REPORT 3

BARACK OBAMA: LONG FORM BIRTH CERTIFICATE:

Bitmap Layer and Color Attributes

by Mara Zebest

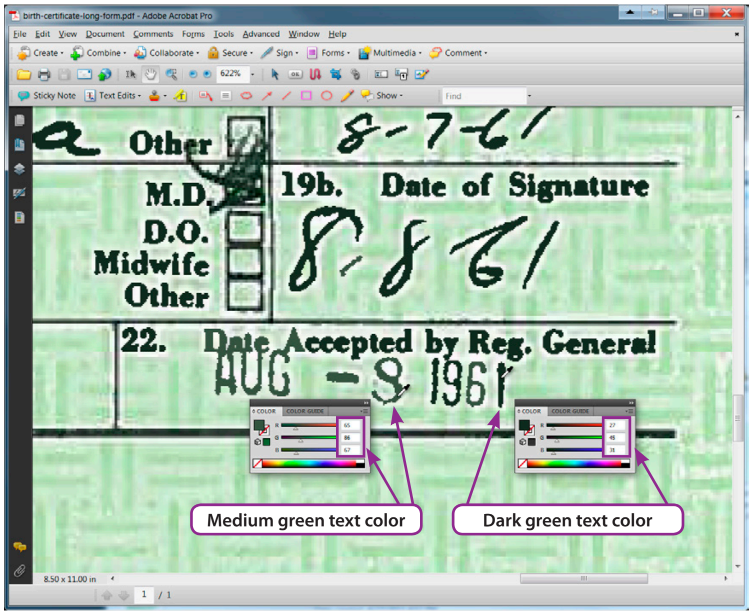


Figure 1: Different color values in text

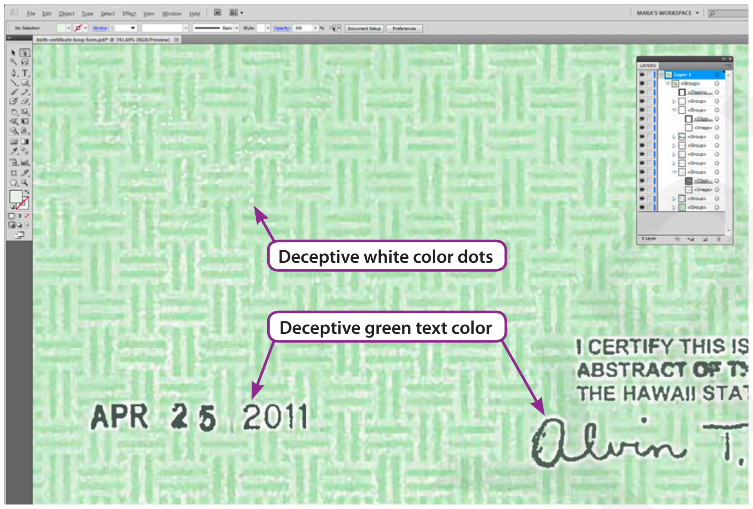


Figure 2: The deceptive color values in Obama's PDF

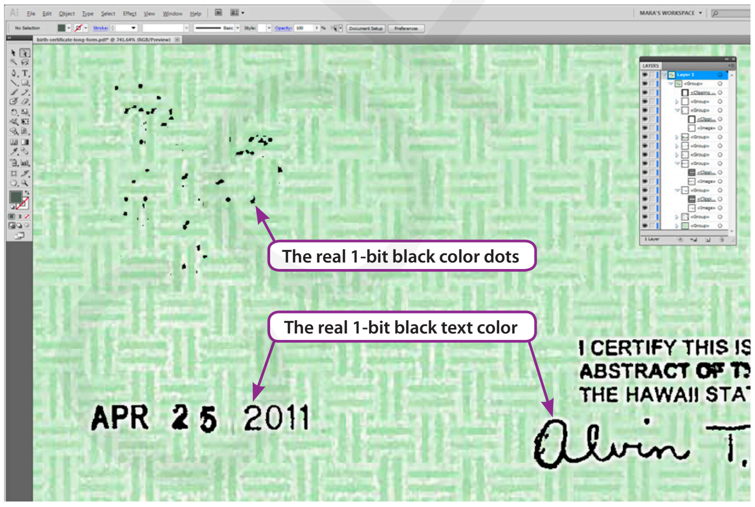

The real 1-bit black color dots
The real 1-bit black text color
APR 25 2011
I CERTIFY THIS IS
ABSTRACT OF T
THE HAWAII STA

Figure 3: Actual 1-bit underlying black color values

Introduction

I, Mara Zebest, am preparing this report as an update to previous report material regarding newly discovered attributes of Obama's PDF long form birth certificate file.

In previous reports, the original observation of color value differences between dates and text were correct, but initial theories offered to account for the color differences need to be revisited in light of further examination.

Figure 1 displays an example of one instance of color inconsistencies within the digital file. Notice that both text color values displayed in **Figure 1** are green color values—not black. Both *Color* panel values are provided in the Figure to demonstrate a pattern in which the *Red* and *Blue* values are similar (a few digits apart), while the *Green* value is stronger—thus the greenish tint to all the colors. At first glance, no true black appears to exist in Obama's PDF file. But nothing could be further from the truth.

Figure 2 offers a close-up of three layer examples in which one layer contains what appears as off-white color dots, and two other layers contain slightly varying greenish-gray colors for the date and registrar text.

Figure 3 offers the same close-up revealing the true color attributes of the layer content. This hidden black color proves the point: *The best hiding place is in plain sight*.

What does all this mean?

It will soon become clear that even color values in the PDF file are ***deceptive***. The report will explore how the colors are illusory and also make it apparent that this level of misrepresentation does not occur through normal document processes—it occurs through image manipulation.

The overall goal of this report will be to evaluate and explain the following concepts:

- Briefly define the two most commonly used color models: RGB and CMYK
- How to recognize pure black, white, and grayscale color values by evaluating the numbers in either color model (RGB or CMYK)
- How to recognize **1-bit** (ImageMask true) layer attributes—meaning layers that contain *one color only—**black***
- How to recognize **8-bit** (ColorSpace) layer attributes—meaning layers capable of containing more than one color
- Optimized layers briefly revisited and evaluated to determine *if* and *when* optimization will produce layers along with the patterns of optimized layers v manufactured layers
- Compare **1-bit v 8-bit** layer patterns in an automated process (such as optimization) against the pattern displayed in Obama's PDF file
- Examine how multiple **1-bit** layers (black color only) can be manufactured within Obama's PDF file—and how a black color value (on a 1-bit layer) is represented as a color *other than black*
- Define PDF printing settings from Illustrator and Preview in obtaining a low file size and attributes seen in Obama's PDF metadata and object code
- Rule out MRC compression as a factor in explaining layers

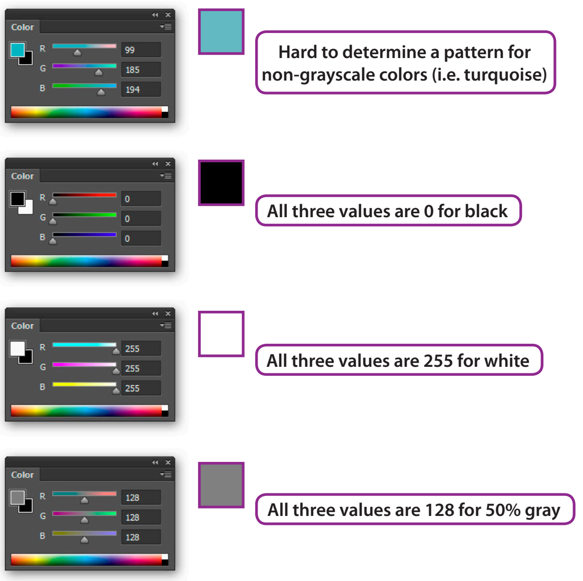

Color
R 99
G 185
B 194
Hard to determine a pattern for non-grayscale colors (i.e. turquoise)
R 0
G 0
B 0
All three values are 0 for black
R 255
G 255
B 255
All three values are 255 for white
R 128
G 128
B 128
All three values are 128 for 50% gray

Figure 4: RGB grayscale color value patterns v non-grayscale

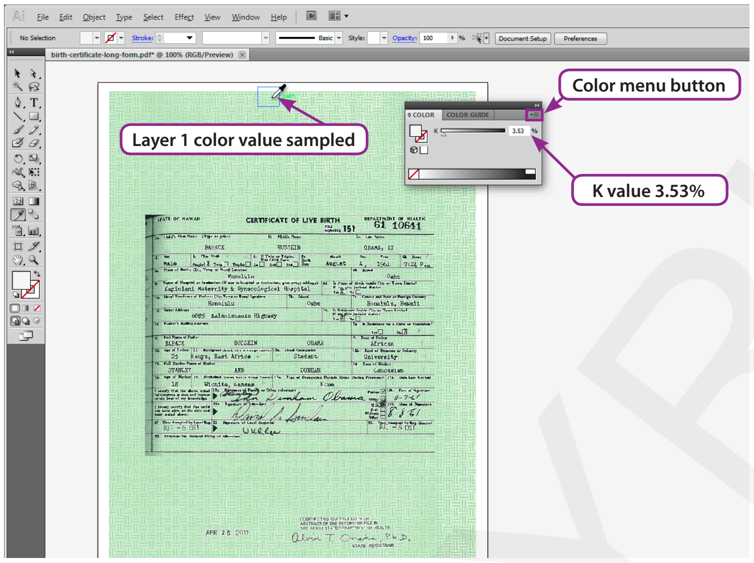

File Edit Object Type Select Effect View Window Help
birth-certificate-long-form.pdf* @ 100% (RGB/Preview)
COLOR COLOR GUIDE
K 3.53 %
Color menu button
Layer 1 color value sampled
K value 3.53%

Figure 5: A K value signals a pure grayscale color value

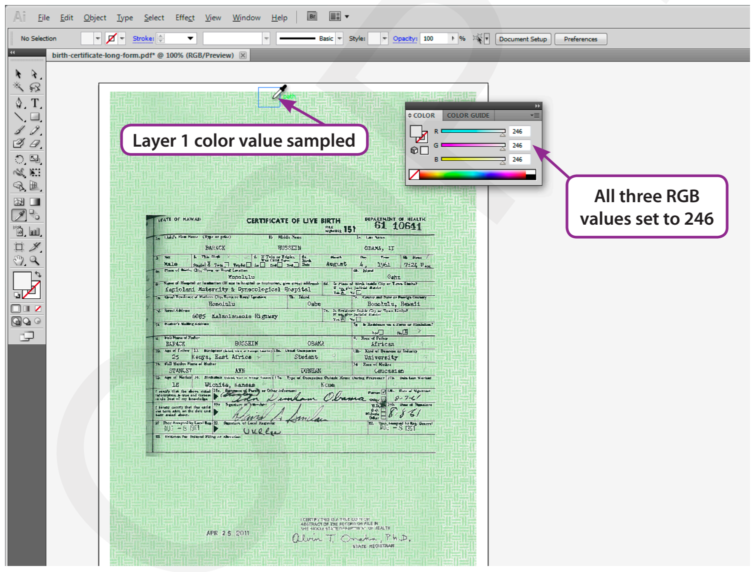

File Edit Object Type Select Effect View Window Help
birth-certificate-long-form.pdf* @ 100% (RGB/Preview)
COLOR COLOR GUIDE
R 246
G 246
B 246
Layer 1 color value sampled
All three RGB values set to 246

Figure 6: The RGB off-white color value numbers

Color Values by the Numbers

When evaluating color within a digital file, it helps to understand how to evaluate by the numbers. In order to understand what the numbers mean, a brief explanation of the two most commonly used color models might help at this point:

- **RGB:** Is an additive *device-dependent* color model typically used by digital cameras, scanners, computer monitors, and other image devices. **R**=Red, **G**=Green, and **B**=Blue. Each of these components will contain a numeric value between **0** and **255**. The RGB color model is preferred for image files viewed on a monitor, such as website images or photos shared via e-mail.
- **CMYK:** Is a subtractive color model typically utilized in the professional printing industry. The CMYK color model measures color value using the percentage of each of the four ink colors to produce the desired composite color: **C** = Cyan, **M**=Magenta, **Y**=Yellow, and **K** represents the percentage of black.

Pure Grayscale Patterns

Chances are slim that the average person would know the numeric combination for creating most color values. For example, if asked the numeric values for a turquoise color (in either color model; RGB or CMYK)—it is rare that those numbers are easily retrieved without looking at a color panel. However, grayscale colors are an exception to that rule and easier to determine on the fly. There's a very simple pattern for either color model, when evaluating white, black, or any grayscale value in between.

Figure 4 demonstrates a turquoise (non-grayscale color) compared to the pattern for RGB grayscale values in which all three values will be the same (unlike the non-grayscale colors). *Black* is represented by **0** (the lowest value) for all three components. *White* will display **255** (the maximum value) for all three components, and grayscale is any value in between (for all three components). Half of *255* can be rounded up to *128*, thus *50% gray* can be represented by using **128** for all three values (R, G, and B).

In *Illustrator*, the *Color* panel will typically default to displaying a sampled grayscale color as a **K** percentage component (from the CMYK model). White is represented by 0% (or the absence of any black). Conversely, 100% would be black and any percentage in between is a grayscale value.

The next two Figures will sample the color value found on Layer 1 from Obama's PDF file since this is one of the rare examples in which a pure grayscale value can be found quickly within Obama's PDF file.

When the color on Layer 1 is first sampled, *Illustrator* will default to display the result as a **K** component—in this case, a percentage value of **3.53%** (as shown in **Figure 5**).

To see the RGB equivalent, click the *Color panel menu button* seen in **Figure 5** to change color models. Click **Show Options** if available, and then click the menu button again and choose the **RGB** option from the panel menu. The 3.53% **K** value is converted to the RGB equivalent of R=246, G=246, B=246 (as seen in **Figure 6**)—which is consistent with the pure RGB grayscale pattern described. Additionally, the 246 value is closer to the maximum numeric value range of 255 (used for white), thus the 246 value is numerically consistent for an *off-white* color.

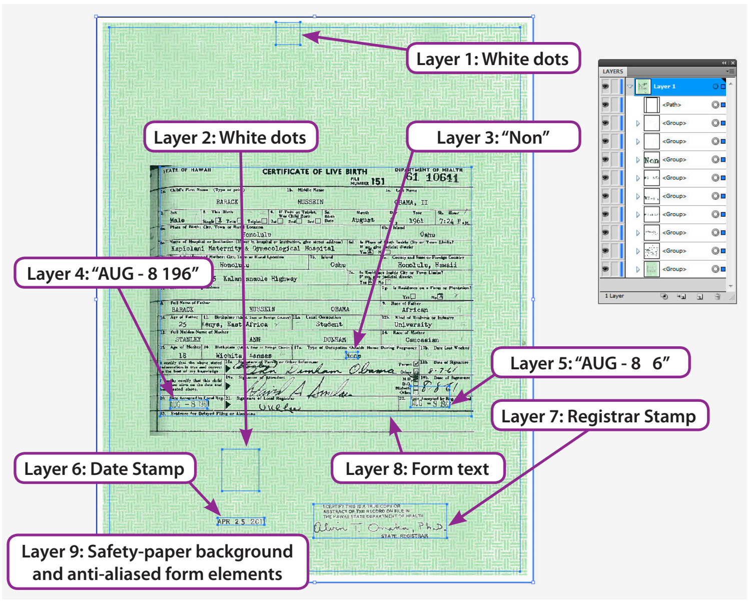


Figure 7: Layer order and layer content

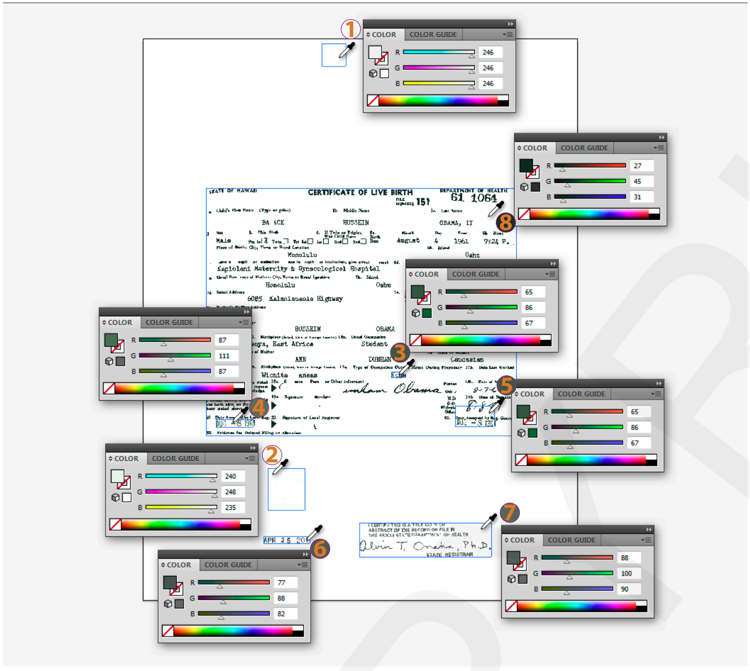


Figure 8: First eight (1-bit) layers displays only one color value each

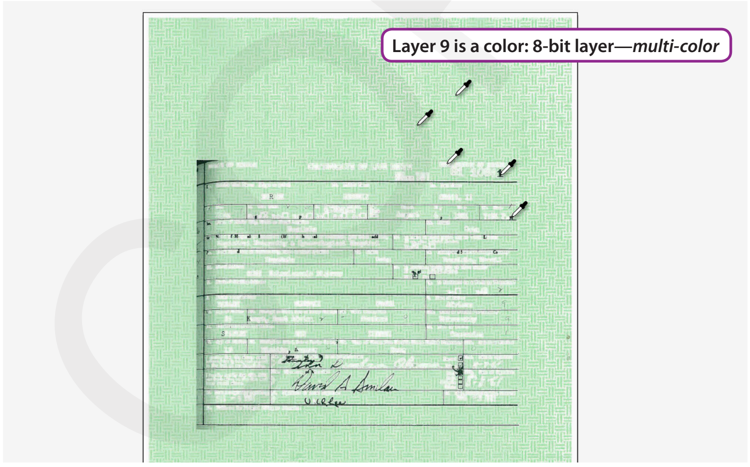


Figure 9: Multiple color values will display on last (8-bit) layer

A Simple Eyedropper Test

Figure 7 revisits the positioning of each layer number and the layer content. Armed with a better insight into recognizing grayscale color value patterns (from the *Color* panel), it becomes painfully obvious (upon opening Obama's PDF), that pure grayscale values are hard to find within the file.

There's another odd behavior that can be observed when sampling color values. It has to do with a **bitmap** (or **1-bit**) attribute found in the first eight layers. This may be a good time to define the two types of layer characteristics contained and verified by object code within Obama's PDF file—**bitmap (1-bit) v color (8-bit)**:

- A **bitmap** (or **1-bit** *ImageMask true*) layer is simply a layer that contains only *one color*—***black***
- A **color** (or **8-bit** *ColorSpace*) layer is a layer that can contain multiple color values (more than one color)

By definition, an *ImageMask true* **1-bit** layer contains a single color of ***black***. When examining the 1-bit layer color values in Obama's PDF, the results do not represent a black color value. The first off-white dot layer contains a grayscale color value, but it is closer to white (not black). The 1-bit layers in Obama's file do adhere to the one color rule but defy the rule of displaying a black color value.

The mystery deepens with the fact that there are eight layers with a 1-bit attribute. A file that has layers via optimization will only have a ***single*** *1-bit layer—not* ***eight*** (optimization will be examined more closely in the next section of this report).

Try this experiment to quickly determine if a layer is a true 1-bit or 8-bit layer: Open the Obama PDF file in Illustrator and choose the *Eyedropper* tool. To perform the experiment, make sure nothing is selected; otherwise the *Eyedropper* tool will inadvertently change the color of a selected object during the experiment.

If nothing is selected, Illustrator will still display a visual clue to the object's boundary box location whenever the *Eyedropper* tool is hovered over the object—these are the blue outlines surrounding each layer object as seen in **Figures 7** and **8**. If this feature is not turned on, go to the **Edit** menu > **Preferences** > **Smart Guides** and make sure the option is checked for **Object Highlighting**.

With no objects selected, hover over any of the first eight layer objects with the *Eyedropper* tool, and click anywhere inside the layer object's boundary box. **Figure 8** shows the location of each object and the color values that will display for each of the first eight layers. For the purpose of easy viewing, **Figure 8** also has the safety-paper layer turned off—but it is preferable to leave all the layers on when performing this test. If the layer is a 1-bit layer, the same color—one color value assigned to the layer—will display on the *Color* panel ***regardless*** of where the *Eyedropper* is clicked within that same boundary box. This behavior only occurs for the top eight layers. Even when the *Eyedropper* tool is clicked over a color that is clearly different from another color within the object box area—the same assigned color value will consistently display in the *Color* panel—this is expected behavior of layers with 1-bit (ImageMask true) properties.

Now contrast the behavior for the last layer (safety-paper layer). Click anywhere—away from the 1-bit objects (or turn them off as seen in **Figure 9**). A variety of colors will display depending where the *Eyedropper* tool is clicked—this is the expected behavior of an 8-bit color layer.

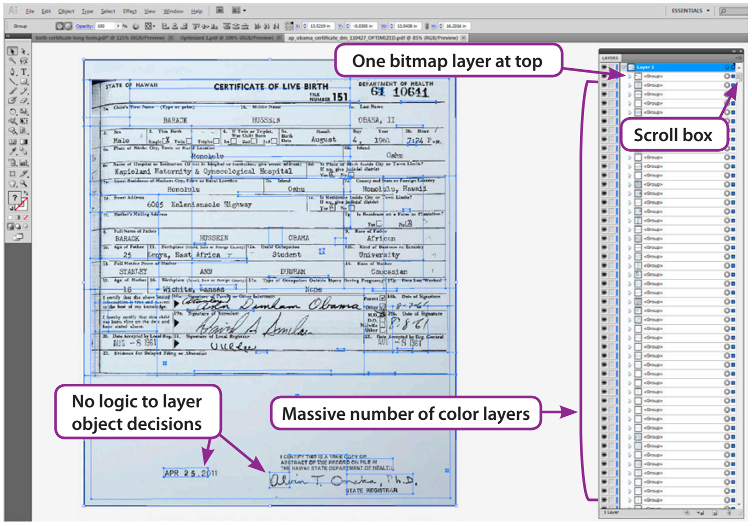

One bitmap layer at top
Scroll box
No logic to layer object decisions
Massive number of color layers

Figure 10: Optimization-produced layers applied to AP file

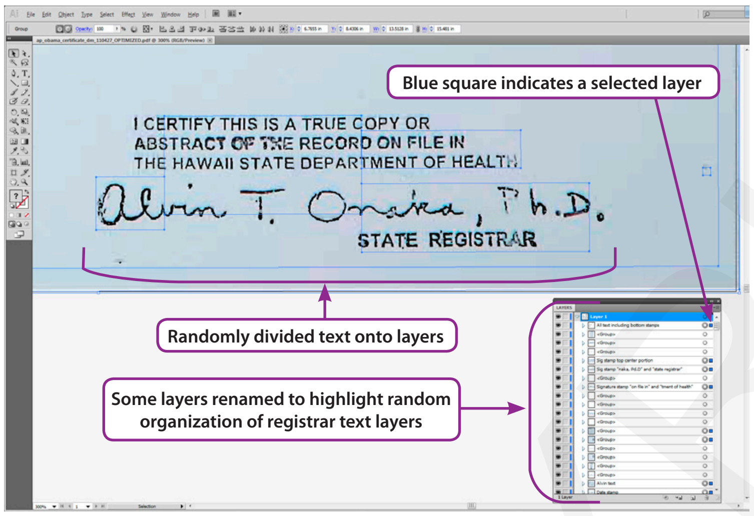

Blue square indicates a selected layer
I CERTIFY THIS IS A TRUE COPY OR ABSTRACT OF THE RECORD ON FILE IN THE HAWAII STATE DEPARTMENT OF HEALTH
STATE REGISTRAR
Randomly divided text onto layers
Some layers renamed to highlight random organization of registrar text layers

Figure 11: Optimized layers randomly divide image information

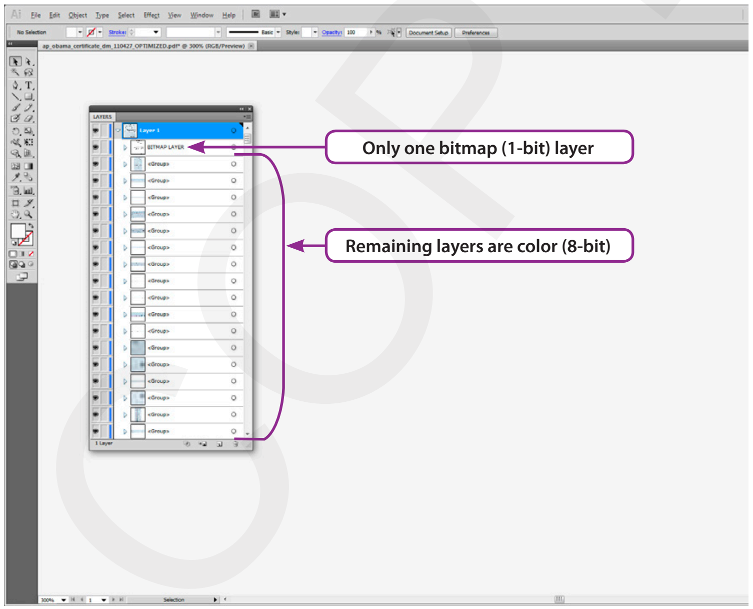

Only one bitmap (1-bit) layer
Remaining layers are color (8-bit)

Figure 12: Layers panel displays typical optimized layer pattern

Optimization Layer Patterns

A detour to discuss optimization is needed. Optimization refers to the process used when saving a file (in this case as a PDF) in which image quality is attempted to be maintained (or *optimized*) while lowering the file size. Attempts are made to explain the presence of layers by using an optimization argument—but the optimization excuse fails—*on multiple levels*. Despite efforts to claim otherwise, the layers in Obama's PDF are a ***big deal*** to prove manipulation.

It is highly recommended to review *page 5* of my previous report (released on March 1, 2012) through the Maricopa County Sheriff's Office, which briefly explains the difference between layers produced by manually manipulation as opposed to those produced by an automated optimized process. The report can be downloaded from the following link:

http://www.mcso.org/MultiMedia/PressRelease/MARAZEBESTREPORT.pdf

To expand on the linked report, one item needs clarification. While it is true that optimization ***can*** cause layers—not ***all optimized PDF files create automated layers***. This is a crucial concept. To restate the point: Optimization only creates layers from specific software programs—not all programs produce automated layers when optimizing. Proponents of the optimization argument would have you believe that layers are *easily* and *always* generated when simply saving as an optimized PDF file—***not true***! Acrobat Pro is the most commonly used program that ***can*** produce automated layers when optimizing as a PDF (but this is still dependent on the method and options used).

Programs like Photoshop or Illustrator do not produce automated layers—these programs only reflect layers that are digitally *manufactured by the user* and ***will not generate any additional layers*** when optimizing as a PDF. In fact, an optimized PDF from Photoshop usually results in a flattened file with one layer only.

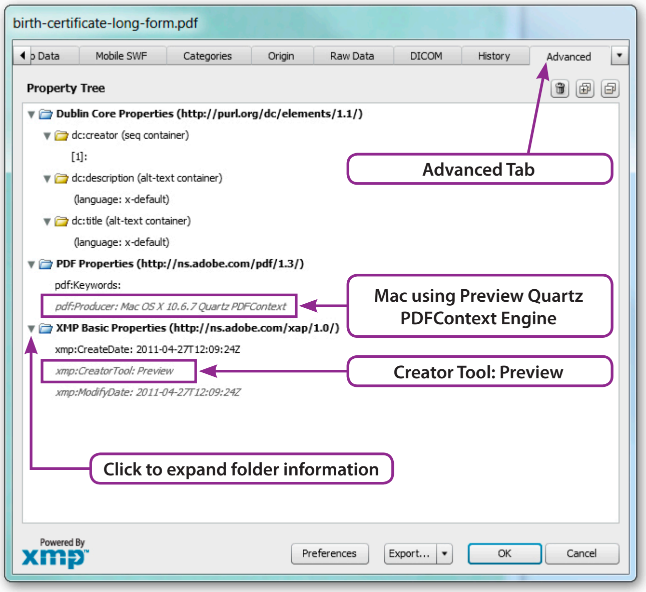
Here's an additional problem for the optimization argument: The metadata code for Obama's PDF states that *Preview* was used to generate the optimized PDF. Preview is a Mac-based program (using a Quartz PDFContext engine). The program Preview, similar to Illustrator, will maintain existing layers created by the user, but will ***not*** generate layers that *do not already pre-exist* in the file.

Figures 10–12 used the AP version of Obama's LFBC as a test file to optimize within Acrobat Pro 9.0 to demonstrate automated layer patterns. **Figure 10** shows the massive numbers of layers typically generated in an optimized PDF (compared to the nine layers in Obama's PDF file).

Figure 11 is a close-up of how the registrar stamp is divided onto multiple layers—cut into random rectangles. Text is chopped haphazardly across layers which often contain surrounding background colors. The text cannot be relocated independently of the background image. Additionally, text is not in one piece. By contrast, Obama's PDF has intelligence to the layer order: The background image is completely contained on one layer and all remaining layers are independent of the background.

Figure 12 illustrates the typical *1-bit v 8-bit* pattern results of optimized PDF layers:

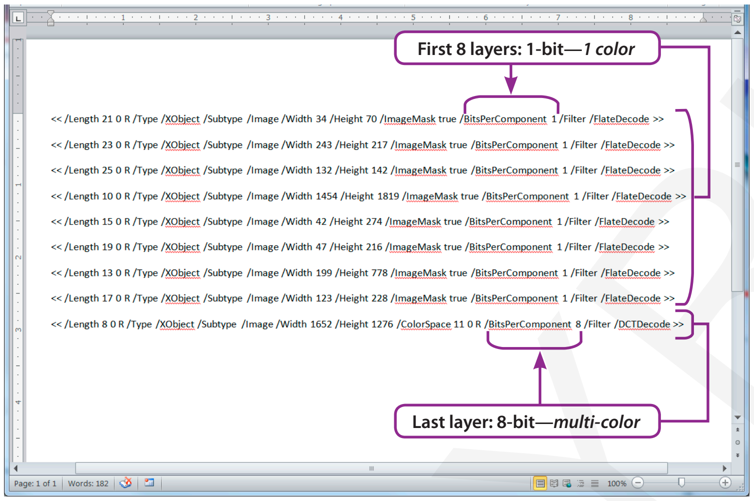
- A ***single*** *1-bit* layer at the top of the layer stack— containing the black color *[Obama's PDF has eight, no visible black]*
- ***All*** remaining layers are *color* layers (*8-bit*) containing a variety of color values *[Obama's PDF has only one color layer]*



Property Tree

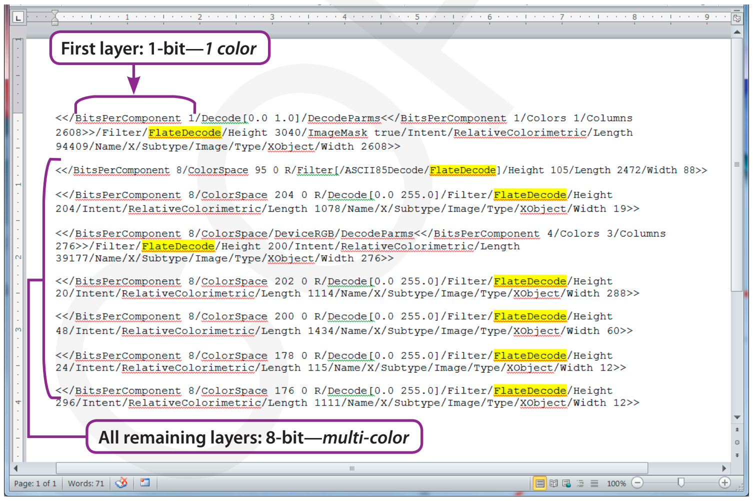
- Dublin Core Properties (http://purl.org/dc/elements/1.1/)
  - dc:creator (seq container)
    - [1]:
  - dc:description (alt-text container)
    - (language: x-default)
  - dc:title (alt-text container)
    - (language: x-default)
- PDF Properties (http://ns.adobe.com/pdf/1.3/)
  - pdf:Keywords:
  - pdf:Producer: Mac OS X 10.6.7 Quartz PDFContext
- XMP Basic Properties (http://ns.adobe.com/xap/1.0/)
  - xmp:CreateDate: 2011-04-27T12:09:24Z
  - xmp:CreatorTool: Preview
  - xmp:ModifyDate: 2011-04-27T12:09:24Z

Figure 13: Metadata code offers Mac and Preview information


First 8 layers: 1-bit—*1 color*
Last layer: 8-bit—*multi-color*
Page: 1 of 1 | Words: 182
100%

```
<< /Length 21 0 R /Type /XObject /Subtype /Image /Width 34 /Height 70 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 23 0 R /Type /XObject /Subtype /Image /Width 243 /Height 217 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 25 0 R /Type /XObject /Subtype /Image /Width 132 /Height 142 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 10 0 R /Type /XObject /Subtype /Image /Width 1454 /Height 1819 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 15 0 R /Type /XObject /Subtype /Image /Width 42 /Height 274 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 19 0 R /Type /XObject /Subtype /Image /Width 47 /Height 216 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 13 0 R /Type /XObject /Subtype /Image /Width 199 /Height 778 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 17 0 R /Type /XObject /Subtype /Image /Width 123 /Height 228 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 8 0 R /Type /XObject /Subtype /Image /Width 1652 /Height 1276 /ColorSpace 11 0 R /BitsPerComponent 8 /Filter /DCTDecode >>
```

Figure 14: Object code extracted for Obama PDF layers


First layer: 1-bit—*1 color*
All remaining layers: 8-bit—*multi-color*
Page: 1 of 1 | Words: 71
100%

```
<</BitsPerComponent 1/Decode[0.0 1.0]/DecodeParms<</BitsPerComponent 1/Colors 1/Columns
2608>>/Filter/FlateDecode/Height 3040/ImageMask true/Intent/RelativeColorimetric/Length
94409/Name/X/Subtype/Image/Type/XObject/Width 2608>>

<</BitsPerComponent 8/ColorSpace 95 0 R/Filter[/ASCII85Decode/FlateDecode]/Height 105/Length 2472/Width 88>>

<</BitsPerComponent 8/ColorSpace 204 0 R/Decode[0.0 255.0]/Filter/FlateDecode/Height
204/Intent/RelativeColorimetric/Length 1078/Name/X/Subtype/Image/Type/XObject/Width 19>>

<</BitsPerComponent 8/ColorSpace/DeviceRGB/DecodeParms<</BitsPerComponent 4/Colors 3/Columns
276>>/Filter/FlateDecode/Height 200/Intent/RelativeColorimetric/Length
39177/Name/X/Subtype/Image/Type/XObject/Width 276>>

<</BitsPerComponent 8/ColorSpace 202 0 R/Decode[0.0 255.0]/Filter/FlateDecode/Height
20/Intent/RelativeColorimetric/Length 1114/Name/X/Subtype/Image/Type/XObject/Width 288>>

<</BitsPerComponent 8/ColorSpace 200 0 R/Decode[0.0 255.0]/Filter/FlateDecode/Height
48/Intent/RelativeColorimetric/Length 1434/Name/X/Subtype/Image/Type/XObject/Width 60>>

<</BitsPerComponent 8/ColorSpace 178 0 R/Decode[0.0 255.0]/Filter/FlateDecode/Height
24/Intent/RelativeColorimetric/Length 115/Name/X/Subtype/Image/Type/XObject/Width 12>>

<</BitsPerComponent 8/ColorSpace 176 0 R/Decode[0.0 255.0]/Filter/FlateDecode/Height
296/Intent/RelativeColorimetric/Length 1111/Name/X/Subtype/Image/Type/XObject/Width 12>>
```

Figure 15: Object code extracted for optimized file layers

Object Code 1-Bit v 8-Bit Layer Patterns

The metadata and object code will be examined in this section to reinforce what has been discussed. **Figure 13** metadata can be viewed by opening Obama's PDF file in Illustrator and then:

- Go to the **File** menu > **File Info**
- Scroll to locate the **Advanced** tab, click to expand all folder information

The metadata seen in **Figure 13** confirms the Mac program—*Preview*—was used. Preview is a program that is limited in what it can do—mostly used to view files (as its name suggests). Preview can open ***any*** program file type—and it was used as a last step to save Obama's file (as an optimized PDF). By resaving within Preview, any previous existing metadata history is eliminated and replaced with Preview metadata. In other words, the mere fact that Preview is the only program displayed in the metadata does ***not*** mean Preview was the only program used in the creation process. Resaving a file (to PDF) from within Preview will remove any prior metadata evidence. The digital trail of image manipulation from previous programs is gone and any prior metadata information is no longer available. And as mentioned on the previous page, no additional layers are generated from within Preview beyond the pre-existing layers created prior to opening the file in Preview. The layers ***had*** to come from another program.

In addition to metadata, the object code can be viewed easily by opening Obama's PDF file with *WordPad* or *Word*. These text editing programs can provide a glimpse into the object code description of layer properties. Press **Ctrl+F** to display the *Find* dialog box and do a search for ***BitsPer*** (no space) to find the layer object code. **Figure 14** displays the Obama PDF object code for each layer (extracted and pasted into a new Word document for a cleaner view).

As a side note, at the end of each code line, notice that *FlateDecode* is used for the first 8 layers, while *DCTDecode* is used for the last color layer. This will be discussed later in the report regarding printing options—but is merely mentioned to debunk an argument floated which claims the *JBIG2* file format played a role in producing layers. If the *JBIG2* argument were true, the object code would reflect *JBIG2Decode* for any of the nine layers. As you can see—*it does not*. Additionally, *JBIG2* applies to black and white images—not color.

Figure 14 also confirms that the first 8 layers of code (presented in no particular order) all contain 1-bit layer information, and only the ninth layer contains color (or 8-bit) information.

In contrast, **Figure 15** reflects a sampling of the layer object code from the AP version (optimized from previous page). The object code is too numerous to extract for all the layers in the Figure, but the pattern is visible in this optimized file as follows:

- ONE 1-bit (black color value) layer
- All remaining object code layers are 8-bit (color) containing any combination of remaining image items (such as text, form lines, and background)

Figure 15 also highlights (in yellow) that *FlateDecode* is the typical decoder applied—no sign of *JPBIG2Decode* as the opposition claims. Again, the AP object code in **Figure 15** confirms the number of *1-bit v 8-bit* layer patterns viewed in the *Layers* panel (seen in **Figure 12**).

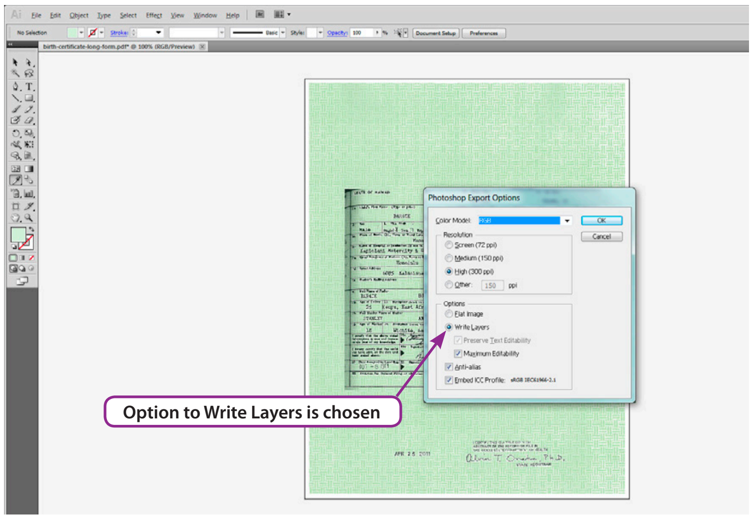


Figure 16: Photoshop Export Options dialog box

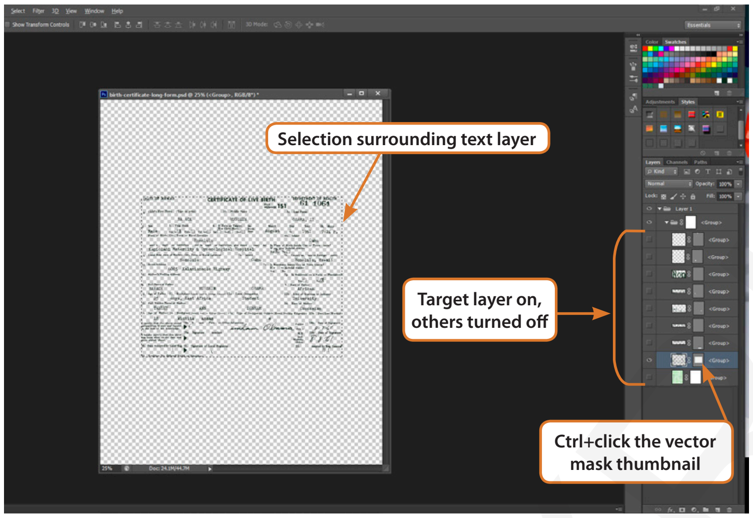

Selection surrounding text layer
Target layer on, others turned off
Ctrl+click the vector mask thumbnail

Figure 17: Layer 8 form text isolated with crop selection

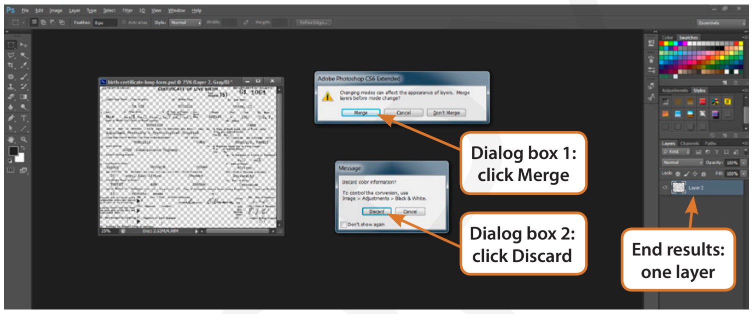


Figure 18: Converting to Grayscale Image Mode

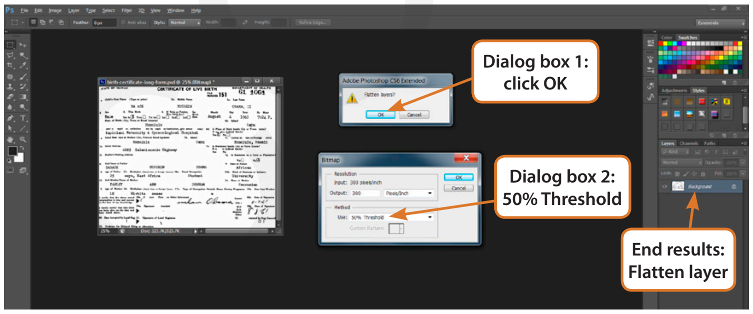


Figure 19: Final conversion to Bitmap Image Mode

Producing Bitmap (1-Bit) Layers

The next two sections will address the following questions respectively:

- How are multiple 1-bit layers produced within a file?
- How can a 1-bit layer represent a color other than black?

NOTE: Answering these questions is not meant to imply that these methods are the same used by the forger(s). The purpose is to offer a *plausible* solution, but it is not offered as the *absolute* method since there is always more than one way to manipulate images.

Converting an image to bitmap mode within Photoshop will produce a black and white image file with the desired 1-bit (*ImageMask true*) attribute. The process of converting to a *bitmap image mode* should not be confused with saving a file in a *bitmap* file format. *Layer 8* in Obama's PDF—the *form text* layer—is the largest and easiest to see and will be used to demonstrate the process. To obtain a similar layered Photoshop file, simply export the file from Illustrator (**File** > **Export**) as a Photoshop (**PSD**) file with the option to preserve the layers as seen in **Figure 16**.

Open the exported PSD file in Photoshop. Notice each layer in the exported PDF file contains a vector mask which will be used to properly crop each layer (to avoid a large layer size). The bitmap color mode will result in a flattened file; thus to produce multiple 1-bit layers (for final exported output), each intended layer would need to be flattened as a separate bitmap layer file. The steps below would then need to be repeated separately for each intended exported 1-bit layer. Thus, the following was repeated for each 1-bit layer compiled in the final Obama PDF file results:

- Turn off any undesired layers while leaving the target layer on
- **Ctrl+click** on the vector mask thumbnail found on the target layer to obtain a selection (as seen in **Figure 17**)
- Go to the **Image** menu > **Crop**. Press **Ctrl+D** to deselect after the crop is complete
- Go to the **Image** menu > **Mode** > **Grayscale** (color information needs to be discarded before the Bitmap option is available)
- Click **Merge** in the first dialog box presented (any layers turned off will be discarded)
- Click **Discard** in the second dialog box presented (to remove any color information). The result will be one-layer, and all color converted to grayscale values (see **Figure 18**)
- Perform this **OPTIONAL STEP ONLY** if the image color is lighter than 50% gray—as in the case of the two top white dot layers—choose black as a foreground color, press **Alt+Shift+Delete** (to fill any pixels on the layer with black)
- Go to the **Image** menu > **Mode** > **Bitmap** and click **OK** in the first dialog box to flatten layers
- In the second dialog box, decide on the resolution output and choose the **50% Threshold** option. This option will turn any 50% gray value or darker to black, and any lighter value to white (the reason behind the recommended optional step for the white dot layers). Click **OK** (see **Figure 19**)

The result is a flattened one layer (1-bit) file, with black as the only color. Save the file as a separate PSD file (or any available file format that supports bitmap mode). Return to the original starting PSD file to repeat the process for each layer as needed.

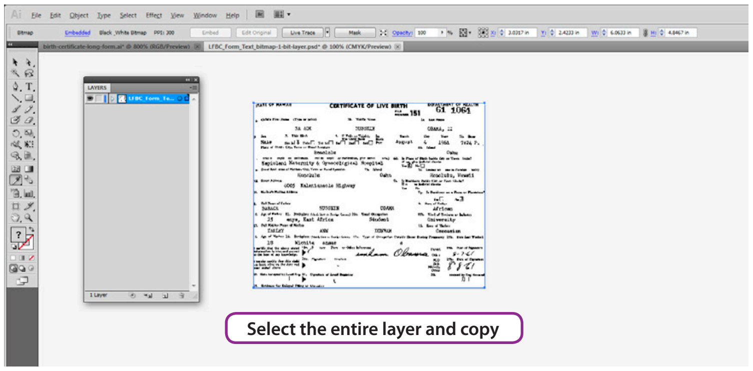


Figure 20: Open the bitmap file in Illustrator, select and copy

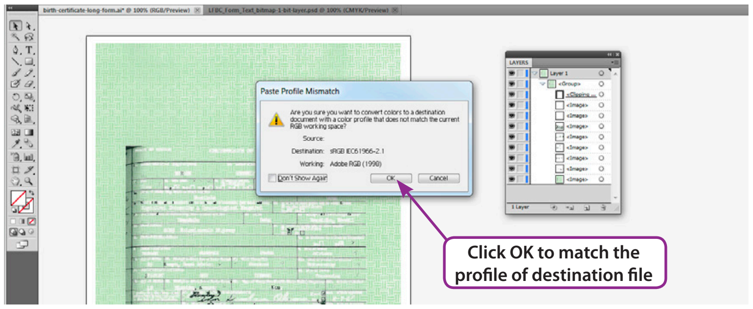


Figure 21: Paste the bitmap file in Illustrator

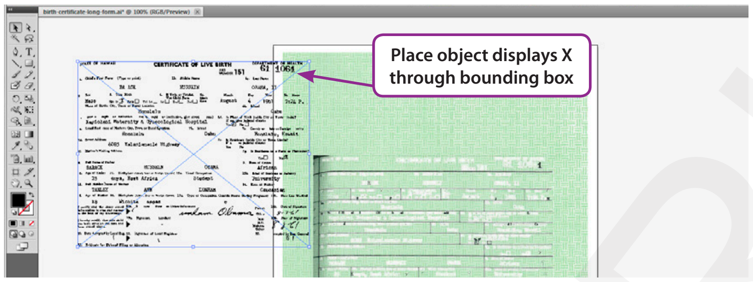


Figure 22: Alternate Place method

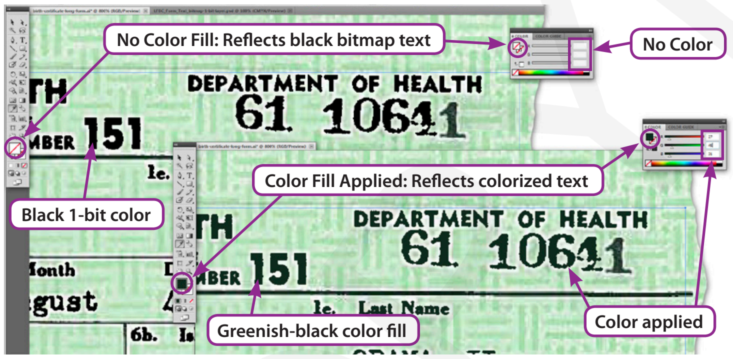

No Color Fill: Reflects black bitmap text
No Color
Black 1-bit color
Color Fill Applied: Reflects colorized text
Greenish-black color fill
Color applied

Figure 23: A Fill color can be applied to colorize a 1-bit layer

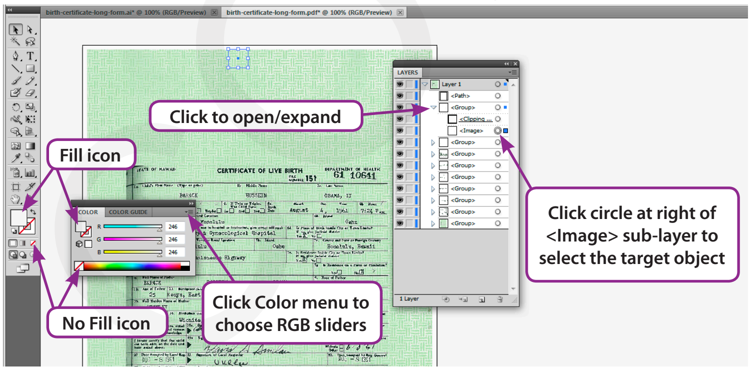

Click to open/expand
Fill icon
Click circle at right of <Image> sub-layer to select the target object
No Fill icon
Click Color menu to choose RGB sliders

Figure 24: Select the <Image> sub-layers to change color Fill

Compiling & Colorizing in Illustrator

After each layer is extracted and converted to a bitmap mode file, the layers are ready to compile and colorize within Illustrator. For the sake of speed, assume the Illustrator file has been compiled with all the layers in position except for the form text layer (extracted from the previous exercise).

Assume all of the 1-bit layers were added with a similar process, merely repeated for each separate layer. With the layered file opened in Illustrator, here is how to add the missing text layer:

- Open the bitmap file in Illustrator. Select the object and press **Ctrl+C** to copy (see **Figure 20**)
- Switch to the opened Obama Illustrator file, and press **Ctrl+V** to paste. Click **OK** if a **Paste Profile Mismatch** dialog box appears as seen in **Figure 21**

NOTE: An alternative method to the steps listed above would be to use **File** > **Place** within the original Obama certificate file. However that method creates a visual clue (of an embedded link) by displaying an "X" placeholder indicator within the object's bounding box (as seen in **Figure 22**). This method is absolutely a more direct route to take—but due to the absence of the "X" linked object indicator in Obama's PDF file (as seen in **Figure 23**), the copy and paste method is a probable workaround method.

As a reminder, there are other minor details that have been covered in prior reports and not the focus of the colorized bitmap discussion. These would include details that occurred when the Obama PDF was compiled, such as the objects brought in requiring a -90° rotation (based on the *Links* panel information). Also, each new layer added to the file will (by default) paste at the top of the layer stack, but the layers can easily be dragged to the preferred position within the stack (and even renamed if desired). The **Flatten Artwork** option in the *Layers* panel would have been applied if additional layers were created during the process, and all objects were grouped to a **Clipping Mask Path**.

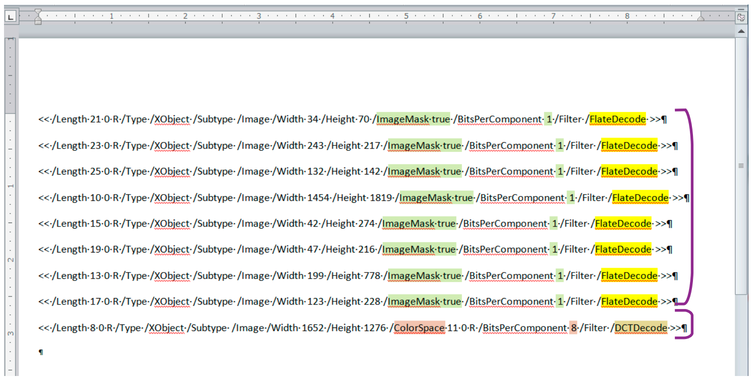
Once the layer is properly positioned, all that remains is assigning a desired color (of choice) as follows:

- With the target object still selected, choose a color **Fill** as desired (refer to **Figure 23**)

It's that easy! When a layer has this 1-bit (ImageMask true) attribute, the white displays as transparency in Illustrator and black will take on ***any*** color applied as a **Fill**. At first glance, it is easy to draw a false impression that no black exists within Obama's digital file—but nothing could be further from the truth. Try this test:

- Open Obama's PDF in illustrator
- On the *Layers* panel, click to open (expand) any of the first eight **<Group>** sub-layer objects
- Click the select circle icon to the right of the sub-layer titled as ***<Image>***—***not*** the *<Clipping Path>* sub-layer (refer to **Figure 24**)
- Turn off the color **Fill** (the icon option with the slash seen in **Figure 24**). Even the off-white dots in the first two layers are black (not off-white)
- To further prove the point, with any of the eight bitmap ***<Image>*** sub-layers selected, pick any **Fill** color. Play with the *Color* panel sliders and watch the layer change color as desired

NOTE: Press **Ctrl+Z** to undo as needed while experimenting.



```
<< /Length 21 0 R /Type /XObject /Subtype /Image /Width 34 /Height 70 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 23 0 R /Type /XObject /Subtype /Image /Width 243 /Height 217 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 25 0 R /Type /XObject /Subtype /Image /Width 132 /Height 142 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 10 0 R /Type /XObject /Subtype /Image /Width 1454 /Height 1819 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 15 0 R /Type /XObject /Subtype /Image /Width 42 /Height 274 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 19 0 R /Type /XObject /Subtype /Image /Width 47 /Height 216 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 13 0 R /Type /XObject /Subtype /Image /Width 199 /Height 778 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 17 0 R /Type /XObject /Subtype /Image /Width 123 /Height 228 /ImageMask true /BitsPerComponent 1 /Filter /FlateDecode >>
<< /Length 8 0 R /Type /XObject /Subtype /Image /Width 1652 /Height 1276 /ColorSpace 11 0 R /BitsPerComponent 8 /Filter /DCTDecode >>
```

Figure 25: Object code extracted for Obama PDF layers

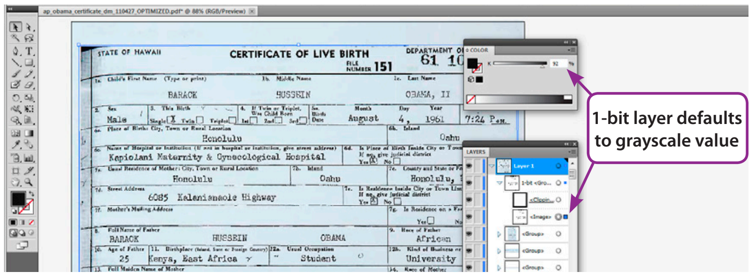


Figure 26: Optimized 1-bit layer reflects black as expected

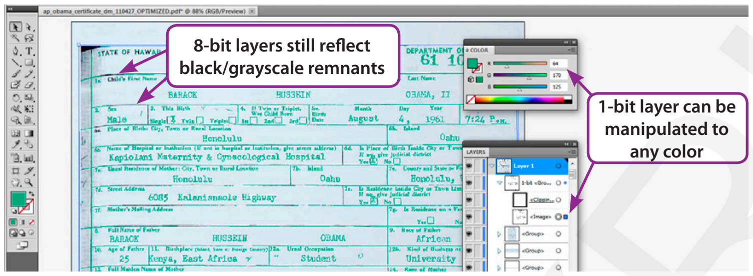

8-bit layers still reflect black/grayscale remnants
1-bit layer can be manipulated to any color

Figure 27: Optimized 1-bit layer Fill color can be manipulated

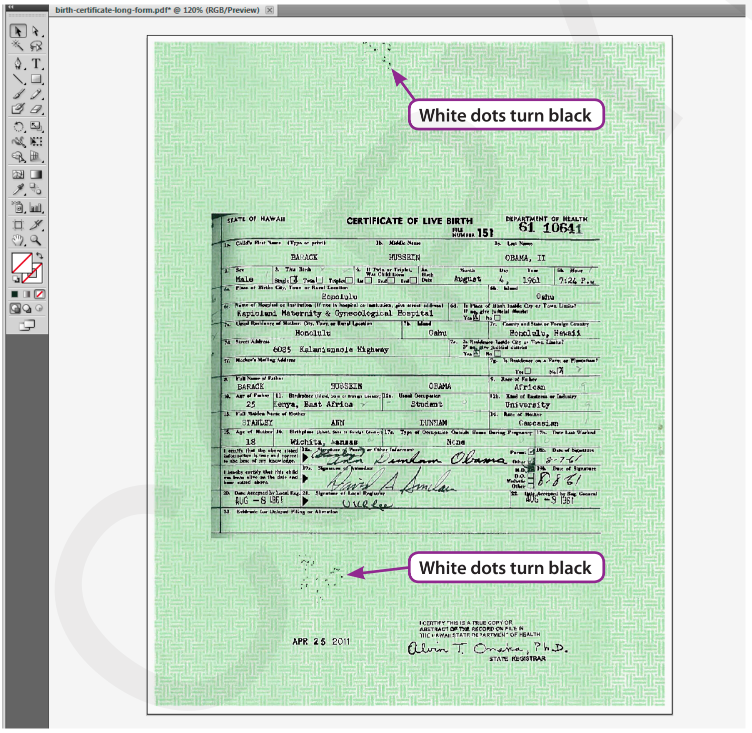


Figure 28: Fill color turned off for all 1-bit layers

ImageMask True Property Behavior

Figure 25 provides a reminder that the object code displayed for Obama's PDF contains 8 layers with the following properties:

- ImageMask true
- BitsPerComponent 1 (1-bit layer)
- FlateDecode

The last layer contains varying colors for the safety-paper and anti-aliased elements with the following object code properties:

- ColorSpace
- BitsPerComponent 8 (8-bit layer)
- DCTDecode

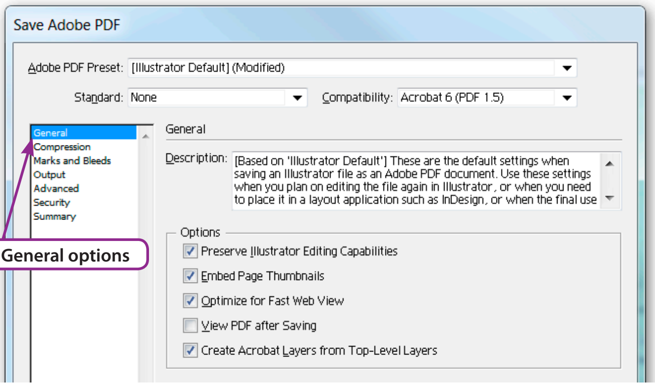
Think of the ***ImageMask true*** property (for all 1-bit objects) as analogous to applying paint through a cut-out stencil. The white areas of the bitmap layer act as the stencil mask (displaying as transparent within Illustrator). The paint applied reaches the layer where black resides. Due to this stencil-like (*ImageMask true*) property, when the **Fill** color is turned *off*, Illustrator's *Eyedropper* tool will not register color values for the black and white stencil. Only when a **Fill** color is applied, will the *Eyedropper* register a color value. This also reinforces the *Eyedropper* behavior (seen earlier) in which the **Fill** color is the only color that displays a value even when clicking in the stencil-like (or transparent) areas within the object.

Understand that while it is possible for optimization to create a 1-bit layer to be colorized (by normal scanning and saving procedures)—the key difference is the optimized file result will contain only ONE 1-bit layer and the color applied to that layer will reflect a grayscale value close to black. **Figure 26** uses the optimized AP file once again, to see an example of this. When the file was optimized, the process attempted to pull all the pixels that had the darkest black color (into the top 1-bit layer) and assigned a grayscale color representing that black value. **Figure 27** shows that the layer behaves similarly as described; in which the black 1-bit layer acts as a stencil—the color can be *manipulated* to any color value (in Illustrator). Notice that other grayscale black values on ***8-bit*** layers cannot be manipulated in the same manner.

Obama's PDF file implies a choice to manipulate the file to contain not **one**—but ***eight*** bitmap layers. A choice has to be made by the user to import and compile this many 1-bit layers. But even more importantly, a choice has to be made to manipulate the color of the 1-bit layers to display a color other than black. The fact that the 1-bit layers in Obama's PDF contain colors ***other than black*** is not due to any variation of scanning settings, nor any variation of optimization settings. Obama's PDF contains color outside the grayscale range. This form of color manipulation is just that—*manipulation*—and further proof of a manufactured file.

Figure 28 displays the results if the color **Fill** is turned off for all 1-bit layers (in Obama's file). Note the two white dot layers also turn black. To be clear—when working in Illustrator—any ***vector-based*** object can easily have a **Fill** color applied (as well as a color *outline border* known as a **Stroke**). However, the layers in the Obama PDF are ***not vector***; they are ***raster-based*** (also known as pixel-based). The 1-bit layers with the *ImageMask true* property are the only raster-based objects that can be colorized as described (using the **Fill** color)—no other ***raster-based*** file formats will behave in such a manner.

Next section will cover the significance of the FlateDecode and DCTDecode properties seen in the object code.


Save Adobe PDF

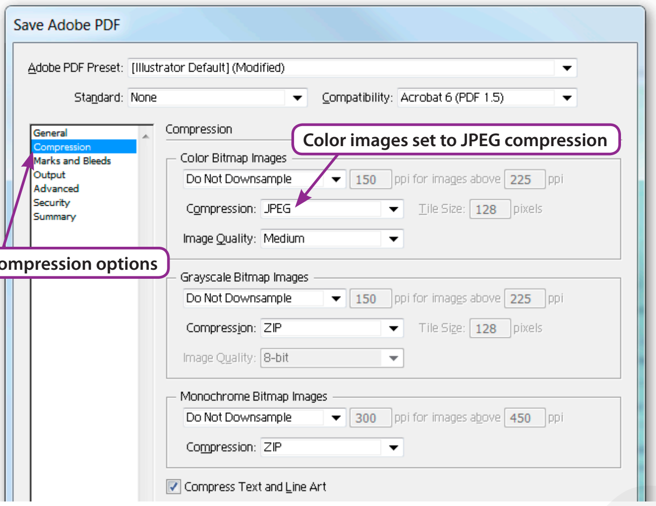
Adobe PDF Preset: [Illustrator Default] (Modified)
Standard: None Compatibility: Acrobat 6 (PDF 1.5)

General

Description: [Based on 'Illustrator Default'] These are the default settings when saving an Illustrator file as an Adobe PDF document. Use these settings when you plan on editing the file again in Illustrator, or when you need to place it in a layout application such as InDesign, or when the final use

Options
- [✓] Preserve Illustrator Editing Capabilities
- [✓] Embed Page Thumbnails
- [✓] Optimize for Fast Web View
- [ ] View PDF after Saving
- [✓] Create Acrobat Layers from Top-Level Layers

Figure 29: Save Adobe PDF dialog box—General options


Save Adobe PDF

Adobe PDF Preset: [Illustrator Default] (Modified)
Standard: None Compatibility: Acrobat 6 (PDF 1.5)

Compression

Color Bitmap Images
Do Not Downsample 150 ppi for images above 225 ppi
Compression: JPEG Tile Size: 128 pixels
Image Quality: Medium

Grayscale Bitmap Images
Do Not Downsample 150 ppi for images above 225 ppi
Compression: ZIP Tile Size: 128 pixels
Image Quality: 8-bit

Monochrome Bitmap Images
Do Not Downsample 300 ppi for images above 450 ppi
Compression: ZIP

[✓] Compress Text and Line Art

Figure 30: Save Adobe PDF dialog box—Compression options

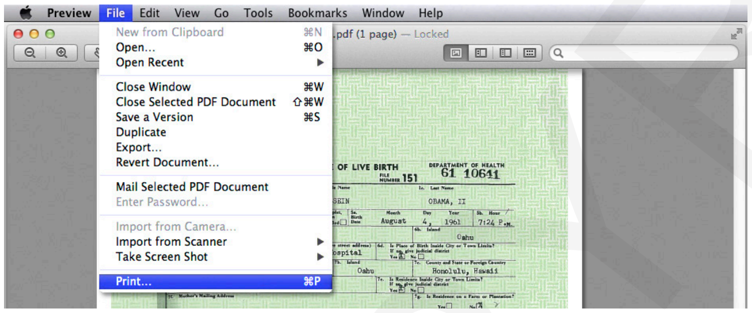
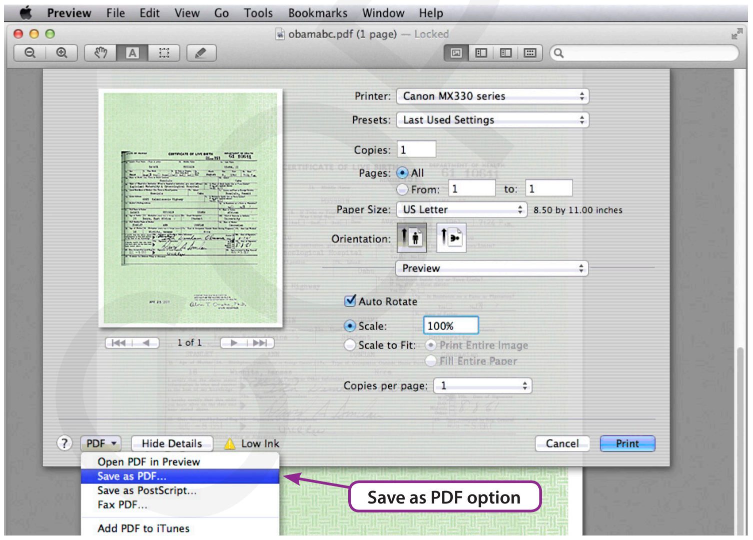


Figure 31: File menu > Print


obamabc.pdf (1 page) — Locked

Printer: Canon MX330 series
Presets: Last Used Settings
Copies: 1
Pages: (●) All ( ) From: 1 to: 1
Paper Size: US Letter 8.50 by 11.00 inches
Orientation:
Preview
[✓] Auto Rotate
(●) Scale: 100%
( ) Scale to Fit: Print Entire Image / Fill Entire Paper
Copies per page: 1

Figure 32: Save Adobe PDF dialog box—Compression options

Printing to PDF

One advantage to compiling multiple 1-bit layers would certainly be to reduce file size. As seen in the previous sections, in order to control the presence of multiple 1-bit layers (along with the colors applied); each layer was saved and compiled separately within Illustrator.

When saving the Illustrator file as a PDF, the default settings will generate FlateDecode for all layers. Recall that using FlateDecode for the 1-bit layer objects is consistent with the object code reflected in Obama's PDF (refer to **Figure 25**). But when FlateDecode is applied to the bottom **color** (safety-paper) layer—it will bloat the file size, making it more challenging to lower the file size while maintaining image quality consistent with Obama's PDF.

As previously seen in **Figure 25**, the object code in Obama's PDF reflects a different decoder for the 8-bit color layer—*DCTDecode*—which is analogous to JPEG compression for color images. By using DCTDecode for this layer, the file size is reduced substantially.

How are different decoders applied to the various layers? This is simply a matter of adjusting a few settings in the PDF dialog box. In this example, from Illustrator's **File** menu, choose **Save As** and be sure to adjust the **Save as type** option to *PDF*. **Figure 29** displays the initial **General** page options in the *Save Adobe PDF* dialog box.

By default, the **Compression** page options are set to ***Zip*** for all three image types: *Color Bitmap*, *Grayscale Bitmap*, and *Monochrome Bitmap*. The Zip option will produce a FlateDecode result. However, by adjusting the option for the *Color Bitmap* **Compression** to **JPEG** using a ***Medium*** or less quality setting (as seen in **Figure 30**), the color information on the bottom 8-bit safety-paper layer will produce *and match* the JPEG DCTDecode compression seen in Obama's object code.

Once the file results are saved from Illustrator as a PDF, the file is then opened and resaved (also known as ***refried***) from Preview for two reasons:

- To scrub any prior metadata evidence (as cover for any digital tracks of manipulation)
- To reduce the file size even further

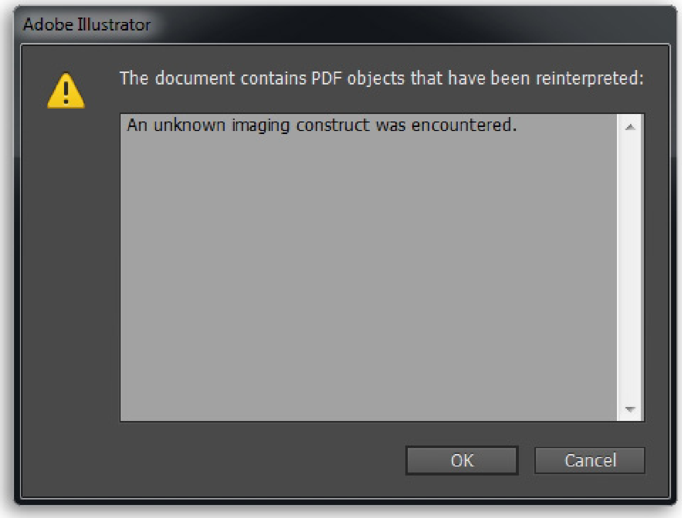
There are two methods to save or refry as a Preview PDF:

- The **File** > **Save As** method (using PDF settings) which produces a noticeably poorer quality result, therefore, the second option was more likely used…
- The **File** > **Print** method seen in **Figure 31** will maintain image quality more effectively while substantially reducing the file size. Choose **Save as PDF** within the *Print* dialog box as shown in **Figure 32**

The final Preview PDF result will contain matching metadata and object code (as seen in Obama's PDF).

To Summarize:

- Layers cannot be created within Preview and had to be manufactured in another program prior to the last step of saving from Preview
- Layers in Obama's file were not produced from an automated *optimization* process which would contain a single layer (not eight) of 1-bit quality (and an optimized 1-bit layer displays a black color value)

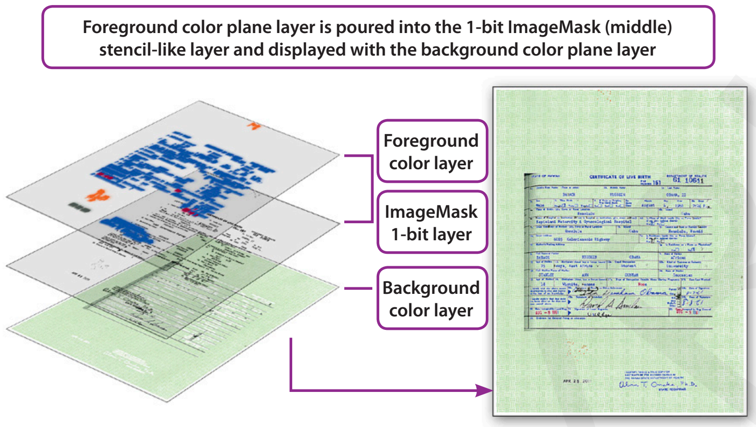


Adobe Illustrator

The document contains PDF objects that have been reinterpreted:

An unknown imaging construct was encountered.

Figure 33: MRC compression generates Illustrator warning message



Foreground color plane layer is poured into the 1-bit ImageMask (middle) stencil-like layer and displayed with the background color plane layer
Foreground color layer
ImageMask 1-bit layer
Background color layer

Figure 34: MRC object code pattern: Two 8-bit layers; one 1-bit layer

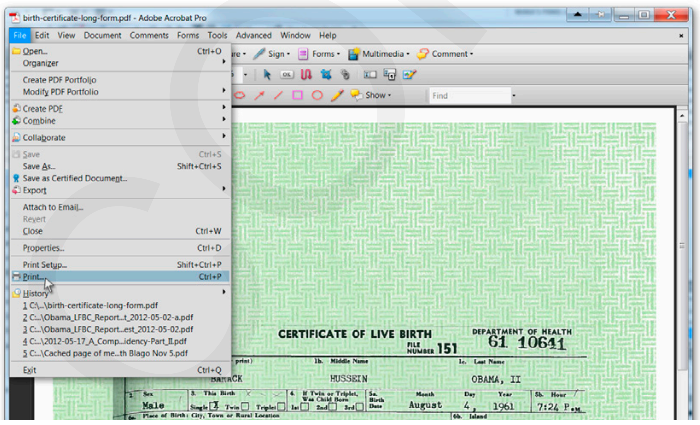


Figure 35: File menu > Print

MRC Explained and Concluding Points

The layers presented in Obama's PDF file were not a product of optimization or any automated process. As previously mentioned, the most likely program to produce optimized layers (from an automated process) would be Adobe's Acrobat Pro—version 9 or earlier uses an *Adaptive* compression method (refer back to the results shown in **Figure 10**).

Acrobat Pro X (version 10) or later uses a method that resembles *MRC* (Mixed Raster Content) compression. MRC was not a factor in Obama's PDF either, but worth noting in case any defenders of Obama's PDF want to throw the MRC argument up against the wall hoping it will stick.

Some disqualifying features of MRC (not found in Obama's PDF) are as follows:

- MRC has a tendency to generate a *warning message* seen in **Figure 33** whenever the file is opened in Illustrator. Obama's PDF file *does not* trigger any such warning message
- Similar to Adaptive compression, MRC will produce only **one** 1-bit (*ImageMask true*) layer in the object code. Obama's PDF file has **eight** 1-bit layers
- Typical layer results will be three object code layers consisting of **two 8-bit** layers and **one 1-bit** layer (seen in **Figure 34**). In contrast, Obama's file has only **one** 8-bit layer (and of course, there's still the problem of the **eight** 1-bit layers)
- **Figure 34** gives an approximate idea of what the layer process and results ***might*** look like ***if*** Obama's PDF file had MRC compression applied. The Figure uses exaggerated colors (for the top 8-bit color layer) to easily demonstrate the process. MRC uses three layers in the object code, but only two layers will display when opened in Illustrator as follows: The bottom 8-bit background layer; and the combined top two layers in which the top 8-bit color layer provides the color values for the 1-bit (stencil-like) middle layer. The process of ***interpreting*** the top two layers (as a combined layer) when opened in Illustrator explains the warning message that objects "*have been reinterpreted*". Again, these layer characteristics are not found in the Obama PDF which contains **nine** layers, and the 1-bit layers do not get their color from a separate 8-bit color layer at the top but rather retain separate **Fill** colors within each 1-bit layer

An accumulated understanding into all the attributes of optimization compared with all the attributes of Obama's PDF file makes it increasingly impossible to defend Obama's file as "normal." Once it becomes clear that the colors in the first eight layers are an applied choice, one does have to wonder why the particular color choices were made. For example, why choose a color for date text that differs from other text layers? But the more important and significant question ***still*** remains: ***Why do these file attributes even exist at all?*** A legitimate file would not contain this many problems—the more that is learned about the file, the more problematic the file becomes.

Obama's PDF file can no longer be referred to as a "*document*," since that term implies it existed and started in paper form. Obama's PDF must be referred to as a "*digital file*" because that is all it has ever been—*manufactured and compiled digitally*—and only exists as a "*document*" when the user goes to the **File** menu and clicks the **Print** option from within the file (as seen in **Figure 35**).