高质量条形码应用开发 SDK (V 2.1.0.409)

1、 Barcode.dll 条码应用开发库简单介绍----------

Barcode.dll 大约 **25k** 左右，其中还包含版本信息、条码名称信息等等辅助数据。**Barcode.dll** 用不到 **20k** 的程序代码，封装了数十种条码的编码和绘制功能。可能，你所知道的其他条码应用开发组件，有数 **M** 大甚至十几 **M** 大！相比这些巨型的条码开发组件而言，**Barcode.dll** 代码的编写是非常精练的、非常高效的，运行速度也是首屈一指的。尤其，其他的条码应用开发组件，大多以位图输出的形式给其他应用软件提供接口，这很不便于用户机动编程，应用灵活性小。再者，以位图为接口数据的情况下，是不利于大批量打印条码的，原因很简单，一张 **600dpi** 的单色条形码位图，数据量就有几百 **k** 大，而一张 **600dpi** 的彩色条码位图，则可能有十几 **M** 大，一张 **1440dpi** 的条形码的彩色位图，则可能在 **50M** 以上！如果应用软件需要批量制作 **10000** 张条形码标签，采用位图为接口数据的话，总数据量可能高达数百 **G**，可以想象，要在计算机上生成数百 **G** 的位图数据，并把这数百 **G** 的位图数据转换成打印数据再传送到打印机，是一项多么艰巨、费时和消耗资源的任务！**Barcode.dll** 则以矢量矩阵数据为接口，一张 **2880dpi** 甚至更高分辨率的条码，不论是彩色还是单色，数据量都在 **7k** 左右！若使用 **Barcode.dll** 输出 **10000** 张条形码，总数据量仅 **70M** 左右！所以，对于大批量、高精度、高速度打印条码的应用，**Barcode.dll** 是有得天独厚的优势。

2、 Barcode.dll 条形码应用开发库的使用特性----------

Barcode.dll 的应用非常简单，因为 **Barcode.dll** 只有一个主函数 **DrawBarcode** 和一个辅助函数 **GetBarcodeNames**（该辅助函数用户可用也可不用）。但请注意：**Barcode.dll** 是基于 **Unicode** 的！

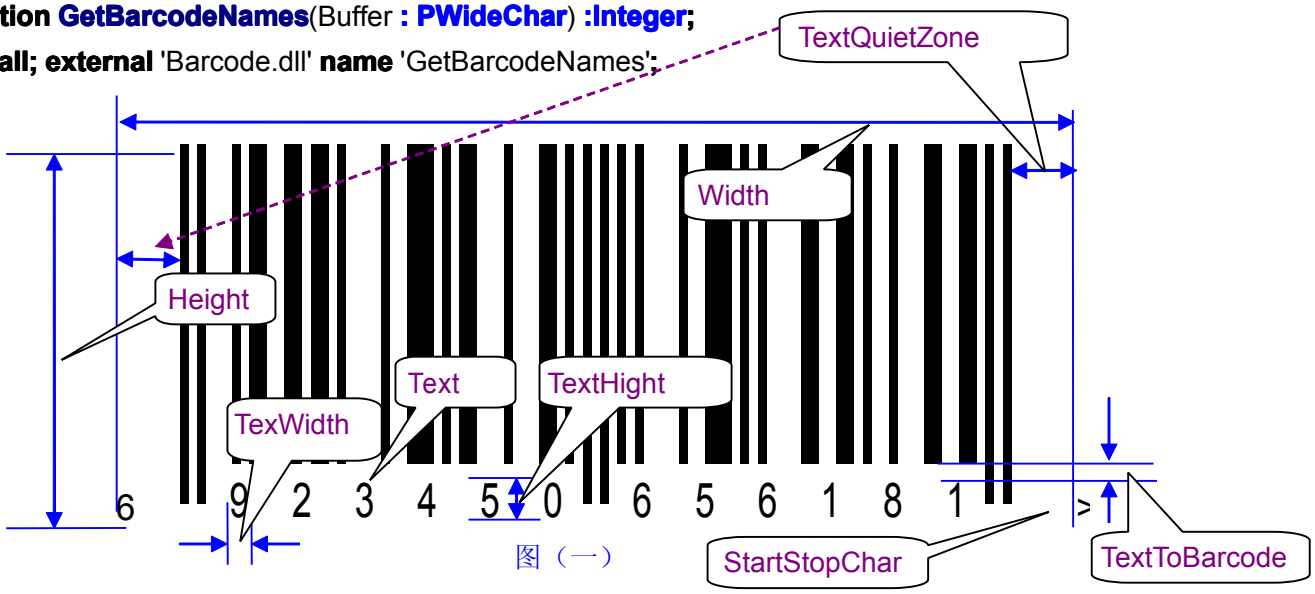
3、 Barcode.dll 的两个函数的声明----------

VC++声明：

```
DWORD DrawBarcode (HDC DC, WCHAR * Text, WCHAR * StartStopChar, WCHAR * FaceName,
    WCHAR * Buffer, RECT * Bounds, RECT * BarcodeMatrix,, int TextWidth, int TextHeight,
    int TextToBarcode, int TextQuietZone, int BarInflate, DWORD Color, DWORD Params);
int GetBarcodeNames (WCHAR * Buffer);
```

Delphi 声明：

```
function DrawBarcode(DC :HDC; Text, StartStopChar, FaceName, Buffer : PWideChar;
        Bounds, BarcodeMatrix : PRect; TextWidth, TextHeight : Integer;
        TextToBarcode, TextQuietZone, BarInflate : Integer; Color, Params : DWord) :DWord;
stdcall; external 'Barcode.dll' name 'DrawBarcode';
function GetBarcodeNames(Buffer : PWideChar) :Integer;
stdcall; external 'Barcode.dll' name 'GetBarcodeNames';
```



图（一）

上图说明了 **DrawBarcode** 函数部分参数的意义。其他参数的意义说明如下：

DC：设备环境。要把条码绘制在窗口，传递窗口的 **DC** 即可，要打印条码，传递打印机的 **DC** 即可。可为 **0**。

Bounds：该参数确定条码绘制在 **DC** 上的位置**(Left、Top)**和尺寸**(Width、Height)**；

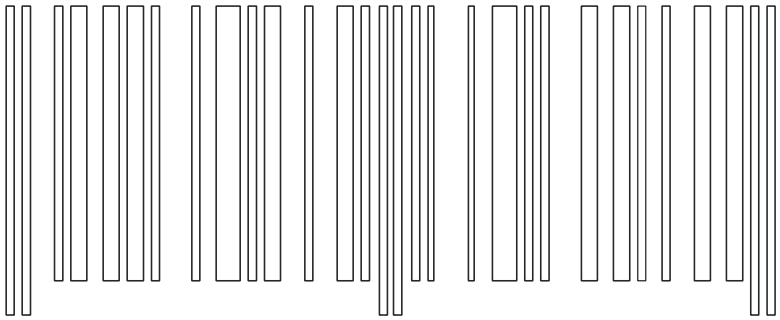
FaceName：条码文本的字体名；

Color：条码的绘制颜色；在 **Delphi** 中，传递该参数请使用 **ColorToRGB ()**把 **TColor** 转换成 **RGB** 类型。

BarInflate：条码 **Bar** 加粗或缩小。条码中，一根黑线条即一个 **Bar**。若 **BarInflate** 为正数，则加粗 **Bar**，若 **BarInflate** 为负数，则减细 **Bar**。该特性主要用于适配不同的打印介质，比如条码的打印一般用激光打印机或热升华打印机，不推荐喷墨打印机，因为用喷墨打印机打印条码时，可能由于墨水的渗透，导致 Bar 变粗，使打印出的条码，条码阅读器不好阅读或者根本无法阅读，此时可调整 BarInflate，使打印出来的条码，合乎规范，条码阅读器可阅读。所以，该特性仅用于修正打印介质的误差，若在屏幕上绘制条码，应使 **BarInflate = 0**；

Buffer：这个参数是一个缓冲区，主要用于自绘条码的文本。比如图（一）中的 **EAN-13** 条码，其数据位是 **12** 位，即 **692345065618**，但我们发现，条码上显示的数据却是 **6923450656181**，是 **13** 位，这个附加的数据 **1** 称为 **CheckDigit**（校验位），若用户想自绘条码文本的话，必须懂得各种条码的 **CheckDigit** 如何计算，才能自绘条码的文本，这是件比较讨厌的事。但 **Barcode.dll** 会把处理好的条码文本存放在 **Buffer** 参数中，用户只需从 **Buffer** 中取出已处理好的文本进行绘制即可，根本不必知道各种条码的 **CheckDigit** 的计算方法。若用户并不需要这个文本，则可传递一个空指针（**VC++：NULL；Delphi：nil**）。

BarcodeMatrix：这个参数是一个缓冲区，用于接收条码的矢量矩阵数据，主要用于自绘条码。条码由一根根粗细不同的黑线条组成，但我们可以提取出所有黑线条的轮廓，从而形成一个矩形阵列，这个矩形阵列就是条码的矢量矩阵。如下图（二），就是图（一）的条码的矢量矩阵，**BarcodeMatrix** 中存放的数据，就是矢量矩阵中的各个矩形的参数（即矩形的 **Left、Top、Right、Bottom**）。



图（二）

获取到条码的矢量矩阵数据，就可对条码进行一些特定的处理。比如转化为 **PDF** 的格式绘制到 **PDF** 文档，转化为 **DXF** 格式插入到 **AutoCAD** 文档，还可对矩阵进行旋转，绘制任意角度的条形码，还可控制条码每个 **Bar** 的颜色，制作彩条的条码，如图（三）……

图（三）

若用户需要自绘条码，则传递一个 **0** 值给 **DrawBarcode** 的 **DC**。若并不需要 **BarcodeMatrix** 数据，则可传递

一个空指针（**VC++：NULL；Delphi：nil**）给 **BarcodeMatrix**。

Params：这是 **DrawBarcode** 中最复杂的参数，也是一个复合型参数。**Params** 是一个 **DWORD** 参数，一个 **DWORD** 有 **4** 字节共 **32 bit**，我们把 **Params** 的各 **bit** 从低到高编号为 **0** 到 **31**。

0 bit – 5 bit：该 **6 bit** 表示条形码类别（取值为 **0-34**），**Barcode.dll** 共支持 **35** 大类条形码如下：

0：Code 25-Interleaved(ITF)
1：Code 25-Standard(Industrial)
2：Code 25-Matrix
3：Code 39-Regular
4：Code 39-Full ASCII
5：Code 128A
6：Code 128B
7：Code 128C
8：Code 128 Auto (**V2.1.0.409** 新增)
9：Code 128 Custom (**V2.1.0.409** 新增)
10：Code 93-Regular
11：Code 93-Full ASCII
12：MSI/Plessey
13：USPS PostNet(Zip,Zip+4,DPBC)
14：Codabar/USD-4/NW-7(A,B,C,D)
15：EAN/JAN-8
16：EAN/JAN-13
17：UPC A
18：UPC E0
19：UPC E1
20：UPC/EAN/JAN Supplemental-2
21：UPC/EAN/JAN Supplemental-5
22：GS1/EAN/UCC-128A
23：GS1/EAN/UCC-128B
24：GS1/EAN/UCC-128C
25：GS1/EAN/UCC-128 Auto (**V2.1.0.409** 新增)
26：GS1/EAN/UCC-128 Custom (**V2.1.0.409** 新增)
27：ISBN Bookland-(13 + 5)
28：Datalogic 25/China Postal Code （注：中国邮政专用条码）
29：Code 11/USD-8
30：Coop 2 of 5
31：IATA 2 of 5
32：USPS PLANET
33：Royal Mail 4-state Customer Code
34：KIX 4-state Customer Code

Barcode.dll 支持以上 **35** 大类条形码，如果象某些条形码组件一样，以细化的分类方式为基准，则 **Barcode.dll** 支持的条形码种类，那就更多了。比方在某些条形码组件中，有 **Code 25-Interleaved(ITF)**，还有 **ITF-14**、**ITF-6** 等，实际上，**ITF-14**、**ITF-6** 只不过是 **14** 位数据、**6** 位数据的 **Code 25-Interleaved** 加个黑边框！再比如有的条码组件，有 **PostNet2**、**PostNet5**、**PostNet10** 等条码，实际上只是不同位数的 **USPS PostNet** 条码。**Barcode.dll**

是以编码和绘制为核心，而不是具体到每一个条形码子类，比如用户可使用 **Barcode.dll** 生成 **PostNet2**、**PostNet5**、**PostNet10**，还可生成 **PostNet7/8/9……N**！

6 bit – 7 bit：保留未用。

8 bit – 9 bit：条码旋转方式（取值为 **0 - 3**）。

0：不旋转（水平样式）

1：逆时针旋转 **90** 度

2：顺时针旋转 **90** 度

3：旋转 **180** 度

10 bit – 11 bit：保留未用。

12 bit：条码是否附加校验数据。为 **1** 则附加校验数据，为 **0** 则不附加校验数据。有些条码必须附加校验数据，比如 **Code 128A/B/C/Auto/Custom**、**GS1/EAN/UCC-128A/B/C/Auto/Custom** 等，有的条码则具有自校验功能，一般性应用无需附加校验位，高可靠要求的应用则需要附加校验位，比如 **Code 25-Interleaved(ITF)**、**Code 39** 等，还有的条形码则无需校验位，比如 **UPC/EAN/JAN Supplemental-2/5**。

13 bit – 15 bit：保留未用。

16 bit：是否绘制条码的文本。为 **1** 则绘制条码的文本，为 **0** 则不绘制条码的文本。

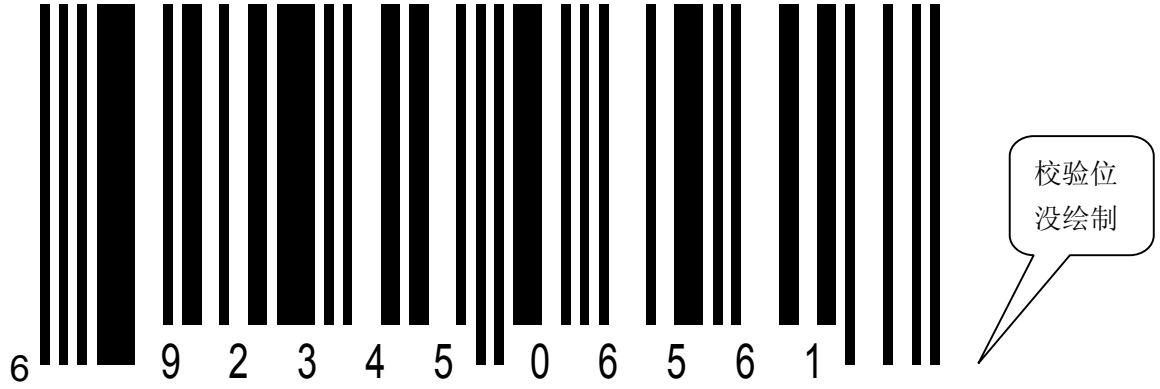
17 bit：是否缩减某些条码（具体指 **EAN/JAN-13**、**UPC A**、**UPC E0**、**UPC E1**、**ISBN Bookland-(13 + 5)**）的头、尾字符的高度。为 **1** 则缩减头尾字符的高度，为 **0** 则不缩减。如下图（四）是 **UPC A** 条码，绘制时缩减了头字符 **6** 和尾字符 **7** 的高度。

图（四）

18 bit：条码的文本是否以粗体绘制。为 **1** 则以粗体绘制，为 **0** 则正常绘制；

19 bit：条码的文本是否以斜体绘制。为 **1** 则以斜体绘制，为 **0** 则正常绘制；

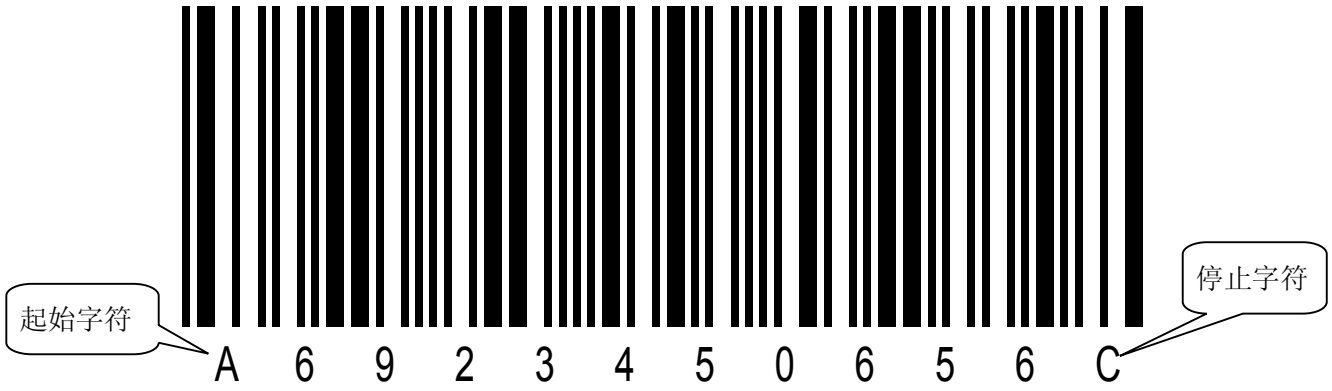
20 bit：绘制条码文本时，是否绘制校验位（**CheckDigit**）。为 **1** 则以绘制校验位，为 **0** 则不绘制校验位；如下图（五）是上图（四）不绘制校验位时的图样。



图（五）

21 bit：有起停字符的条码（具体是这些条码：**Code 39-Regular**、**Code 39-Full ASCII**、**EAN/JAN-13**、**ISBN**

Bookland-(13 + 5)、**Codabar/USD-4/NW-7(A,B,C,D)**），是否绘制条码的起停字符（即 **StartStopChar**）。为 **1** 则绘制起停字符，为 **0** 则不绘制起停字符；如下图（六），就是一个有起停字符的 **Codabar** 条码，它的起始字符是 **A**，而停止字符是 **C**。



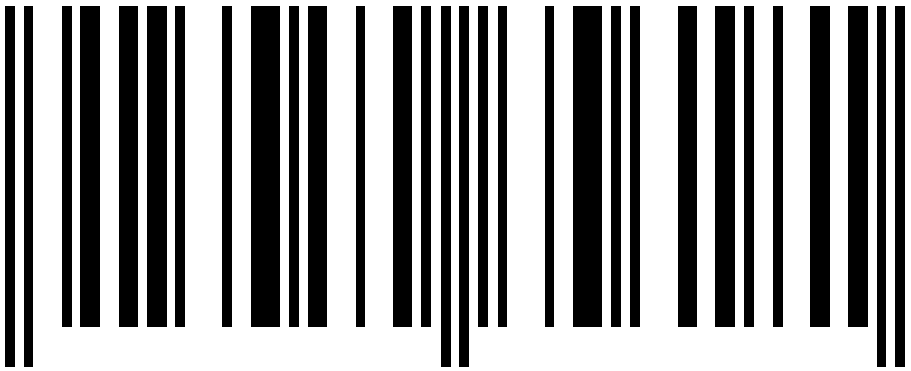
图（六）

注：1、**Code 39-Regular**、**Code 39-Full ASCII** 的起停字符规定都是 *****，**Barcode.dll** 会自动附加该字符；

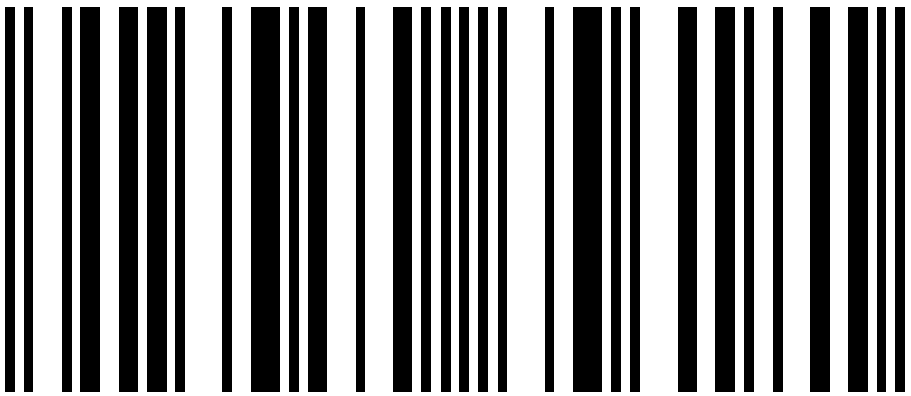
2、**Codabar/USD-4/NW-7(A,B,C,D)** 的起停字符规定是 **A**、**B**、**C**、**D** 四个字符之一，可由用户设定并由参数 **StartStopChar** 传递给 **DrawBarcode** 函数。**StartStopChar** 第一个字符为起始字符，第二个字符为停止字符。若无起（或停）字符，则把起（或停）字符设为空字符 **Chr(0)**即可。

3、**EAN/JAN-13**、**ISBN Bookland-(13 + 5)**这两类条码只有停止字符，且可设定为任意字符。

22 bit：有规定样式的条码（具体指 **EAN/JAN-8**、**EAN/JAN-13**、**UPC A**、**UPC E0**、**UPC E1**、**ISBN Bookland-(13 + 5)**），不绘制文本时，是否保持规定的样式。为 **1** 则保持规定的样式，为 **0** 则不保持样式。下图（七）是不绘制文本但保持条码规定样式的，下图（八）则是不绘制文本也不保持样式的。



图（七）：不绘制条码文本但保持规定样式



图（八）：不绘制条码文本也不保持规定样式

该特性允许用户按照自己的文本绘制样式，来绘制条码的文本。

23 bit – 31 bit：保留未用。

DrawBarcode 的返回值：**DrawBarcode** 的返回值是一个 **DWORD** 类型，复合了两个数据：低 **16 bit** 是条码文本（可能附加了 **CheckDigit**）的长度，而高 **16 bit**，则是条码矢量矩阵中矩形的数量，但若 **BarcodeMatrix** 为 **NULL**（或 **nil**），则返回值的高 **16 bit**，始终是 **0**。利用返回值，用户很容易从 **Buffer** 中提取到条码的文本进行绘制，也可知道 **BarcodeMatrix** 中，有多少个矩形需要处理。假设 **R** 是 **DrawBarcode** 的返回值，**X**、**Y** 是两个变量，用于存放条码文本长度和条码矩阵中矩形的个数，那么：

X = R 且 **0x0000FFFF**（**Buffer** 中有 **X** 个条码字符）

Y = R 右移 **16** （**BarcodeMatrix** 中有 **Y** 个矩形）

调用 **DrawBarcode** 前，如何给 **Buffer** 和 **BarcodeMatrix** 分配内存呢？很简单，给 **Buffer** 分配 **64** 个字符的空间足够，因为一般不会有超过 **64** 字符的 **1D** 条码。给 **BarcodeMatrix** 分配 **8 x 64 = 512** 个矩形的空间足够。

函数 **GetBarcodeNames** 说明：该函数可获得 **Barcode.dll** 所支持的 **31** 大类条形码的名称。**GetBarcodeNames** 只有一个参数 **Buffer** 用于接收 **31** 大类条码的名称，各个名称之间以回车字符 **Chr(13)**分割。获得的 **Buffer** 中的字符串结构如下：

Code 25-Interleaved(ITF)**Chr(13)**……**Chr(13)**KIX 4-state Customer Code**Chr(13)**，应用时可根据 **Chr(13)**对 **Buffer** 进行分割，获得条码的名称列表。

GetBarcodeNames 的返回值：若调用正常应为 **31**（即 **31** 大类条形码）！利用该返回值，用户在分割 **Buffer** 时，若已获得 **31** 个条形码名称，即可及时对循环 **break**。

4、使用 Barcode.dll 易犯的错误

Barcode.dll 是基于 **Unicode** 的，在 **VC++**中使用，所有字符串相关的函数参数，应使用 **WCAHR**，而不能是 **char**。而在 **Delphi** 中使用，所有字符串相关的函数参数，应使用 **PWideChar**，而不能是 **PAnsiChar**！随着 **Windows 98** 的消失，基于 **AnsiChar** 的应用程序，已经越来越没有了市场。

5、试用 Barcode.dll

可能你在使用其他条形码组件的时候，有过沮丧的感觉。其原因可能因为以下情况而产生：

1）你试用各种条码组件生成某种条码，会发现生成的条码竟然有些不同，你不知道究竟哪个组件生成的条码是对的，所以，你也不知道究竟选择哪个条码组件。

2）你打印出的条码，用眼睛检查，看起来是对的，可条码阅读器却很难读出或者根本读不出。

3）你打印条码的打印机精度很高，在 **1200dpi** 以上，你以为你这样好的打印机，定能打印出高质量的条码，可是你打印时，计算机却告诉你“存储空间不足，无法处理……”！

4）在某些应用场合，需要打印超小型的条码，你虽然拥有高精度的打印机，也拥有高精度的条码阅读器，可你打印出的小型条码，高精度的条码阅读器也认不出来；

5）你想使用廉价的喷墨打印机打印条码，然而你打印出的条码，条码阅读器很难识别。

……

所以，在准备使用 **Barcode.dll** 时，请下载作者编写的 **FreeBarcode** 软件，这个软件提供了各种条码输出方式：位图文件输出、**EMF** 矢量文件输出、矢量矩阵数据直接输出到剪贴板、打印输出。尤其，矢量矩阵数据输出到剪贴板后，你可把条码的矢量数据，直接粘贴到 **CorelDraw**、**Illustrator**、**Word**、**Excel**、**WPS** 等常见的应用软件中，并保持其矢量特性。也就是说，有了 **FreeBarcode** 软件，在 **CorelDraw**、**Illustrator**、**Word**、**Excel**、**WPS** 等软件中，即可制作完全矢量化的条码。**FreeBarcode** 就是基于 **Barcode.dll** 开发的，是完全免费的，且支持 **Vista** 和 **Windows 7**。请再下载国外著名的条码标签制作软件 **BarTender**，对比 **FreeBarcode** 制作出的条形码，是否和 **BarTender** 制作出的条码完全一样（对比时请注意校验位，如果一方附加了校验位，而另一方没附加校验位，那肯定不一样），以检查 **Barcode.dll** 对各种条码的编码是否合乎通用规范（当然，作者开发 **Barcode.dll** 时，已全部检查过，但你不妨再次检查一下）。需要提醒用户的是：一定要和著名的条码软件对比检查，不可与那些名不见经传的小软件对比，否则你自己都会弄糊涂。若你对 **Barcode.dll** 有了足够的信心，你就可以试用 **Barcode.dll** 了。首先，创建一个名为 **FreeBarcode** 的

工程（试用时必须使用该工程名，否则你的程序不能加载 **Barcode.dll**），再把 **Barcode.dll** 拷贝到你的工程目录下，这样就可试用 **Barcode.dll** 了。

试用时，你可参照作者提供的 **Delphi** 和 **VC++**源代码。但并非是 **Barcode.dll** 仅能在 **VC** 和 **Delphi** 里使用，而是可在所有的编程语言里使用（比如 **VB**、易语言、**PB** 等等），因为 **Barcode.dll** 是标准的 **API** 封装，而所有编程语言都是支持调用 **API** 的。

特别说明：**Barcode.dll** 是非常适合于多核多线程应用的，且已经支持到 **Vista** 和 **Windows 7**！

6、购买 Barcode.dll 授权----------

1） **Barcode.dll** 全功能的价格为人民币 **1500** 元/授权。这里所说的每个授权，是指授权在一个指定的应用程序（比如 **MyApp.exe**）中使用，也就是说，使用 **Barcode.dll** 的应用程序不论发售了多少份，都只需购买一个授权。在国内，现在不少组件开发商所谓的授权，是指装机量，比方 **A** 软件使用 **B** 组件，**A** 软件的作者每卖出一份软件，应支付 **B** 组件的作者一份授权费。所以，当你看见 **2000** 元/**50** 个授权时，应该引起注意：这是看起来很便宜，其实很贵的授权，尤其当你的软件销量较大时。而 **Barcode.dll** 的授权，是一个应用程序，无论发售多少份，都只需购买一个授权。

2） 若用户的应用程序只需 **Barcode.dll** 中某一种或某几种条码，则按人民币 **200** 元/条码/授权的价格。比方某应用程序，只想使用 **EAN -13** 条码，则只需支付 **200** 元即可。若用户需要的条码种类超过了 **7** 种，请购买全功能版本的授权。

注：未经授权请勿擅自在您的软件中使用 **Barcode.dll**，除非您的软件只是编写给自己使用的。若您的软件是商业软件或者是共享软件，请自觉购买 **Barcode.dll** 的授权。

7、定制特定样式的条码输出----------

用户可定制特定的条码输出样式。需定制的用户，除授权费之外，如果需定制的条码属于 **Barcode.dll** 所支持的 **31** 大类之中，定制费收取标准为人民币 **500** 元/种条码；若需定制的条码类型不在 **Barcode.dll** 所支持的 **31** 大类之中，定制费收取标准为人民币 **1000** 元/种条码，当然，用户还得提供该条码的完整资料（英文版或中文版均可），否则不予定制，因为作者所能收集到的条码的技术资料，已全部封装在 **Barcode.dll** 中，其他条码的资料，尚不能收集到或者收集不完整，无法完成其编码。

8、2D 条形码开发库 PDF417Enc.dll、QRCodeEnc、DMatrixEnc.dll----------

以上介绍的是一维条形码开发库 **Barcode.dll**，我处还推出了国内常用的 **3** 大类二维条形码的开发库 **PDF417Enc.dll**、**QRCodeEnc**、**DMatrixEnc.dll**，它采用和 **Barcode.dll** 一样精练的编码方式，一样的矢量矩阵式数据接口，创造 **2D** 条码的全矢量、高精度、高速、高效率的应用。

---------- （完）

开发者：李辉宇
电　话：135 8886 7730
QQ　：1497 96232
网　站：http://www.3wcad.com

附录 A：V2.1.0.409 升级说明

1、Barcode.dll 的 **V2.1.0.409**，对 **Code 128A/B/C** 和 **GS1/EAN/UCC 128A/B/C** 进行了拓展，可以对 **ASCII** 控制字符进行编码。在一维条码中，**Code 128A/B/C** 和 **GS1/EAN/UCC 128A/B/C**，是相对复杂的条形码，每个码字有 **11** 个 **Modules**，且共有 **106** 个码字，既对可见字符编码，同时也对不可见的控制字符编码。事实上，几乎所有条码软件都支持 **Code 128** 和 **EAN 128** 条码，但 **90%**的条码软件是只支持一半，也就是仅支持对可见的字符进行编码。可能有的用户遇到过这种情况：客户拿来的样品条码，显而易见是 **Code 128** 或 **EAN 128** 条码，原样输入条码上的文字，出来的条码却与样品完全不一样！这就是带不可见控制字符的条形码。**Barcode.dll V2.1.0409**，支持输入控制字符，输入方法是：\控制字符名。

共有 **32** 个控制字符：

\NUL、\SOH、\STX、\ETX、\EOT、\ENQ、\ACK、\BEL、\BS、\HT、\LF、\VT、\FF、\CR、\SO、\SI、\DLE、\DC1、\DC2、\DC3、\DC4、\NAK、\SYN、\ETB、\CAN、\EM、\SUB、\ESC、\FS、\GS、\RS、\US

输入控制字符时，一定要记得加“****”并使用大写字母。比方要设计一个带回车字符的 **128** 码（比如 **123456ABCDEF** 回车 **abcdef123456**），我们应输入：**123456ABCDEF\CRabcdef123456**。条码制作出来之后，我们在条码上看见的数字是 **123456ABCDEFabcdef123456**，所以 **128** 码，照着样品条码上的数字制作，可能会不对，因为它可能包含不可见的控制字符。

注：绝大多数条码软件是无法制作这种带控制字符的条码的，即便是一些大型条码制作软件，也少有支持的。

2、**Barcode.dll** 的 **V2.1.0.409** 新增了 **Code 128 Auto**、**Code 128 Custom**、**GS1/EAN/UCC 128 Auto**、**GS1/EAN/UCC 128 Custom** 四类条码。**Code 128 Auto**、**GS1/EAN/UCC 128 Auto** 是现在使用比较多的条码，因为单纯使用 **128A\B** 类条码，条码面积太大，而 **128C** 码使用了压缩方式，但只能编码阿拉伯数字符号，应用也受到限制。所以，现在流行 **Code 128 Auto**、**GS1/EAN/UCC 128 Auto**，该类条码混合了 **128A\B\C** 的优点，既能字母、标点、数字混排，还能使生成的条码最小。

所以，现在诸如 **JZMR897632** 这样的条码，你也许可以认出是 **128** 一类的条码，但它可能不是 **128A\B\C** 之中任何一个，而可能是 **128 Auto** 条码。

注：**A**、目前支持 **128 Auto** 类条码的软件也不是很多。

B、**128** 条码至少有 **4** 个基本类别：**128A**、**128B**、**128C**、**128 Auto**，现在用得最多的可能就是 **128 Auto** 条码，而这个 **128 Auto** 却只有少数软件支持。有的用户拿到一个其实是 **128 Auto** 类条码，用的软件却只有 **128A\B\C**，往往是折腾了很久，硬是做不出来，可看起来分明是 **128** 条码。

3、**Barcode.dll** 的 **V2.1.0.409** 新增的 **Code 128 Custom**、**GS1/EAN/UCC 128 Custom** 可编程定义 **128** 条码的输出，制作高难度 **128** 条码。识别 **128** 条码，行家从起始码字和终止码字即可识别。但 **128** 码有很多特例，比如 **128 Auto** 就是，它混同了 **A**、**B**、**C** 字符集，而 **128 Custom** 就不但混同了 **A**、**B**、**C** 字符集，而且还完全由人工控制。要生成 **128 Custom** 条码，除上面的 32 个控制字符外，还有 **11** 个功能字符：**\STARTA**、**\STARTB**、**\STARTC**、**\FNC1**、**\FNC2**、**\FNC3**、**\FNC4**、**\CODEA**、**\CODEB**、**\CODEC**、**\SHIFT**（这些功能字符的意义，请查阅相关文档）。如何用编程的形式制作 **128 Custom** 条码呢？其实这种编程很简单，比如下面一个条码文字：

\STARTAABCDE\LF\CODEC123456，意思是以字符集 **A** 开始(**\STARTA**)编码 **ABCDE** 五个字母和换行符号**\LF**，然后转入字符集 **C**（**CODEC**），再编码 **123456** 这 **6** 个数字。也就是说，**ABCDE** 和换行符号 **\LF** 使用字符集 **A** 的规则编码，而 **123456** 使用字符集 **C** 的规则编码。

由我处发行的免费软件 **FreeBarcode**，支持 **128 Auto** 和 **128 Custom** 条码的设计。