QS-Beleg.
(p_lib: tag_RotInfo)

11.3.13 iBC_Orientation
Gibt die Orientierung des gefundenen Barcodes zurück.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultOrientation)

11.3.14 iBC_TwoDimLen
Länge des Ergebnisses in Byte.
(p_lib: TwoDimResult_tag, f_dll: TwoDimResult_tag, f_ocx: ResultLength)

11.3.15 iBC_TwoDimRows
Anzahl der erkannten Zeilen im Barcode.
(p_lib: TwoDimResult_tag, f_dll: TwoDimResult_tag, f_ocx: Result2dRows)

11.3.16 iBC_SizeX
Breite des gefundenen Barcodes in Pixel.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultSizeX)

11.3.17 iBC_SizeY
Höhe des gefundenen Barcodes in Pixel.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultSizeY)

11.3.18 iBC_StartX
x-Koordinate des gefundenen Barcodes (obere linke Ecke) in Pixel.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultStartX)

11.3.19 iBC_StartY

y-Koordinate des gefundenen Barcodes (obere linke Ecke) in Pixel.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultStartY)

11.3.20 iBC_Status

Der Status des erkannten Barcodes:

- `BC_OK` für eindeutig erkannte Barcodes
- `BC_GUESS` für den Wert, der im Modus `BC_MULTI_BESTGUESS` am häufigsten vorkommt
- `BC_EXISTS` wenn kein Wert ermittelt werden konnte, oder bei Existenzprüfung.

(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultStatus)

11.3.21 iBC_Type

Hier wird der gefundene Barcodetyp zurückgegeben. Wenn mit mehreren Typen gesucht wurde, so kann der Benutzer hier erkennen, um welchen Barcode es sich handelt.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultType)

11.3.22 iBC_Type2

Hier wird der gefundene Barcodetyp zurückgegeben. Wenn mit mehreren Typen gesucht wurde, so kann der Benutzer hier erkennen, um welchen Barcode es sich handelt.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultType2)

11.3.23 szBC_Barcode

Dies ist das eigentliche Ergebnis der Suche. Hier wird der Barcodewert zurückgegeben.
(p_lib: BarcodeResult2_tag, f_dll: BarcodeResult2_tag, f_ocx: ResultBarcode)

12 Erweiterte Suche

Die erweiterte Suche (ab QS-Barcode SDK Version 4.9.1) bietet die Möglichkeit, verschiedene Parameter während einer Barcodesuche zu variieren. Wird mit den eingestellten Parametern dieser Suche kein Barcode gefunden, kann automatisch ein weiter Durchlauf gestartet werden, der das Bild mit einem engeren Suchabstand abtastet.

Achtung: Die "Erweiterte Suche" kann nur bei linearen Barcodes verwendet werden.

12.1 Parameter der Erweiterten Suche

12.1.1 iBC_RemovePixel

Bei vielen Vorlagen gibt es nach der Bilderzeugung und der Binarisierung (dem Wandeln der Bilder in Schwarzweiß) schwarze und weiße Schmutzpunkte.

Das Eingabebild

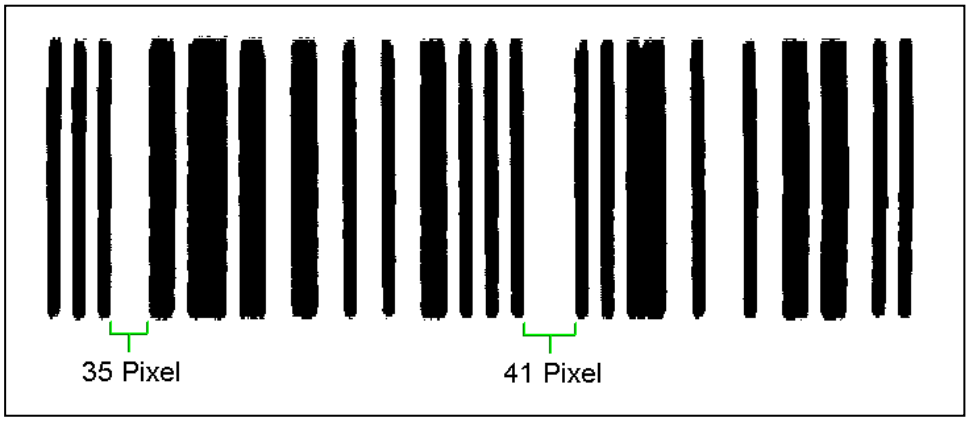
weist viele schwarze Punkte zwischen den Barcodelinien auf. Dadurch ergeben sich bei der zeilenweisen Abtastung zusätzliche Wechsel, die eine Erkennung des Barcodes verhindern.
Hier können sowohl schwarze Punkte zwischen als auch weiße Löcher in den Barcodelinien entfernt werden. Die Angabe der Pixelbreite (1,2) gilt immer jeweils für weiße und schwarze Pixel gleichermaßen.
Soll diese Funktion nicht benutzt werden, so wird der Wert auf 0 gesetzt.

12.1.2 iBC_LightMargin1/iBC_LightMargin2/iBC_LightMargin3

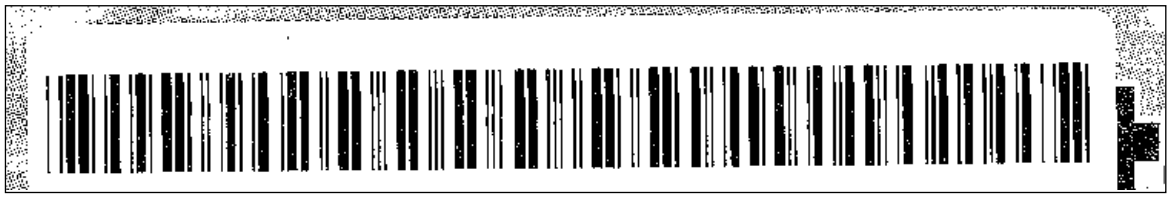
Vor und hinter einem Barcode wird ein weißer Bereich als Ruhezone erwartet. Dieser weiße Bereich muss breiter sein als jede weiße Lücke innerhalb des Barcodes und breiter als jeder schwarze Strich in dem Barcode. Die Abbildung rechts verdeutlicht dies noch einmal.

Die Angabe erfolgt in Pixel (Bildpunkte). Der Standard-Wert von 30 Pixeln bezieht sich auf gängige 200dpi Bilder. Sie sollten mit einem höheren Wert arbeiten, wenn Ihre Scans eine hohe Auflösung haben.

Der folgende Ausschnitt wurde mit 600dpi gescannt. Die hohe Auflösung führt dazu, dass innerhalb des Barcodes weiße Lücken auftreten, die deutlich über dem Standardwert von 30 Pixeln liegen. Erhöht man den Ruhezone-Wert auf 45 Pixel, wird der Barcode problemlos erkannt.



Das nächste Beispiel zeigt ein Barcode-Etikett, auf dem der Barcode zu nah an den Rand gedruckt wurde. Die weiße Ruhezone hinter dem Barcode ist so kurz, dass ggf. das Ende des Barcodes nicht erkannt wird und der Strich auf dem Papier neben dem Etikett mit zum Barcode gerechnet wird. Dadurch wird eine korrekte Erkennung verhindert. In diesem Fall wäre eine Reduzierung der Ruhezone auf 15 Pixel notwendig, damit das Ende des Barcodes richtig erkannt wird.



Während eines Suchlaufes lassen sich Barcodes daher mit bis zu drei unterschiedlichen Ruhezonen gleichzeitig suchen. Damit lassen sich in einem Dokument Barcodes unterschiedlicher Ruhezonen finden. Außerdem ist man flexibler, wenn sich die Ruhezone auf den untersuchten Dokumenten z.B. aufgrund des Aufklebens von Barcodes verändert.
Nicht benötigte Ruhezonen werden mit dem Wert 0 ausgeschaltet.

12.1.3 Automatischer zweiter Suchdurchlauf

Falls im ersten Durchlauf keine Barcodes gefunden wurden, kann automatisch ein zweiter Suchlauf gestartet werden, der das Bild mit einem engeren Suchabstand abtastet. Für die erneute Abtastung des Bildes werden **iBC_ScanDistance2** und **iBC_ScanDistBar2** (siehe Kapitel 11.1.15/11.1.16) herangezogen. Die vorhandenen Werte für das Entfernen von Schmutzpixeln und der Ruhezonen werden auch in diesem Suchlauf benutzt.
Soll kein zweiter Suchlauf durchgeführt werden, so müssen beide Werte auf 0 gesetzt werden.

12.2 Dynamischer Schwellwert (DynamicThreshold)

Bei der Umwandlung (Binarisierung) von Farb- oder Graustufenbildern wird ein Schwellwert (engl. threshold) verwendet. Punkte, die heller sind als dieser Schwellwert, werden weiß, die anderen schwarz.
Mögliche Werte für den Schwellwert liegen zwischen 0 (schwarz) und 255 (weiß).

Sind die Bilder zu hell oder zu dunkel gescannt, liegen aber als Farb- oder Graustufenbilder vor, lässt sich dies möglicherweise noch bei der Umwandlung (Binarisierung) in ein Schwarzweißbild (monochrom) korrigieren.

Bei der internen Umwandlung wird normalerweise der Schwellwert 160 verwendet. Dieser Wert hat sich für sehr viele Fälle als gut erwiesen. Ist das Bild aber ggf. zu hell oder zu dunkel, kann ein fester Schwellwert zu Problemen führen. Bei zu dunklen Bildern können die schwarzen Balken des Barcodes z.B. verschmelzen, so dass eine Erkennung nicht mehr möglich ist.

Hier gibt es jetzt die Möglichkeit, den Schwellwert über das ganze Bild bzw. den Bildausschnitt automatisch ermitteln zu lassen. Dabei wird der Schwellwert aus den im Bild vorkommenden Helligkeitswerten ermittelt. Dies entspricht der „automatischen Helligkeitseinstellung“, wie sie in vielen Kopiergeräten zu finden ist. Es wird für die Ermittlung des „dynamischen Schwellwerts“ das ganze Bild verwendet, nicht nur der Barcodebereich

Dieses Bild

wird mit Schwellwert = 160 so binarisiert:

Mit dynamischem Schwellwert aber so:

Gültige Werte für DynamicThreshold sind 0 = aus, 1 = ein.
Wenn Sie den Schwellwert doch lieber selber bestimmen möchten, können Sie den DynamicThreshold auch ausschalten und über iBC_Threshold einen individuellen Schwellwert angeben.

13 Bildbearbeitung

Über die Schnittstellen QSBarcode6() der p_lib bzw. QSReadBarcode6() der f_dll und h_dll lassen sich jetzt Befehle zur Bildbearbeitung direkt übergeben, die in den älteren Funktionen aus einer "qsbc.ini" ausgelesen wurden.

Die Befehle werden jeweils vor der Barcodeerkennung auf einer Kopie des zu verarbeitenden Bildes durchgeführt, die Eingabedatei wird dabei nicht verändert.

Für die Befehle zur Bildverarbeitung stehen intern zwei Libraries zur Verfügung. Es sind dies die Freeimage-Library sowie die Graphlib-Library.

Für die p_lib dürfen nur Befehle aus der Graphlib-Library benutzt werden. Außerdem haben einige Befehle keine Wirkung, da hier immer auf Bildern mit einer Farbtiefe von 1 Bit gearbeitet wird.

Für die h_dll und die f_dll können die Befehle aus beiden Libraries genutzt und auch gemischt angegeben werden.

Ein oder mehrere Befehle zur Bildbearbeitung werden der jeweiligen Barcodefunktion über die Variable "szImaging" übergeben. Jeder einzelne Befehl muss dabei zwingend mit einem Semikolon abgeschlossen werden, z.B.

"FreeImage:GrayScale,8;GraphLib:SubImage("c:\test1.bmp");"

Die Befehle der Library FreeImage:

In dieser Library stehen aktuell nur zwei Befehle zur Verfügung.

Befehl	Beschreibung
FreeImage:GrayScale,8;	Umwandlung eines Bildes in ein Graustufenbild (8 Bit Farbtiefe). Die Umwandlung kann von jeder beliebigen Farbtiefe aus vorgenommen werden.
FreeImage:Threshold,160;	Umwandlung des Bildes in ein s/w-Bild. Der bei der Umwandlung zu verwendende Schwellwert kann direkt angegeben. Mögliche Werte für die Schwelle liegen zwischen 0 (schwarz) und 255 (weiß).

Die Befehle der Library GraphLib:

Befehl	Beschreibung
GraphLib:Binarize(160);	Umwandlung des Bildes in ein s/w-Bild mit der Übergabe eines Schwellwertes (160). Mögliche Werte für die Schwelle liegen zwischen 0 (schwarz) und 255 (weiß).
GraphLib:GrayScale(0);	Umwandlung eines Bildes in ein Graustufenbild (8 Bit Farbtiefe). Die Umwandlung kann nur von 24 Bit Farbtiefe aus erfolgen. Der übergebene Wert spielt hierbei keine Rolle, er muss lediglich numerisch sein
GraphLib:Dynamic;	Der Schwellwert wird aus den im gesamten Bild vorkommenden Helligkeitswerten ermittelt. Dieser Befehl funktioniert nur auf Graustufenbildern.

Befehl	Beschreibung
GraphLib:DynamicEx("0");	Der Schwellwert wird wie bei "Dynamic" bestimmt, allerdings noch um einen Versatz verschoben. Dabei bewirkt ein positiver Versatz eine Verdunklung, ein negativer eine Aufhellung des Bildes. Dieser Befehl funktioniert nur auf Graustufenbildern. Wertebereich: -255 bis +255

Befehl	Beschreibung
GraphLib:Contrast(2.0);	Einstellung des Kontrastes (immer als Dezimalzahl). Beim Grenzwert von 1.0 wird der Kontrast nicht verändert. Werte zwischen 0.0 bis 1.0 erhöhen den Kontrast, alle Werte über 1 verringern den Kontrast. Dieser Befehl funktioniert nur auf Graustufenbildern.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:Brighten(50);	Helligkeit des Bildes in %. Der Wertebereich liegt zwischen -100 und 100. Werte zwischen 0-100 machen das Bild heller, Werte zwischen -100 - 0 machen das Bild dunkler. Dieser Befehl funktioniert nur auf Graustufenbildern.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:DespeckEx(5,5);	Entfernung schwarzer und weißer Schmutzpunkte. Die beiden Werte geben den maximalen Durchmesser der zu entfernenden schwarzen und weißen Pixel an. Es werden nur „alleinstehende“ Punkte in der angegebenen Größe entfernt. Dieser Befehl funktioniert nur auf Bildern mit 1 Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:Dilation;	Allgemeine Schwarzverstärkung. Dieser Befehl funktioniert nur auf Bildern mit 1 Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:Linearcontrast;	Linearer Kontrastausgleich. Dieser Befehl funktioniert nur auf Graustufenbildern.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:Subimage("c:\test.bmp");	Speichern des aktuellen Zustands des Bildes, Das Dateiformat wird durch die Dateinamenendung ".tif" oder ".bmp".bestimmt.
GraphLib:SubimageEx("c:\test.bmp",0,0,500,500);	Speichern eines Ausschnittes eines Bildes. Übergeben werden Zieldatei sowie StartX, StartY, Breite und Höhe des Ausschnitts.
GraphLib:Invert;	Invertieren des Bildes. Dieser Befehl funktioniert nur auf Bildern mit 1 Bit Farbtiefe.

Befehl	Beschreibung
GraphLib:LocalThreshold(0,20,"0.9");	Drei Parameter steuern diesen Befehl. Der erste Parameter ist 0 und kann nicht verändert werden. Der zweite Parameter steuert die Größe der Region, in der der lokale Schwellwert ermittelt wird. 20 hat sich hier als guter Wert erwiesen. Ein kleinerer Wert erhöht zwar die Geschwindigkeit aber ebenso die Gefahr, dass in einheitlich weißen oder schwarzen Regionen durch kleinere Helligkeitsvariationen Löcher entstehen. Die „0.9“ in der letzten Stelle wiederum besagt, dass alle Bildpunkte, deren Helligkeit über 90% der mittleren Helligkeit der Umgebung liegt, als weiß betrachtet werden. Der Befehl arbeitet nur auf Graustufenbildern (8 Bit) Diese Funktion ist sehr zeitaufwendig.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:VMedian(2);	Verwischen/Unschärfe schwarzer Bereiche Die Nachbar-Farbwerte einer übergebenen Pixelgröße werden sortiert und der aktuelle Pixel wird durch den Mittelwert ersetzt. Je größer der übergebene Wert, desto größer wird der veränderte Bereich. Dieser Befehl funktioniert auf Bildern mit 1Bit und 8Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:VDilation(1);	Verstärkung schwarzer Bereiche Die Nachbar-Farbwerte einer übergebenen Pixelgröße werden sortiert und der aktuelle Pixel wird durch den größten Wert ersetzt. Je größer der übergebene Wert, desto größer wird der veränderte Bereich. Dieser Befehl funktioniert auf Bildern mit 1Bit und 8Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:VErosion(1);	Ausdünnung (überbetonter) schwarzer Bereiche. Die Nachbar-Farbwerte einer übergebenen Pixelgröße werden sortiert und der aktuelle Pixel wird durch den kleinsten Wert ersetzt. Je größer der übergebene Wert, desto größer wird der veränderte Bereich. Dieser Befehl funktioniert auf Bildern mit 1Bit und 8Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:VOpening(3);	Öffnet dünne Verbindungen zwischen schwarzen Bereichen. Je größer der übergebene Wert, desto stärker werden Verbindungen geöffnet. Dieser Befehl funktioniert auf Bildern mit 1Bit und 8Bit Farbtiefe.

Vorher:

Nachher

Befehl	Beschreibung
GraphLib:VClosing(3);	Schließt dünne weiße Lücken in schwarzen Bereichen Je größer der übergebene Wert, desto stärker werden Lücken geschlossen. Dieser Befehl funktioniert auf Bildern mit 1Bit und 8Bit Farbtiefe.

Vorher:

Nachher

Weitere Informationen entnehmen Sie bitte dem Dokument "bcTipps.pdf".

14 Besondere Einstellungen

Bei einigen Barcodetypen gilt es Besonderheiten zu beachten, die sich aus den Eigenarten des jeweiligen Codes ergeben. Insbesondere gibt es für die 2D Barcodes einige Sonderregelungen.

14.1 Besondere Einstellungen beim Patchcode

Der Patchcode zeichnet sich durch seine sehr langen, aber sehr wenigen (immer 4) Striche aus. Wir raten daher dazu, den Parameter **iBC_MinHeight** von 15 auf einen deutlich höheren Wert zu setzen. Dies verhindert, das Patchcodes erkannt werden wo gar keine sind.
Achtung! Die Werte von "iBC_MinHeight" beziehen sich auch beim Patchcode wie bei allen anderen linearen Barcodes auf die Länge der Striche im Barcode. Da Patchcodes häufig "liegend", also als 4 waagerechte Linien gedruckt werden, kann es hier zu Mißverständnissen kommen.
Die Ruhezone **iBC_LightMargin** gibt wie bei anderen Barcodes den weißen Bereich vor und hinter den Patchcode-Strichen in Pixeln an. Dieser Parameter muss höher gewählt werden als die maximale Dicke eines Patchcode-Striches (also nicht nur höher als die Lückenbreite, sondern auch höher als die Dicke der schwarzen Striche). Die **iBC_Orientation** ist bei der Patchcode-Erkennung von hoher Wichtigkeit, weil der Barcodewert je nach Leserichtung ein unterschiedlicher ist. Der erkannte Patchcode wird wie üblich im Barcode-Result zurückgegeben. Die zurückgegebenen vierstelligen Werte (0/1 Kombinationen) können der nachstehenden Tabelle entnommen werden.

Tabelle der Patchcodes (1=dicke Linie, 0=dünne Linie)

Patch 1	1100
Patch 2	1001
Patch 3	1010
Patch 4	0110
Patch T	0101
Patch 6	0011

14.2 Besondere Einstellungen beim Data Matrix Code

Die Erkennung des zweidimensionalen Data Matrix Codes ist als Option zusätzlich erhältlich.
Zu beachten ist, dass einige Übergabe-Parameter beim Aufruf der Data Matrix Erkennung andere empfohlene Werte annehmen sollten:

iBC_LightMargin	DM_DEFAULT_LIGHT_MARGIN = 10
iBC_Percent	*irrelevant, nicht verwendet*
iBC_ScanDistBarcode	*irrelevant, nicht verwendet*
iBC_ScanDistance	DM_DEFAULT_SEARCH_DISTANCE = 10
iBC_Tolerance	DM_DEFAULT_TOLERANCE = 2
iBC_MaxNoVal	*irrelevant, nicht verwendet*
iBC_MinHeight	DM_DEFAULT_MIN_HEIGHT = 30
iBC_MaxHeight	DM_DEFAULT_MAX_HEIGHT = 2000

Neu ab Version 4.7: Die DataMatrix Erkennung verfügt über einen Modus "Intensive Suche". Dieser Modus erkennt durch Variation einiger interner Parameter bei verdreckten, verdrehten und leicht verzogenen DataMatrix einige Barcodes mehr. Allerdings auf Kosten der Erkennungszeit.

Bei QS-Barcode Versionen vor 4.7 war dieser Modus standardmäßig immer aktiviert. Nun ist er standardmäßig deaktiviert, kann aber gezielt wieder eingeschaltet werden. Nutzen Sie dazu den Parameter iBC_Orientation = DM_INTENSIVE_SEARCH (entspricht 0x00080000L). iBC_Orientation = 0 schaltet die intensive Suche aus. Beachten Sie dabei, dass die anderen Einstellungen, die bei linearen Barcodes für iBC_Orientation gewählt werden können, bei DataMatrix keinen Effekt haben.

14.3 Besondere Einstellungen beim QR Code

Die Erkennung des zweidimensionalen QR Codes ist als Option zusätzlich erhältlich. Zu beachten ist, dass die acht Übergabe-Parameter aus dem erweiterten Dialog (iBC_LightMargin, iBC_Percent, iBC_ScanDistBarcode, iBC_ScanDistance, iBC_Tolerance, iBC_MaxNoVal, iBC_MinHeight, iBC_MaxHeight) nicht von Bedeutung sind. Die Inhalte dieser Parameter werden von QS-Barcode bei der QR Code Erkennung nicht ausgewertet.

14.4 Besondere Einstellungen beim PDF417

Die Erkennung des zweidimensionalen PDF417 Codes ist als Option zusätzlich erhältlich.
Zu beachten ist, dass einige Übergabe-Parameter beim Aufruf der PDF417 Erkennung andere empfohlene Werte annehmen sollten:

iBC_LightMargin	PDF_DEFAULT_LIGHTMARGIN = 4 *wichtig: Nie verändern!*
iBC_Percent	*irrelevant, nicht verwendet*
iBC_ScanDistBarcode	*irrelevant, nicht verwendet*
iBC_ScanDistance	PDF_DEFAULT_SCANDISTANCE = 10
iBC_Tolerance	*irrelevant, nicht verwendet*

iBC_MaxNoVal	*irrelevant, nicht verwendet*
iBC_MinHeight	PDF_DEFAULT_MIN_HEIGHT = 15
iBC_MaxHeight	PDF_DEFAULT_MAX_HEIGHT = 2000

14.5 Besondere Einstellungen beim Aztec Code

Die Erkennung des zweidimensionalen Aztec Codes ist als Option zusätzlich erhältlich. Zu beachten ist, dass die acht Übergabe-Parameter aus dem erweiterten Dialog (iBC_LightMargin, iBC_Percent, iBC_ScanDistBarcode, iBC_ScanDistance, iBC_Tolerance, iBC_MaxNoVal, iBC_MinHeight, iBC_MaxHeight) nicht von Bedeutung sind. Die Inhalte dieser Parameter werden von QS-Barcode bei der Aztec Code Erkennung nicht ausgewertet.

14.6 Besondere Einstellungen bei PostNet Codes

Die Erkennung von 2-state und 4-state Post Ident Codes fällt unter die Lizenz für lineare Barcodes. Ohne gültige Lizenzoption L werden die Barcode-Ergebnisse durch den DEMO-Modus verfälscht.

Aktiviert wird die Erkennung von PostNet Codes über die richtige Typeinstellung im Parameter iBC_Type2. Es kann nur genau ein Typ aus der Liste der PostNet Codes zurzeit eingestellt werden.
Zu beachten ist, dass einige Übergabe-Parameter beim Aufruf der PostNet Erkennung folgende empfohlene Werte annehmen sollten:

iBC_LightMargin	PN_DEFAULT_LIGHT_MARGIN = 1
iBC_Percent	PN_DEFAULT_PERCENT = 0
iBC_ScanDistBarcode	*irrelevant, nicht verwendet*
iBC_ScanDistance	PN_ DEFAULT_SEARCH_DISTANCE = 4
iBC_Tolerance	PN_DEFAULT_TOLERANCE = 2
iBC_MaxNoVal	PN_DEFAULT_MAX_NO_VAL = 0
iBC_MinHeight	PN_DEFAULT_MIN_HEIGHT = 15
iBC_MaxHeight	PN_DEFAULT_MAX_HEIGHT = 200

Die Default-Werte können Sie auch variieren, dabei gelten die folgenden Hinweise:

iBC_LightMargin

Anzahl weißer Pixel, die mindestens vor jedem Strich im PostNet Code liegen. Gilt auch als Mindestwert für die Lücke zwischen den einzelnen PostNet Strichen. Eine Änderung des Standardwerts 1 auf einen höheren Wert bewirkt eine Beschleunigung der Erkennung, da weniger Objekte im Bild als mögliche Elemente in Betracht kommen.

iBC_ScanDistance

Abstand zwischen den einzelnen Abtastungen des Bildes in y-Richtung. Der Suchabstand wird für die Suche des Barcodes auf dem Bild verwendet. Dieser Wert darf 1/3 der "minimalen Höhe" nicht überschreiten.

iBC_Tolerance

Gibt die Toleranz im Hinblick auf Bildfehler in den einzelnen Strichen bei Lücken und Ausfransungen an den Kanten an.

iBC_MinHeight und iBC_MaxHeight

Die minimale Höhe muss auf einen Wert (in Pixeln) gesetzt werden, der unter der Höhe der kurzen "Tracker"-Striche des PostNet-Codes liegt. Die maximale Höhe muss größer sein als die Höhe eines "Full"-Strichs, sollte aber nicht übermäßig darüber liegen, da andernfalls zu viele andere Elemente auf dem Dokument zunächst als Teil des Barcodes in Betracht gezogen werden, was die Erkennung verlangsamt und erschwert.

iBC_Percent

Achtung: Dieser Parameter hat bei PostNet Codes die Bedeutung **"Min. Breite"**)
Gibt an, wie breit jeder einzelne Strich des PostNet Codes mindestens ist.
Ein Wert von 0 hat die Bedeutung beliebig. Eine Änderung des Standardwerts 0 auf einen höheren Wert bewirkt eine Beschleunigung der Erkennung, da weniger Objekte im Bild als mögliche Elemente in Betracht kommen.
Beachten Sie, dass schmalere Striche auf dem Bild dann nicht mehr beachtet werden, wählen Sie also auf keinen Fall einen zu hohen Wert!

iBC_MaxNoVal

Achtung: Dieser Parameter hat bei PostNet Codes die Bedeutung **"Max. Breite"**)
Gibt an, wie breit jeder einzelne Strich des PostNet Codes maximal ist.
Ein Wert von 0 hat die Bedeutung beliebig. Eine Änderung des Standardwerts 0 auf einen höheren Wert bewirkt eine Beschleunigung der Erkennung, da weniger Objekte im Bild als mögliche Elemente in Betracht kommen.
Beachten Sie, dass breitere Striche auf dem Bild dann nicht mehr beachtet werden, wählen Sie also auf keinen Fall einen zu niedrigen Wert!

Empfehlungen

Setzen Sie auf jeden Fall die Werte für maximale und minimale Höhe auf geeignete Werte, nur dann kann die Erkennung zuverlässig funktionieren.
Verändern Sie die Werte für Ruhezone, minimale Breite und maximale Breite nur, wenn Sie die Erkennungszeit reduzieren müssen. Positive Effekte im Hinblick auf die Erkennungsrate sind bei vom Default abweichenden Werten nicht zu erwarten, nur eine Beschleunigung. Werden ungeeignete Werte gewählt, beeinflusst dies natürlich eventuell die Erkennungsrate negativ.

15 Troubleshooting

15.1 ActiveX-Registrierung

Die Registrierung des ActiveX Controls kann bei Problemen auch manuell über den Befehl

```
regsvr32.exe OCX-Name.ocx
```

ausgeführt werden. Beachten Sie, dass zur Nutzung des OCX außerdem eine Visual Basic Runtime Umgebung installiert und alle notwendigen DLLs vorhanden sein müssen. Bei neueren Betriebssystemen müssen Sie für die Registrierung Administratorrechte besitzen.

15.2 DLL nicht gefunden

Wenn Sie beim Ausführen einer Komponente eine Fehlermeldung der Art "XXX.DLL nicht gefunden" bekommen, überprüfen Sie noch einmal, ob alle DLLs, die für die entsprechende Schnittstelle benötigt werden, in einem Verzeichnis liegen, in denen Ihr System nach DLLs sucht (z.B. Windows-Systemverzeichnis). Informationen über die von einer Schnittstelle benötigten Dateien finden Sie in den jeweiligen Auslieferungsdateien-Abschnitten.
Beachten Sie, dass eine Fehlermeldung "QSBarDLL_F.DLL nicht gefunden" auch bedeuten kann, dass eine von ihr benötigte DLL nicht gefunden werden konnte!

15.3 Alignment-Problematik

Alle Programme dieser Barcode-Library sind mit 8byte-Alignment kompiliert. Dies bedeutet, dass bei der Verwendung der p_lib das Projekt, in das die Library eingebunden wird, ebenfalls 8byte-Alignment verwenden sollte. Bei Schwierigkeiten können wir aber auch für Sie eine spezielle Version erstellen.
Bei Verwendung der DLL-Schnittstellen spielt das Alignment auch eine Rolle, wenn es um die Übergabe der Strukturen an die QS-Barcode DLLs geht. Das aufrufende Programm muss die Strukturen ggf. in der richtigen Größe übergeben.

15.4 Linker Problem

Bei der Verwendung von QS-Barcode SDK Schnittstellen (besonders bei QSBarLib.Lib, aber auch bei QSBarDLL_F.lib und QSBarDLL_H.lib) kann es unter MS-Visual Studio zu folgendem Linker-Fehler kommen:

Linker Tools Error LNK2005
<symbol> already defined in <object>

Die Ursache hierfür ist häufig, dass das QS-Barcode verwendende Projekt andere Linker-Einstellungen hat als das QS-Barcode SDK.
Alle QS-Barcode Libraries werden in vier Versionen ausgeliefert, die mit allen möglichen Compiler-Optionen "Use Run-Time Library" kompiliert wurden. Das einbindende Projekt muss dieselben Run-Time Libraries einbinden wie die genutzte QS-Barcode Library.

Es folgen Auszüge aus Microsoft-Quellen, die diesen Fehler betreffen:

Online-Hilfe zum Linker-Fehler

```
Linker Tools Error LNK2005
symbol already defined in object

The given symbol, displayed in its decorated form, was multiply defined.

Tips

One of the following may be a cause:

The most common cause of this error is accidentally linking with both the
single-threaded and multithreaded libraries. Ensure that the application
project file includes only the appropriate libraries and that any third-
party libraries have appropriately created single-threaded or multithreaded
versions.


The given symbol was a packaged function (created by compiling with /Gy)
and was included in more than one file but was changed between
compilations. Recompile all files that include the symbol.


The given symbol was defined differently in two member objects in different
libraries, and both member objects were used.


An absolute was defined twice, with a different value in each definition.
This error is followed by fatal error LNK1169.
```

MSDN-Artikel "/MD, /MT, /LD (Use Run-Time Library)"
Link: https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2013/2kzt1wy3(v=vs.120)

15.5 Barcode wird nicht erkannt

- Überprüfen Sie mit Hilfe des unter **"**6 Das Programm bcTester**"** beschriebenen Programms, ob der Barcode mit Ihren Einstellungen gelesen werden kann (Download des Freeware Programms unter https://bctester.de).
- Lesen Sie unser Whitepaper "Barcode nicht erkannt – was tun?" (BCTipps.pdf), das Sie aktuell im Verzeichnis "Dokumentation" der QS-Barcode Testapplikation oder auf unserer Website im Download-Bereich finden.
- Probieren Sie den Barcode mit einer Abwandlung der Beispielprogramme zu lesen.
- Schicken Sie Ihre gescannten Bilder an QS QualitySoft (info@qualitysoft.de), wir analysieren Sie kostenlos für Sie!

15.6 Systematische Ergebnisverfälschungen

Befindet sich QS-Barcode im **Demo-Modus**, werden systematisch die Ergebnisse verfälscht, d.h. es werden folgende Umsetzungen gemacht:
3 -> 1, A,a -> Q,q , B,b -> S,s

Damit sich QS-Barcode nicht im Demomodus befindet, muss eine gültige **Lizenzdatei** (QSBC.LIC) vorliegen. Überprüfen Sie, ob diese Datei am richtigen Ort liegt.

Nach der Lizenzdatei werden die unten angegebenen Verzeichnisse wie folgt durchsucht:
1. Verzeichnis der Applikation
2. Aktuelles Verzeichnis
3. Windows System Verzeichnis
4. Windows Verzeichnis
5. Alle Verzeichnisse aus der Umgebungsvariable PATH

16 Anhang

16.1 Unterstützte Dateiformate

Für das Öffnen der Bilddateien folgenden Dateiformate wird die Open Source FreeImage.dll verwendet.
Den Source Code und Details zur FreeImage Public License 1.0 finden Sie im Internet unter http://freeimage.sourceforge.net .

Von QS-Barcode unterstützte Dateiformate bei Verwendung der Schnittstellen OCX oder F_DLL:

- BMP files
- Dr. Halo files *
- DDS files
- EXR files
- Raw Fax G3 files
- GIF files
- HDR files
- ICO files
- IFF files
- JBIG **
- JNG files
- JPEG/JIF files
- JPEG-2000 File Format
- JPEG-2000 codestream
- KOALA files
- Kodak PhotoCD files
- MNG files
- PCX files
- PBM files
- PGM files
- PNG files
- PPM files
- PhotoShop files
- Sun RAS files
- SGI files
- TARGA files
- TIFF files
- WBMP files
- XBM files
- XPM files

* only grayscale

Wollen Sie Dateien in einem nicht aufgeführten Format verarbeiten? Senden Sie uns ein Beispiel. Wir überprüfen, ob eine Verarbeitung möglich ist.

16.2 MultiPage-Unterstützung

MultiPage-Dateien sind Dateien, die mehrere Seiten enthalten. QS-Barcode SDK unterstützt MultiTiff Dateien sowie mehrseitige Adobe PDF Dokumente.

Seit QS-Barcode Version 3.7 wird für die Schnittstellen F_DLL und F_OCX die Barcodeerkennung auch auf Folgeseiten von MultiTiff-Dateien unterstützt. Bei QS-Barcode Version 3.8 kam die Unterstützung von Adobe PDF Dokumenten hinzu.

So nutzen Sie die MultiPage Unterstützung: Der Schnittstellen-Parameter ist mit dem Bilddatei-Namen wie folgt zu belegen:
"Bilddateiname<Seitennummer", also zum Beispiel C:\TEMP\MULTIPAGE.TIF<2, um auf die zweite Seite von C:\TEMP\MULTIPAGE.TIF zuzugreifen. Die Seitenzählung startet dabei bei 1. Der Aufruf "C:\TEMP\MULTIPAGE.TIF<1" wird dabei genauso behandelt wie "C:\TEMP\MULTIPAGE.TIF".
Wenn die angegebene Seite nicht existiert, wird die Barcode-Erkennung mit dem Rückgabewert BC_NO_SUCH_PAGE (Hexadezimal=0x00000013L; Dezimal=19) abgebrochen, weil das Dateiende erreicht ist.

16.3 Adobe PDF Dokumente – Spezielle Einstellungen

Bei der Verarbeitung von Adobe PDF Dokumenten muss QS-Barcode SDK zunächst eine Umwandlung der zu lesenden Seite des Dokuments in ein herkömmliches Raster-Bildformat vornehmen. Dies ermöglicht einen universellen Umgang mit PDF-Dokumenten, egal ob es sich um von Scannern erzeugte Dokumente mit eingebetteten Scan-Abbildern oder um aus Programmen heraus generierte PDF-Dokumente mit eingebetteten Schriftarten und anderen Objekten handelt.

Als Nutzer von QS-Barcode SDK können Sie diese Umwandlung bei Bedarf über eine Konfigurationsdatei QSBC.INI parametrisieren.
Platzieren Sie diese Datei in dasselbe Verzeichnis wie die QSBarDLL_F.DLL, die von Ihrer Endanwendung angesteuert wird.

Die Datei hat das folgende Aussehen:

```
[PDF]
DPI=200
```

Die hier angegebenen Parameter sind die Default-Parameter, die auch genutzt werden wenn keine Datei QSBC.INI gefunden wurde.

DPI = Auflösung, mit der das Dokument eingelesen wird. Wenn es Probleme mit der Lesung von Barcodes gibt ist es sinnvoll, mit höheren Werten zu arbeiten (beachten Sie aber: Sehr hohe Werte kosten viel Rechenzeit und Speicher. Außerdem müssen Sie den Erkennungsparameter iBC_Ruhezone ggf. anpassen). Der maximale DPI-Wert kann 400 betragen.

Eine Angabe der Farbtiefe beim Einlesen von PDF-Dokumenten ist nicht möglich. PDF-Dokumente werden immer mit 24 Bit Farbtiefe eingelesen.

16.4 Bildverbesserung über die "qsbc.ini"

Über die Konfigurationsdatei "qsbc.ini" lassen sich erweiterte Parameter zur Bildverbesserung vor der Barcodeerkennung nutzen. Bis einschließlich Version 4.8.1 musste diese "qsbc.ini" im Programmverzeichnis des aufrufenden Programmes liegen. Ab Version 4.8.2 gibt es zusätzliche Schnittstellen die es ermöglichen, bei der Barcodeerkennung explizit eine "qsbc.ini" zu übergeben. Diese Datei muss jetzt auch nicht mehr "qsbc.ini" heißen sondern kann einen beliebigen Dateinamen haben und in einem beliebigen Verzeichnis liegen. Somit ist es jetzt möglich, die Erkennung verschiedener Barcodetypen und -Qualitäten noch feiner zu steuern.

Folgende Aufrufe können ab Version 4.8.2 dafür benutzt werden:

Library mit Pointer (p_lib)

```
int QSBarcode3( BarcodeParam2 *pBarcodeParam2, char *szIniFile );
```

DLL mit Handle (h_dll)

```
int QSReadBarcode3(HANDLE hDIB, Barcode2 *pBarcode2,
     int iNumResults, char *szIniFile);
```

DLL mit Datei (f_dll)

```
int QSReadBarcode3 (char *szImageName, Barcode2 *pBarcode2,
     int iNumResults, char *szIniFile);
```

Wenn die bei den obigen Funktionen die übergebene "qsbc.ini" nicht existiert, wird der Fehlercode `BC_QSBCINI_NOT_EXISTS` zurückgegeben.

Das Verhalten bezüglich der "qsbc.ini" bei Benutzung von `QSBarcode2()` und `QSReadBarcode2()` hat sich nicht geändert.

Beispiel:

Gelegentlich kommt es z.B. durch eine schlechte Scanqualität vor, dass ein Barcode schwarze Schmutzpunkte oder kleine weiße Pixellücken in den schwarzen Barcodebalken aufweist.

Diese Lücken in den schwarzen Balken können die korrekte Erkennung von Barcodes verhindern. Über die Bildverbesserung "Bildfehler-Ausgleich" lassen sich diese Lücken schließen, so dass eine Erkennung des Barcodes wieder möglich ist.

Hierzu benötigt man folgenden Eintrag in einer "qsbc.ini":

```
[CleanBresenham]
WhitePoints=1
BlackPoints=0
```

Diese "qsbc.ini" speichert man unter beliebigem Namen, z.B `"d:\barcode\ini\qsbc_weisse_pixel.ini".`

Der Aufruf zum Lesen des Barcodes kann dann z.B. wie folgt lauten:

```
QSReadBarcode3 (szImageName, pBarcode2,iNumResults,
     "d:\barcode\ini\qsbc_weisse_pixel.ini")
```

(Zu den einzelnen Parametern in der "qsbc.ini" siehe auch das Dokument "\sdk_doku\bcTipps.pdf".)

16.5 QR Code Extended Channel Interpretation (ECI) Mode

Die QR Code Spezifikation bietet die Möglichkeit, im ECI-Modus die in einem QR Code vorhandenen Daten über verschiedene Zeichensätze zu interpretieren.

Die bisherige Rückgabe der Daten erfolgte z.B im Format "]Q2\000009ÁÂÃÄÅ". Die Zeichenfolge startete mit "]Q2\", gefolgt von ECI-Code (000009) und den zugehörigen Daten "ÁÂÃÄÅ". Diese Art der Rückgabe erfolgt weiterhin, wenn die folgend beschriebene Art der Rücklieferung nicht verwendet wird.

Um die Daten im ECI-Modus zu erhalten, müssen in der Übergabestruktur für die Barcodelesung folgende Parameter gesetzt sein.

```
bcParams2.iBC_Type = (int)BC_Types.BC_QR_CODE;
bcParams2.iBC_Type2 = (int)BC_Types.BC_QR_CODE_ECI;
```

Die Rückgabe der gelesenen ECI-Codes und deren zugehörige Daten erfolgen analog der Rückgabe der Daten von 2D Barcodes. Auch hier sind die Variablen in der erweiterten Struktur `BC_TwoDimRes` gefüllt. `hBC_TwoDimRes` enthält dabei einen Handle auf einen Speicherbereich, in dem die ECI-Codes und deren zugehörige Daten in einem ggf. multidimensionalen Array vom Typ `QRECIResult_tag` stehen. Das komplette Ergebnis der Gesamtgröße aller Strukturen steht ebenfalls in `iBC_TwoDimLen`. Die Anzahl der Strukturen im Handle lässt sich somit über die Gesamtgröße im Verhältnis zur Strukturgröße ermitteln.

Der Aufbau der Struktur ist wie folgt:

```
typedef struct QRECIResult_tag
{
      Int    iECI;                    // 4 BYTE: ECI Type
      void   *hBC_ECI_Item;           // 32-bit: 4 BYTE / 64-bit 8 BYTE:
                                      // ECI value stored in a handle
      Int    iBC_ECI_ItemLength;      // 4 BYTE: Length of hBC_ECI_Item
}QRECIResult;
```

Der ECI-Code wird in `iECI` abgelegt, die zugehörigen Daten befinden sich im Handle `hBC_ECI_Item`, die Datenlänge wird in `iBC_ECI_ItemLength` abgelegt.

Falls ECI-Codes gesucht werden, es sich aber keine im Barcode befinden bzw. es sich eine Mischung von ECI-Daten und non ECI-Daten im Barcode befindet, werden die Daten trotzdem in der ECI-Struktur zurückgegeben. Bei non ECI-Daten wird der ECI-Code lediglich auf den Wert "0" gesetzt.

Details zum Auslesen der Daten entnehmen Sie bitte dem Programmbeispiel.

16.6 Update-Hinweise

Dieses Kapitel enthält wichtige Hinweise für alle Kunden, die QS-Barcode SDK 4.5 oder älter nutzen und auf QS-Barcode 5.0 updaten wollen!
Mit QS-Barcode 4.6 wurden neue Versionen der Schnittstellen eingeführt, die auch für QS-Barcode 5.0gelten.

Bei der Umstellung von der 32-Bit auf die 64-Bit Version des QS Barcode SDK genügt es, den Quellcode einmal neu zu kompilieren. Änderungen am Quellcode müssen nicht vorgenommen werden. Sie müssen lediglich die benötigten DLLs für die richtige Plattform ausliefern.

Zunächst einmal: Die Schnittstellen **f_dll**, **h_dll** und **p_lib** bieten nach wie vor die dieselbe Schnittstelle an, wie sie unter Version 4.5 und früher genutzt wurde.
Sie können also ein Update ohne Programmänderungen vornehmen.
Die **f_ocx** Schnittstelle bietet nur die neue, geänderte Schnittstelle. Sie werden hier um eine Anpassung in Ihrem Programm nicht herumkommen, wenn Sie das neueste OCX nutzen wollen. Allerdings haben Sie auch die Möglichkeit, das Version 4.5 OCX weiterhin zu nutzen, aber die QS-Barcode 4.7 DLLs mit diesem alten OCX auszuliefern. Da die eigentliche Barcode-Erkennung bei der Nutzung des OCX in der QSBarDll_F.DLL erfolgt, profitieren Sie so trotzdem von den Verbesserungen der Version 5.0.

Nur wenn Sie komplett auf die neuen Schnittstellen-Versionen umstellen, haben Sie die Möglichkeit, die neuen GS1 Databar Barcode-Typen zu lesen. Auch wenn QS-Barcode in zukünftigen Versionen um die Erkennung weiterer Barcode-Typen ergänzt wird, werden Sie die Erkennung dieser Typen nur mit den neuen Schnittstellen-Versionen nutzen können.

Der Grund für die Schnittstellen-Änderung ist einfach: Die Grenze von 32 unterstützten Barcodetypen wurde inzwischen überschritten, das 4-Byte Feld iBC_Type reicht somit nicht mehr, um alle Typen zu steuern.

Alle Schnittstellen haben daher einen neuen 4-Byte Parameter iBC_Type2 erhalten, um mehr Platz für neue Typen zu schaffen. Dies betrifft dann natürlich auch die Result-Struktur.
Bei der Namensvergabe haben wir alle so erweiterten Strukturen und Funktionen mit einem "2" am Ende kenntlich gemacht.

Eine weitere Änderung hat es gegeben: hBC_ErrorFile in iBC_Debug, zwei Parameter der Schnittstelle, die früher für Log-Ausgaben zuständig sind, haben nun keine Bedeutung mehr. Sie sollten mit NULL bzw. 0 initialisiert werden.
Unserem Wissen nach haben sehr wenige Kunden diese Log-Parameter jemals verwendet. Diese können sich gern an uns wenden, um sich über die alternativen Log-Verfahren der aktuellen Version zu informieren.

Schlussbemerkung

QualitySoft bemüht sich, diese Dokumentation immer aktuell zu halten. Hinweise auf Fehler oder unverständliche Stellen nehmen wir gern entgegen. QS-Barcode SDK wird stetig gepflegt und verbessert. Änderungen am Produkt sind so jederzeit möglich.

QS QualitySoft GmbH
Tempowerkring 21a
21079 Hamburg

Tel: +49 (0) 40 790 100 40

info@qualitysoft.de
https://qualitysoft.de