Developer and Business Tips for Using Barcode Technology in Document Imaging

Barcode technology was created in the 1940s and has really taken off in adoption ever since the 1970s. Today, it's widely used in industries like retail, transportation, healthcare, logistics, public security, automated production and much more. It's a growing market worth billions of dollars, with forecasts showing growth through at least 2019. As a result, more and more developers are implementing barcode technology into various applications. In particular, they are being tied-in with document management. So, what are some critical tips for developers to consider when embarking on this implementation?

Barcodes are widely used in document, inventory and other software management systems to automate document indexing, do batch separations, process error detection, to itemize supplies or deliveries, and much more. As a result of their usefulness, they are deployed everywhere. They show up on everything, from packages delivered all over the world to wrist straps that identify patients in a hospital. Their use increases efficiency and can significantly reduce operational costs. Barcodes can even save lives.

Barcode Evolution

Barcode technology has evolved over the decades beyond the original 1-D (linear) barcode. While still suitable for many applications, the 1-D barcode is also not suitable for many others. This is due to its limited character capacity or the limited information it can hold. It also relies heavily on a database and network. A 2-D barcode provides much higher data density. It can also encode images and multiple types of characters and has superior error correction and encryption capabilities.

As previously touched upon, the 1-D bar code can be used in a document imaging environment to implement batch separation and to do document classifying. This barcode type is also commonly used for the automatic routing, naming and indexing of documents.

On the other hand, 2-D bar codes represent the newest paradigm shift for data capture via barcodes. Compared to a 1-D bar code, which usually contains a single field, a 2-D barcode can encode complete indexing information. This makes possible full automation of document indexing without the need to access a remote database. In addition, they are popular for using in forms, for encoding of forms IDs, form layout definition and form data content. So, for example, with a computer you can then retrieve data from a survey or from different types of application forms, etc.

Barcode Creation

There are usually two ways to add barcodes. You can opt for either preprinted barcodes or self-adhesive barcode labels. Using forms with preprinted bar codes usually yields better decoding performance. This is because they are always printed in a known orientation. However, self-adhesive labels are well known for their flexibility and convenience. As a result, they are widely used in healthcare, logistics, etc.

A related consideration is their resolution. It would seem that barcodes could get away with lower resolutions when printing or storing their images. Not so. The barcode resolution matters and almost always a high resolution of 200 dots per inch (dpi) or better is recommended to ensure decoding accuracy. In many cases, 300 dpi is required – that's photo quality. Don't save on storage or printing costs at the expense of productive decoding accuracy. Once printed, make sure you don't violate the quiet zone – that's the required spacing between each barcode and other information or images. This also makes decoding perform smoothly.

Barcode Recognition

The process of barcode recognition mainly consists of two parts – barcode localization and decoding. Obviously one needs to optimize systems for maximum performance of barcode recognition. It's also important to optimize for what you will deem as typical unreadable barcodes so you can avoid decoding problems as much as possible.

Optimize for Recognition Performance

In general, it's best to pre-identify the location, orientation, symbol type, size and number of bar codes in your document. Doing so can enable you to put in place certain mechanisms for faster processing. For example, if you know you will be using a certain barcode type, pre-specify it within the code of your application. This will save time in your application process to not have to perform a check for a barcode type. It also adds assurance against incorrect barcode type auto-selections. You may even want to pre-identify the rough barcode area within your application code. Also, if you only need to decode one barcode in a document, it can speed decoding to specify this. This saves time and effort in the barcode application trying to locate where the barcode is or if there are more.

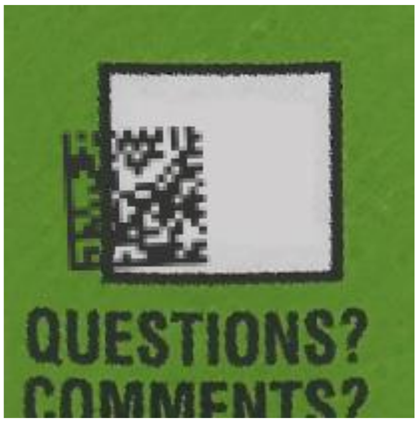
These small considerations can add up to significantly improve barcode reading performance. The workflow capabilities are also crucial. It's a good idea to allow users to verify scanned data and manually correct it in cases where errors are found. This can help improve workflow efficiencies by removing, re-reading or correcting improper barcode data.

We touched on quiet zones, which are the clear areas around a bar code. During the scanning process, quiet zones provide long periods of "computing time" in which no light to dark or dark to light transitions occur. Quiet zones physically separate the bar code from other information on the page. This makes it easier for scanners or software to differentiate bar code data from other information in the document image. For good bar code reading performance, quiet zones must be free of all irrelevant marks. The width of the required quiet zone varies by barcode symbol. In the document imaging environment, quiet zones should extend on all four sides of the symbol. Be sure to test your quiet zones, and then test it again.

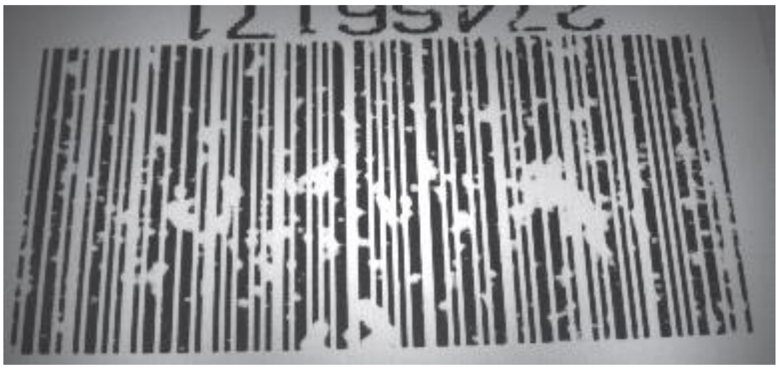
Typical Unreadable Barcodes

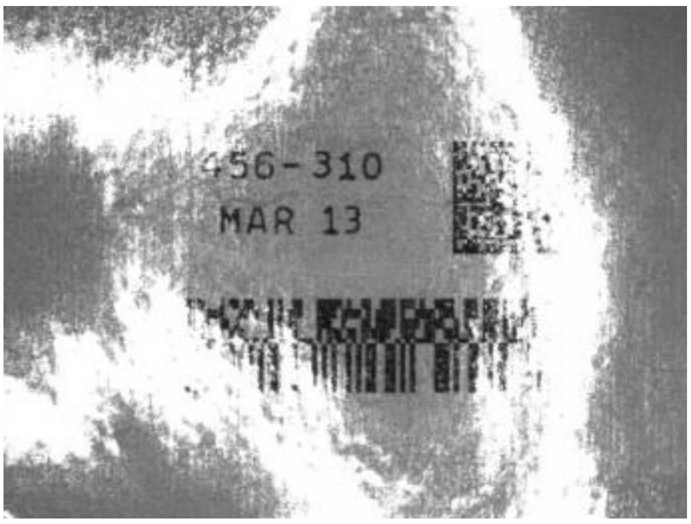
Following are images showing some common physical examples where improper barcode implementation will likely result in low or no performance.

Low contrast



Insufficient quiet zone


Damaged or Distorted



Uneven exposure

(Source: MICROSCAN)

Choosing a Barcode SDK

The correct barcode can be effectively applied in various environments: desktop, web or mobile applications. For a mobile app, users generally scan barcodes with the built-in camera. For desktop and web applications, the barcode images might come mostly from scanners or sometimes webcams. There are many details to consider when developing for different environments. A lengthy paper can be written for each one. Of course, choosing a good SDK can go a very long way in helping development and in providing enhanced capabilities.

There are some open source barcode libraries available. A popular one is ZXing, also known as "Zebra Crossing." You might instead opt for a third party barcode reader SDK. Usually at a nominal cost, commercial SDKs typically provide better barcode processing results. They also come with wider barcode type support, and their providers give better customer service and technical support. In terms of savings, they can ensure months of time saved on development which also means huge savings on development and support costs. A good barcode reader SDK can reduce work on enabling a barcode reader from four to eight months to four to eight days. The ROI on such savings can be quite dramatic.

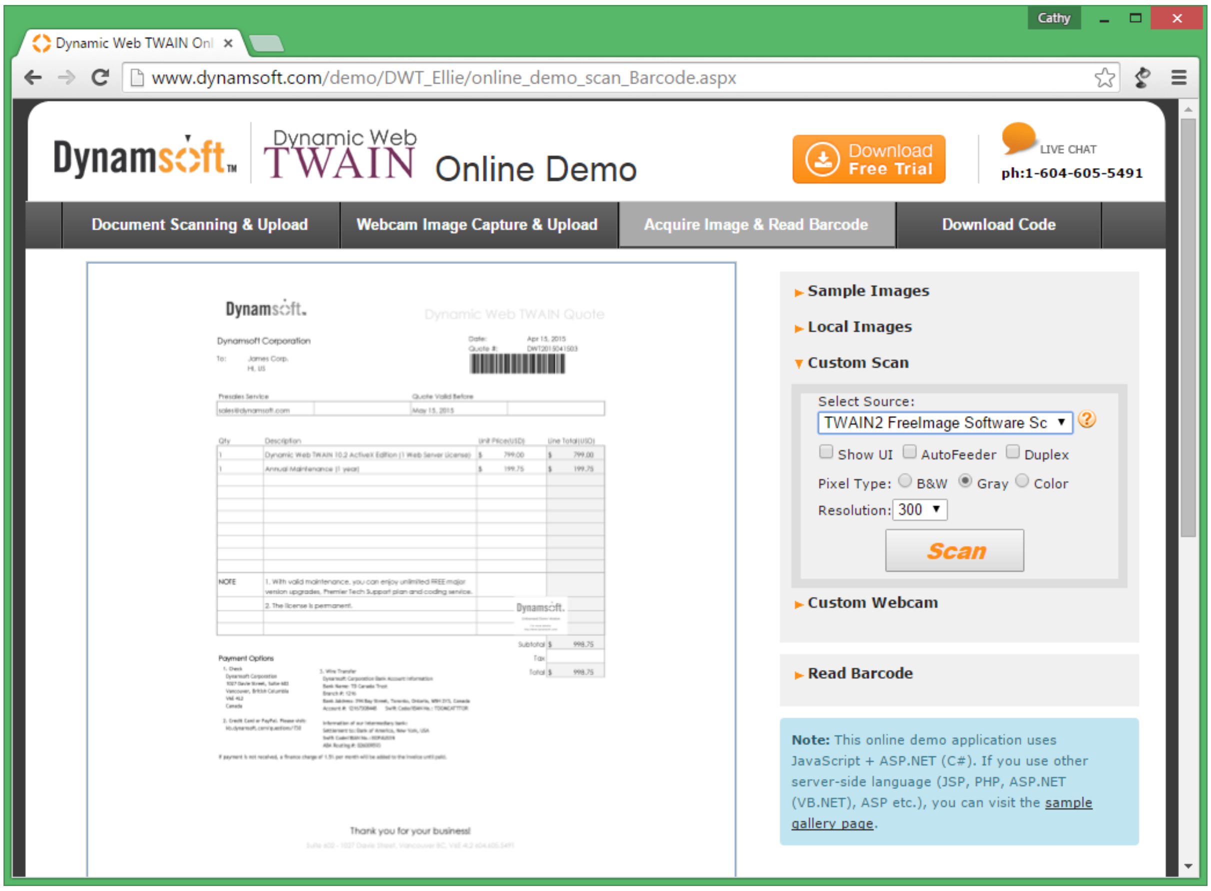
Barcode Reader SDK Capabilities

You obviously want to find a cost-effective and comprehensive barcode reader SDK. So, it's important to point out there are many critical bar code reading capabilities that must be part of any candidate solution. You need to properly balance a need on how much you need the SDK to automatically do things for you against how much you expect to be allowed to customize things for your needs:

- Automatic Location: The SDK can allow automatic finding of barcodes wherever they might be on the document
- Orientation: The SDK can decode barcodes whether they are to be read horizontally or vertically
- Skewing: Allows the decoding of barcodes in random skewed angles
- Multiple Symbols: The SDK enables reading of multiple barcodes on a page
- Checksum Handling: The SDK allows the decoding of a barcode's checksum
- Search Strategy: To speed processing, the SDK should enable developers to create barcode search strategies based on location, orientation and size
- Symbol Support: The SDK should have built-in support for all key barcode symbols that you need, such as: PDF-417, Code 93, Code 39, Code 128, CODABAR, QR Code, etc.

Example Barcode Reader SDK Code Implementation

Below is a simple example of how to use an SDK to implement scanning documents and reading barcodes in a web application. Here, we use the Dynamic Web TWAIN SDK along with the companion Barcode Reader SDK. This is to implement client-side document scanning and barcode reading. Below is a sample screenshot of an application UI:


Dynamic Web TWAIN Onl
www.dynamsoft.com/demo/DWT_Ellie/online_demo_scan_Barcode.aspx
Dynamsoft
Dynamic Web TWAIN Online Demo
Download Free Trial
LIVE CHAT
ph:1-604-605-5491
Document Scanning & Upload
Webcam Image Capture & Upload
Acquire Image & Read Barcode
Download Code
Sample Images
Local Images
Custom Scan
Scan
Custom Webcam
Read Barcode

Select Source: TWAIN2 FreeImage Software Sc

Show UI AutoFeeder Duplex

Pixel Type: B&W [✓ Gray] Color

Resolution: 300

Note: This online demo application uses JavaScript + ASP.NET (C#). If you use other server-side language (JSP, PHP, ASP.NET (VB.NET), ASP etc.), you can visit the sample gallery page.

The basic steps are as follows:

1. Include the Dynamic Web TWAIN JavaScript scanning library and also the Barcode Reader SDK in the web page code. This is provided with the SDK.

```
<script type="text/javascript"
src="Resources/dynamsoft.webtwain.initiate.js"> </script>
<script type="text/javascript"
src="Resources/dynamsoft.webtwain.config.js"> </script>
<script type="text/javascript"
src="Resources/addon/dynamsoft.webtwain.addon.barcode.js"> </script>
```

2. Also provided with the SDK, this code is implemented to scan documents that have a barcode from TWAIN scanners.

```
function AcquireImage() {
    if (DWObject) {
        DWObject.SelectSource();
        DWObject.OpenSource();
        DWObject.IfDisableSourceAfterAcquire = true; // Scanner source will
be disabled/closed automatically after the scan.
        DWObject.AcquireImage();
    }
}
```

3. Now, to enable reading of the barcode from the scanned images, this code is added. Again, the code for you would be essentially cut and paste

```
function ReadBarcode() {
    var DWObject =
Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
    if (DWObject) {
        if (DWObject.HowManyImagesInBuffer == 0) {
            alert("Please scan or load an image first.");
            return;
        }
        //Get barcode result.
        switch (document.getElementById("barcodeformat").selectedIndex) {
                case 0:
                    result = DWObject.Addon.Barcode.Read(
                        DWObject.CurrentImageIndexInBuffer,
EnumDWT_BarcodeFormat.CODE_39, GetBarcodeInfo, GetErrorInfo);
                    break;
                case 1:
                    result = DWObject.Addon.Barcode.Read(
                        DWObject.CurrentImageIndexInBuffer,
EnumDWT_BarcodeFormat.CODE_128, GetBarcodeInfo, GetErrorInfo);
                    break;
                default:
                    break;
            }
        }
}
```

Finishing Up

Any good software developer understands you've got to be sure what you created works AND that users find it helpful. A good software developer also knows that good planning before starting is the first step to ensuring this. Analysis and testing of the application is important. But, equally important to whether it works or not is whether it actually helps intended users.

So, it helps to get different departments involved during pre and post development of the solution. If it's a healthcare practice, this may mean office admins, doctors and nurses – even third-party pharmacies may be important. In a corporate enterprise perhaps it's sales, manufacturing, marketing, HR and the research department. So, start with identifying who the stakeholders are so no one's input is missed.

One basic testing point is to ensure platform compatibility. This means properly determining that the application supports all the required user platforms and devices. This might include workstations, browsers, flatbed scanners, even a webcam to be used, and more. For other hardware you obviously want to ensure barcode scanner compatibility. This means you want to ensure your barcode scanner devices are able to read the barcode images and properly pass the data to your system. Finally, you must test data transfer workflows. This includes making sure the data from barcodes can be successfully added/updated to a database.

There's a lot to know but, there's also a lot of help too, particularly if you select a good SDK vendor. Below are some additional resources to help you take the next step.

References, resources and credits/attributions:

Dynamsoft Barcode Reader online demo, sample code, and more

All My Papers, Bar Codes in Document Imaging

The Most Common Causes of Unreadable Barcodes

More about the basics of barcode technology