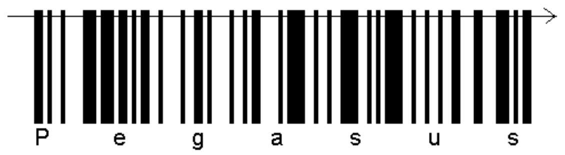
Using Barcodes in Documents — Best Practices

Barcode Basics

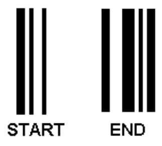
A basic understanding of barcode recognition algorithms can help you to make implementation choices that optimize your success. Basic 1-dimensional (1D) barcodes, such as Code 3 of 9, Code 128, and other codes that consist of bars of varying widths, are detected using quite different methods than the more complicated 2-dimensional (2D) barcodes such as PDF417, DataMatrix, and QR code. There are two basic steps for either: finding any barcode(s) on the page image, often called "***location***," and determining what information the barcode contains, or "***decoding***."

To locate 1D barcodes on a page, the software essentially scans the entire width, over the full height, of the document image, searching for black/white patterns of pixels that are typical of barcodes. In the example below, the more



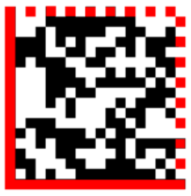
consecutive horizontal scans that turn up with a specific pattern, the more likely you are to have located vertical bars that are likely to represent a barcode. These areas containing bars in the image, called ***regions of interest***, are marked for later decoding.

The 1D decoding process consists of counting and comparing groups of image pixels in each region of interest to determine the relative width and location of the bars and the white spaces between them. Predefined patterns at the start and end of each barcode generally



identify its type. Pegasus' Barcode Xpress can identify about 30 different 1D barcode types. The pattern between the start and end identifiers can then be interpreted based on the code type specifications to decode each numeral or character that the barcode contains, along with any special check characters that some barcodes use to confirm that decoding results are valid.

Locating and decoding 2D barcodes is significantly more complex. Here, the location process entails finding "markers" that uniquely identify the barcode type. The PDF417 start and end pattern includes several vertical bars that appear much like those in a typical 1D barcode. DataMatrix codes are located by their distinct pattern of solid lines on the left and bottom edges, along with alternating filled and empty boxes (called ***modules***) along the top and right edges. These identifying features are shown in red in the illustration to the right. The inside portion of the barcode also consists of black or white modules. A QR code uses different identifying

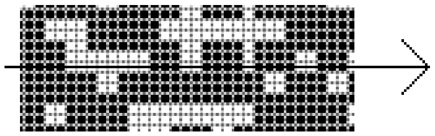


marks. It includes distinct square "targets" in three of the four corners. The "missing" fourth target also indicates the correct orientation for the QR code. These features of the 2D barcodes also outline the complete region of interest that surrounds the content encoded within.

Once the distinguishing features of a 2D barcode are found, the presence of black or white pixels in each module can be used to decode the contents of the barcode according to its specified design. The 2D barcodes can be created with varying amounts of redundant data, allowing the barcodes to be accurately decoded even when up to 30% of the modules are damaged or missing. The higher the redundancy, which results in a physically larger barcode, the more tolerant of this "noise" the decoded barcode will be.

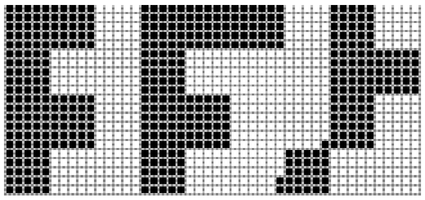
How Size Affects Accuracy

Now that we have an idea how barcodes are seen by the software, we can begin to understand how size will affect our recognition accuracy. The software will scan across (or down) a 2D barcode such as the small piece of a DataMatrix code shown here:



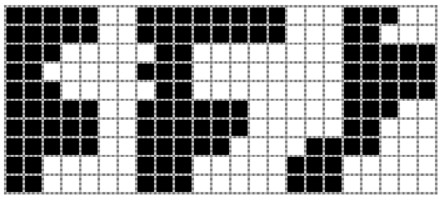
If the scan pixels were to align perfectly with the image it wouldn't take very many of them to accurately identify the contents of each module, which is within a 2 x 2 pixel box in this example.

But let's take a look at what happens when an actual image is read by a scanner, FAX machine, or multi-function printer (MFP). The DataMatrix code shown here is approximately three-quarters of an inch square. It contains the message: "**www.pegasusimaging.com Pegasus Imaging Corporation**" representing 50 characters, plus correction characters that were automatically added by the Barcode Xpress Professional + 2D Write Edition. This image was printed on a typical laser printer at 600 dpi with enhancement, then placed on a flatbed scanner and scanned at 200 dpi to a black-and-white TIFF file. Zooming in on a portion the resulting image, with grid lines inserted to show the pixel locations, we can see edges of the individual modules quite well. Each module is about 5 x 5 pixels but there is, of course, no guarantee that the pixels will always align with the grid. At 200 dpi, Barcode Xpress is able to



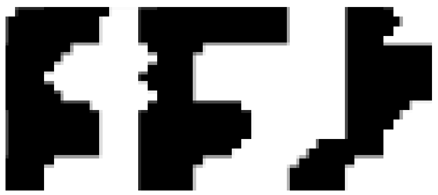
decode this barcode with 100% confidence. This is because there is enough certainty for enough of the pixels in each of the modules to reliably determine whether each module is actually filled or unfilled.

If we were to scan the exact same barcode at only 100 dpi, however, you can see that there is not enough reliable data to properly determine the state of each module. Instead of 5 x 5 pixels



per module, now we only have an average of 2.5 x 2.5 pixels to consider. When pixels are added or dropped at the edge of each module, a necessary result of the fact that the scanner is not lock-synchronized to the modules, many of them can no longer be discerned. Even though there are error-correction characters included in the barcode, most likely not enough of ***those*** can be identified to help us, either. As a result, the barcode can no longer be decoded.

Even with the limited amount of information available, it's actually still possible to make modifications to this image after scanning to recover the data contained in this barcode. By applying a ScanFix operation called ***Smooth-Zoom***, the barcode becomes readable again.



SmoothZoom doubles the resolution of the barcode, but does so intelligently. For example, rather than taking one black pixel and simply making it into four black pixels, each new pixel is filled in based on the pixels surrounding it. The smoothed image, although it looks little like the one that was scanned at 200 dpi, becomes readable again. While this operation has improved this relatively clean image enough to be decoded, it would have been preferable to scan the image at 200 dpi, so that there would be enough clean pixels available to handle other problems that might occur along the way.

Choosing Your Barcode Size

The key factor in barcode decoding is the number of pixels per line (1D) or per module (2D) that are available to be evaluated in the final image. Any barcode can be printed in almost any size. A very large barcode could be easily decoded even if it was scanned at a very low resolution. The trick is to choose a barcode that doesn't take up the whole page, yet still does not need to be scanned at a high resolution that will rapidly consume storage space. Each time you double the scanning resolution, the resulting file typically ***quadruples*** in size. Unnecessary overkill can thus result in a great waste of image storage and transmission resources.

To allow for any kind of averaging, at least ***three pixels*** are needed for each minimum-width bar (and each minimum gap) in a scanned 1D barcode. Barcode Xpress imposes limits when writing barcodes, to prevent generation of barcodes that are unlikely to be readable under even the best conditions. You can gain a rough idea of the minimum required width in inches for a typical 1D barcode from the following table. Note that these sizes actually vary for every individual 1D barcode type, with the following widths being rough averages.

Characters in typical 1D code	**Scan resolution (dpi)**	**Minimum Width (inches)**
5	100	2.5
	200	1.25
	300	.83
10	100	4.0
	200	2.0
	300	1.3
15	100	6.0
	200	3.0
	300	2.0

The sample 50-character DataMatrix barcode we looked at above is about ¾-inch square, and contains 26 modules across by 26 modules down. That is 32.5 modules per inch, or about 6 pixels per module at 200 dpi. Since finding modules within a field is a more difficult than locating vertical bars, a minimum of 4 or 5 pixels per module is the recommended size for 2D barcodes. Using this guideline, calculations can be performed for a barcode containing any number of modules. Some sample minimum DataMatrix barcodes sizes for 5 pixels per module, and PDF417 barcode **s**izes for 4 pixels per module, are shown below.

Note that the use of the word ***columns*** in PDF417 is different than it is for DataMatrix. A PDF417 column is actually a group of 17 horizontal modules, each containing the specified number of rows of modules. The height of each individual module in a properly-constructed PDF417 barcode should be 3 to 5 times its width. PDF417 is also not a square 2D barcode, so that the width shown is based on default settings for the number of columns required. The PDF417 sizes include the default error characters (two full columns) and the width of the start and end bars.

DataMatrix matrix size (rows x cols)	**Scan resolution** (dpi)	**Minimum Height and Width** (inches)
16 x 16 (about 16 characters)	100	.80
	200	.40
	300	.27
16 x 48 (about 70 characters)	100	0.8h x 2.4w
	200	0.4h x 1.2w
	300	.27h x .72w
32 x 32 (about 90 characters)	100	1.6
	200	.80
	300	.53
64 x 64 (about 400 characters)	100	3.2
	200	1.6
	300	1.1

PDF417 matrix size (rows x cols)	**Scan resolution** (dpi)	**Minimum Width** (inches)
5 x 3 (7 to 10 characters)	100	4.8
	200	2.4
	300	1.6
10 x 5 (about 50 characters)	100	6.2
	200	3.1
	300	2.1
14 x 8 (about 150 characters	100	8.2
	200	4.1
	300	2.7
20 x 11 (about 300 characters	100	10.3
	200	5.2
	300	3.5

Some quick calculations will also reveal that either a QR code or DataMatrix barcode typically requires only about ***one fourth of the total area*** as a PDF417 barcode containing the same amount of data. DataMatrix bar codes can also be created in a rectangular format (see 16 x 48 size above) as well as the more common square form, which can be more convenient for positioning on a form.

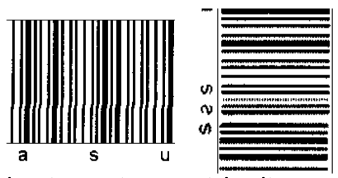
Just the FAX, Ma'am

While the above guidelines will help you to optimize your success with scanners and MFPs, FAXes can be quite a different animal. The ***standard*** mode for FAX transmission, which was designed to transmit page-sized images as quickly as possible over once-expensive telephone lines, scans documents at 203 dpi horizontally, but only 98 dpi vertically. This means that, while the resulting image may contain adequate resolution horizontally, the vertical resolution may be only half as good. As illustrated above, a resolution of around 100 dpi is barely adequate for many barcodes. This loss in vertical resolution does not occur when the sender chooses ***fine*** resolution, which specifies 203 x 196 dpi vertical resolution, or ***superfine*** resolution, for which the vertical dimension is actually scanned at double the horizontal resolution (typically 391 x 203 dpi).

Note that the appearance of a standard-mode FAX when it is displayed will not reveal the lost resolution. It won't appear "squashed" because the vertical pixels are normally doubled by any FAX viewing product, or the ImagXpress® component. But detail within your barcode may still be lost. Since this loss is in the vertical direction only, you may wish to use only 1D barcodes in FAXed documents. The loss of vertical resolution will have little effect on the vertical bars unless the document is significantly skewed. PDF417 barcodes can also be created with higher module dimensions, which will reduce their sensitivity to low vertical resolution. You can also ***resample*** the image using ImagXpress to double the vertical resolution while keeping the horizontal resolution the same. This will restore the number of vertical pixels available, but does not replace any information that was dropped by scanning the image at half the resolution. In conclusion, if you require advanced features of 2D barcodes such as DataMatrix or QR code, be certain that your testing includes FAXes sent in both fine and standard resolution modes.

Where on the Page?

If you are designing the form, you may have a choice as to how you can place a barcode on the page. Since sheet-fed scanners are most likely to pull the pages through from the short end, you should orient any 1D barcodes horizontally. That way, if there is a small amount of slippage, instead of the bars being moved together or apart, or repeated, they will be stretched or compressed, which may not affect readability. In the sample shown below, a section of the same 3 of 9 barcode was placed on a page in both ho-



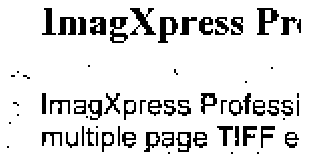
rizontal and vertical orientations, with slippage created during scanning. The horizontal barcode (vertical bars) was readable with 100% confidence, while the vertical barcode caused recognition errors because a few of the bars were actually scanned twice when the document slipped. This is also an excellent argument for using a barcode type that includes at least one checksum character, such as Code 93 or Code 128. Then you are much more likely to know that the barcode has been damaged.

As far as location on the page is concerned, there are a few things to consider. First, you don't want to position the barcode where it is likely to be damaged with a staple or punch hole. You also need to be certain to establish a ***quiet zone***, a clear area without any text or other marks, around every barcode. This ensures that the recognition software can tell where the edges of the barcode are. The required quiet zone varies by barcode, but more is generally safer. Providing 1/4-inch on all sides would be ideal, but at least 1/8-inch of empty space is essential for most barcodes.

With Barcode Xpress you can specify the maximum number of barcodes to be located on an image. If you reach that maximum before searching the entire page, for example, you can stop searching the page. Since barcode scanning proceeds from the top of the page to the bottom, you can potentially speed up the location phase by placing all barcodes toward the top edge of the page.

Using Cleanup to Improve Recognition Accuracy

Sophisticated image cleanup software, called ScanFix®, is included with Barcode Xpress. The ScanFix toolkit includes a large number of functions that can be used to reduce the effects of damage that often occur to document images en route, through operations such as copying, FAXing, etc. For example, a small portion of a document containing many "specks" is shown here:



ImagXpress Pr

ImagXpress Professi
multiple page TIFF e

It contains numerous small dots, typically caused by dust on the scanning platform. They can cause significant errors during both OCR and barcode recognition functions. ScanFix includes a ***despeckle*** function that can effectively remove these unwanted specks from most documents.

A different problem can occur when a scanner, copier, or FAX machine scans a barcode whose lines are not dark enough, as it converts it to black and white. This "binarizing" function is often done during scanning, as it greatly reduces the size of the resulting files. The resulting barcodes will then have ***white*** specks on the, as shown here: Although the despeckle function does not directly remove white specks, the

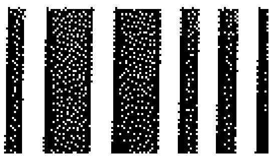
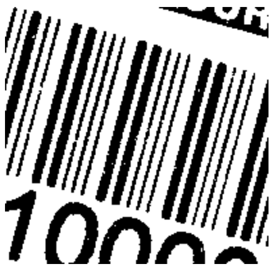


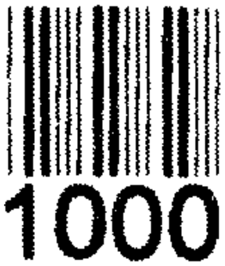
image can be negated (reversing the black and white pixels), then despeckled, and negated again to return the original image without specks. This barcode went from unreadable to 100% confidence after that simple series of cleanup steps.

The SmoothZoom function described above is also often very helpful in converting an unreadable barcode into a one that is reliable. Because both ScanFix and Barcode Xpress are so fast, it is often practical to perform a variety of cleanup sequences, checking for the presence of barcodes and their reported confidence after each attempt. This may be especially helpful when documents arrive from a wide variety of sources, with various types of damage along the way.

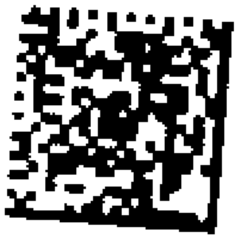
Another ScanFix function that can be very useful is ***dilation***. Let's say that you know that you are expecting horizontal barcodes with vertical bars that might have irregular white scratches across them. The dilate function simply finds all black pixels in the images and converts the adjoining white pixels to black pixels also. The direction of the adjoining pixels, and the number of pixels to extend the dilation, are fully selectable. Thus, by dilating 2 pixels up or down, a white scratch could be completely removed from a barcode with vertical bars, with little or no damage to the locations of the bars themselves. Similarly, if you had (or expected) black lines across your barcodes, you could ***erode*** the image in the vertical direction, reducing the height of any stray black lines. This will have very little effect on the longer black lines that form the barcode. Note, however, that this method will usually not help with 2D barcodes that are sensitive to pixels changes in both vertical and horizontal directions. Also, dilation is much more likely to be successful if the barcodes have been accurately deskewed. Otherwise, the white spaces between the bars may be reduced, adversely affecting decoding.

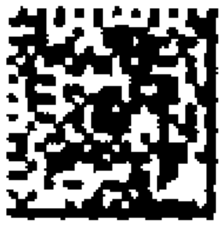
The deskew function ***can*** be very helpful, as when it is used with dilation above. However, it cannot always be assumed that deskew will improve recognition. The sample Code 39 barcode to the right was scanned with a 15 degree skew angle. Although there is a stair-step effect along the diagonal lines, the width of the bars and spaces appears quite uniform across the image. Even with this relatively high skew angle, it was recognized with 100% confidence. But notice what happens when we deskew this image. Because each pixel had to be forced into a new orientation, the previously stair-stepped bars now appear to be much more jagged, and the spaces between them are much more variable. This image is still decoded, but the confidence level has dropped to 97%. Clearly, deskew can cause more problems than it solves. When the entire page on which a barcode appears is deskewed, there is also no guarantee that the barcode itself will end up deskewed, unless it was perfectly positioned and scanned on the page.





Damage to 2D barcodes can be even more severe, since we now rely on both horizontal and vertical fidelity. The DataMatrix code here is recognized at 100% confidence. After deskewing it at about 4 degrees you can see that there has been significant damage to many of the modules, and the resulting image has become completely unde-





codable. In this particular example, the image could be repaired by a SmoothZoom operation and was still decoded at 100% confidence. But there is no guarantee that this will always be the case.

There are two things to learn about image cleanup from these examples. First, allow the decoding software to do its best at finding the information in a skewed barcode, whether 1D or 2D. Second, if you are unable to predict the kinds of damage that may occur in your workflow, it's ideal if you can selectively apply several different cleanup sequences, retaining the original image each time so that you can go back to it, while checking the confidence level of the recognized barcodes after each cleanup option is tried.

When applying cleanup operations, it can pay to find opportunities to "learn" from the document. For example, if a page deskew shows that it was rotated more than 2 degrees, you may want to automatically perform a SmoothZoom function after deskew, whereas you might otherwise prefer to undo the deskew operation before barcode recognition, and perform it again afterward for visual improvement. Certainly, a check of the image resolution could be used to direct you through two or more different image processing paths. Or if a despeckle operation reveals that hundreds of specks were found, you might send that image through a wider variety of cleanup options than a known-clean document. The presence of one particular barcode or its content may be also used to trigger different kinds of follow-up processing. The possibilities are endless, limited only by the return on investment resulting from fewer errors and a reduction in costly manual intervention.

Where Should I Binarize?

Except in the case of FAX, you may have a choice over whether you scan your documents in color or grayscale, or obtain them in black and white only. The reason that the binarization process results in such a significant reduction in file size is precisely because a lot of information is discarded along the way. Once the file is scanned or saved without this data, it can never be recovered. So if your overall business process and your available storage facilities allow it, you will have a lot more options to try during binarization if you can retain a copy of the documents in color or grayscale. This may allow you to try various cleanup sequences, as outlined above, during the barcode recognition stage.

The file sizes required to keep documents in grayscale are For example, the first page of this document alone, scanned at 300 dpi in 8 bit-per-pixel grayscale, consumes about 3.1 megabytes of storage in TIFF format with LZW compression. When the same page is binarized and stored with Group 4 compression, the size is about 62 kilobytes. This is a reduction of about 50-to-1, a very significant factor even with today's inexpensive storage options. As a result, unless the business process allows for grayscale images to be used and then discarded early on, it probably is cost-prohibitive.

Optimizing Recognition Speed

There are two separate opportunities for speed optimization. One method involves applying the multi-threading capabilities of Barcode Xpress and other controls. The other primarily consists of reducing the number of activities performed based upon the predictability of your arriving images.

The rapid spread of multi-core processors has greatly increased interest in multi-threading as an optimization strategy. The optimal application of multi-threading will require experimentation with your particular workflow, based on which operations depend upon others, and must therefore be serialized, and which can be performed in parallel. You may, for example, be able to process two or more separate batches of images simultaneously.

The 1D, 2D, patch codes, and 4-state postal groups are processed differently and separately. To avoid having to process every page with all four types of recognition, it is helpful to know which types of barcodes are expected. Within the standard 1D barcodes types, however, there is no significant speed improvement gained by limiting recognition to the expected barcode types, except that you can avoid the work of discarding any results of barcode types that may turn up that are not of interest to your application.

Many parameters of Barcode Xpress can be tuned to speed up overall recognition. Which of these may be applied will depend on whether or not your application can predict the types of images that it will receive to process. For example, if you know that all of your 1D barcodes will be horizontal on the page, you can set Barcode Xpress ***not*** to scan vertically or at 45-degree angles for barcodes. In one sample image containing seven horizontal barcodes, for example, this reduced total processing time from about 35 milliseconds to 14 msec…a 60% time savings. These speed changes can be very significant if you are processing many images. You could also use horizontal-only scanning to ignore a vertical barcode on the same page, if desired.

If the maximum number of 1D barcodes searched on that same page is set to 1 instead of 100, the first barcode is recognized in under 3 msec. A sample page image with only one barcode near the bottom of the page shaved recognition time from about 35 msec to 9 msec, just by changing the maximum number of barcodes searched from 2 to 1. Further, moving the one barcode from the bottom of the page to the top of the page reduced the recognition time from 9 msec to about 4 msec. This also brings an important fact to light—searching a page image for a barcode when none is present always means searching the ***entire*** page. That 35 msec time is indicative of the time that will be consumed for every non-barcoded page that is searched for a barcode. If you know, for example, that only page one of each multi-page document can possibly include a barcode, you can eliminate time wasted searching every page of every document.

If you know the approximate location where a barcode can be expected on a page image, you can also greatly increase processing speed by cropping and examining only that particular region of interest. For example, if you know that a barcode is always printed in the top, right 1-inch by 3-inch corner of a specific document, you can search only in that area. If you're concerned that the document may be inverted, you should also search the same area in the bottom, left corner.

Another tunable parameter for 1D barcodes is ***scan distance***. There is a tradeoff between the separation of the lines scanned across the image, and the speed and accuracy of recognition. If every pixel is scanned, not only does performance suffer, but there can also be "false positives" as even letters of text could have enough similarity to barcodes to be tagged as such. The default scan distance of 5 pixels is usually nearly optimal for most image densities. But increasing that scan distance to 10 pixels reduces the processing time to recognize the seven barcodes in the sample above with horizontal-only scanning from 14 milliseconds to about 9 milliseconds – a 40% reduction – with no loss in accuracy. The barcodes in this example averaged 70 to 80 pixels high on an 8 ½ x 11-inch page scanned at 200 dpi.

Recommendations

1. **Pixels per element** – For best readability of 1D barcodes, create at least three pixels for the minimum bar width. For 2D barcodes, you should create at least 4 or 5 pixels for each module.

2. **Variable length data** – Do not allow barcodes to be automatically generated with unlimited length data. You will not be able to guarantee the overall barcode size required for accurate recognition if more and more elements are squeezed into a fixed area. If you require variable length data, be certain to set a limit to the maximum size of each field, and test the resulting barcodes under the expected worst case conditions for damaged images.

3. **Avoid skewed input** – In most cases, higher skew angles reduce barcode recognition accuracy. Whenever possible, take steps to reduce skew. This may involve routine maintenance, using better sheet-fed scanners, deskewing barcodes as separate regions of interest on a form, or simply placing a box on your forms to help operators to align manual barcode stickers. But, once you are stuck with skewed images, remember that electronic deskew does not always improve recognition accuracy.

4. **Optimize cleanup** – Experiment to determine the optimal cleanup features for the types of damage your documents usually experience. This could be SmoothZoom, dilation, deskewing the barcode's region of interest, or an algorithm that tries each and compares decode confidence.

5. **Optimize performance** – If you are processing a very large number of barcodes, you may be able to significantly increase performance by limiting scanning to either horizontal or vertical (if only one orientation is expected), limiting the number of expected barcodes per image, utilizing the multi-threading capabilities of Barcode Xpress, searching only those pages or areas on each page where barcodes are expected, and other methods.

6. **Know thy image** – The more you can predict about the images containing barcodes, the better you can optimize their recognition. If you know the number, type, resolution, rotation, and placement of barcodes, you can speed recognition. This may also suggest locking down your business processes to reduce overall variation. Using cues about the image can allow you to optimize cleanup and improve recognition speed and accuracy.

7. **Test, test, test** – Try to simulate the expected business processes that document images will experience on their way to your system. If documents will be FAXed twice, FAX some samples three times and then validate. If they will be printed, copied, and scanned, at least repeat all of the expected steps on the worst acceptable hardware in your system.

Paul B. Firth
Pegasus Imaging Corp.
Product Manager
pfirth@jpg.com