Table of Contents

The Acrobat User

Acrobat Page Capture
Acrobat's built-in OCR functionality allows you to turn a folder full of scanned documents into selectable, searchable text and graphics.

PostScript Tech

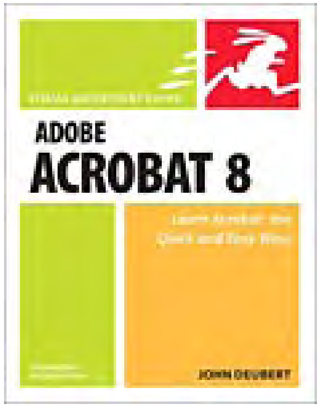
Explicitly Masked Images, Part 2
We continue last June's discussion of masked images, this time seeing how to do it with real images. This will entail learning about the *ReusableStreamDecode* filter.

Class Schedule

October, November, December, January

What's New?

Finished a new book: *Acrobat 8 Visual Quickstart Guide*
Coming out in November from Peachpit Press.

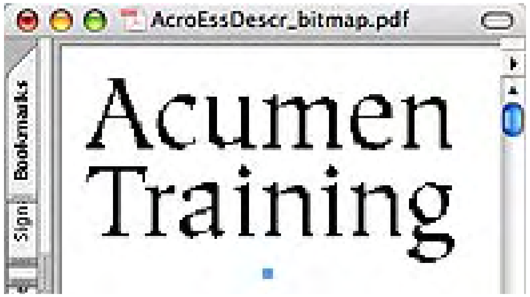


Contacting Acumen

Telephone number, email address, postal address

Journal feedback: suggestions for articles, questions, etc.

Acrobat Page Capture



One task common to many organizations is the conversion of a large collection of paper documents into PDF. Typically, these documents are scanned and the scanned pages are assembled into a PDF document.

While the resulting PDF pages look like the original pages of text, there is, in fact, no text on them at all. Rather, each page contains a bitmapped picture of the original text, rather than the text, itself, illustrated above right.

This is perfectly fine if you simply want to read the documents by eye. However, if you want to apply any of the common computer text functions to the contents of the document (such as searching for words or phrases), then you must convert the bitmap text to actual text, a process known as *Optical Character Recognition* or *OCR* for short.

As it happens, Acrobat has OCR capability built into it and can convert your scanned pages into real pages of searchable text. Adobe calls this feature *Acrobat Page Capture.*

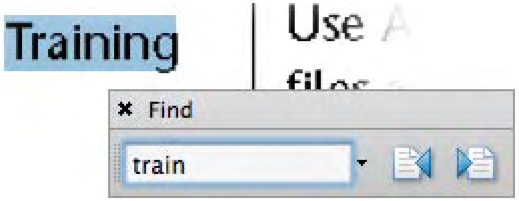
In this issue, we shall see how to use this feature.

Next Page ->

Acrobat Page Capture

Acrobat's *Page Capture* can take an image-only PDF file (that is, a PDF file whose pages consist only of image data, usually from a scanner) and apply OCR to determine the text that is on the page. Acrobat can create two kinds of pages from this conversion:

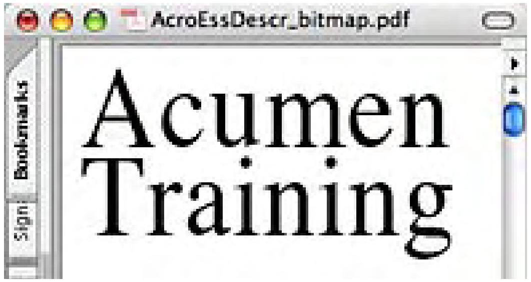
- *Searchable Image* - In this case, Acrobat leaves the image untouched so that the page looks exactly as it did originally. The searchable text is added as an invisible layer that will be used by Search and other text functions.



  While the converted document is searchable, the visual page remains a bitmap, so if you zoom in on the page, the text looks increasingly jagged.

- *Text and Graphics* - Acrobat replaces the full-page image with a combination of text and line art, attempting to use the original fonts.

  While this allows you to zoom in on the page indefinitely without being distracted by escalating jaggies, the font that Acrobat picks will probably not quite match the original, so the text will change its appearance. (Compare the text at right with the original bitmap text, on the previous page; they are clearly different fonts.)

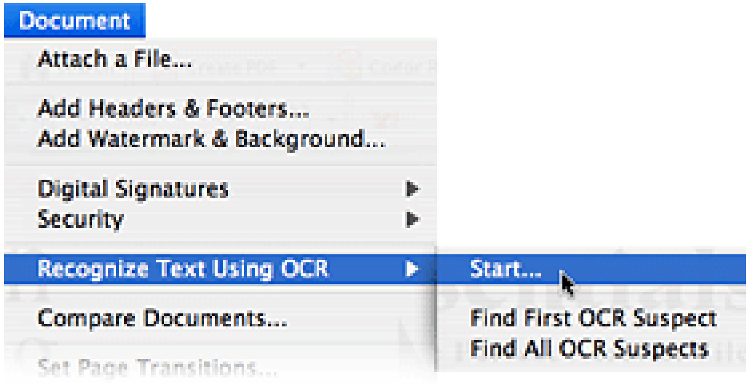


Next Page ->

Converting an Image to Text

To convert a PDF image document to a searchable image or to text and graphics, do the following:

1. With the scanned documen open, select *Document>Recognize Text Using OCR>Start.* Acrobat will present you with the Recognize Text dialog box (below, right).

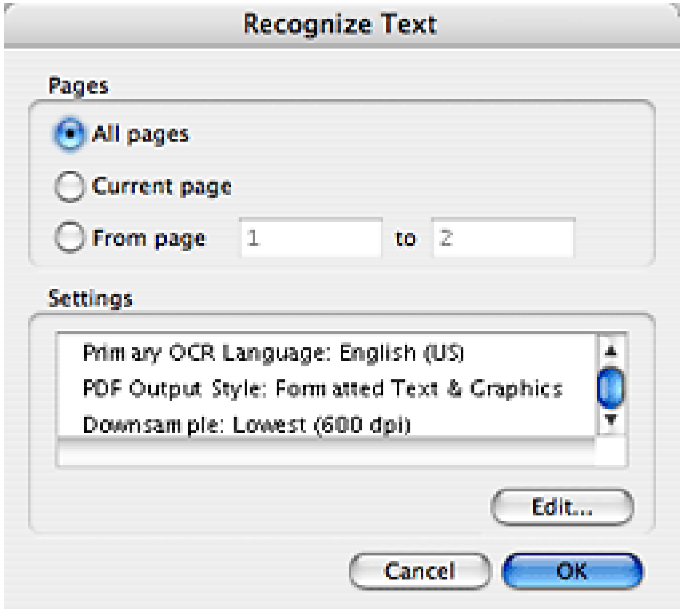


2. Choose the radio button corresponding to the pages you want converted to a searchable image.

   Most of the time, you will probably choose *All pages.*

3. Click the *Edit* button.

   Acrobat will present you with the *Recognize Text Settings* dialog box, on the next page.



Pages

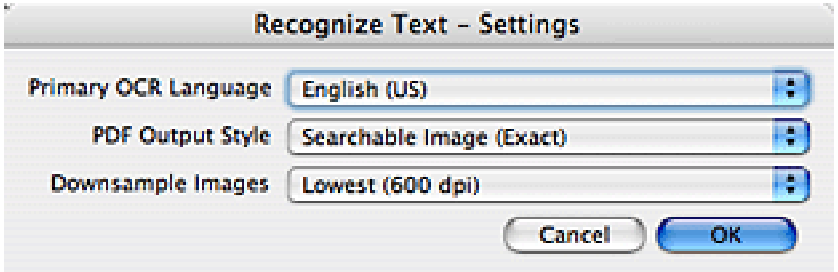
● All pages
○ Current page
○ From page 1 to 2

Settings

Primary OCR Language: English (US)
PDF Output Style: Formatted Text & Graphics
Downsample: Lowest (600 dpi)

Next Page ->

4. In the *PDF Output Style* pop-up menu, select the type of conversion you want Acrobat to perform. Your options are:



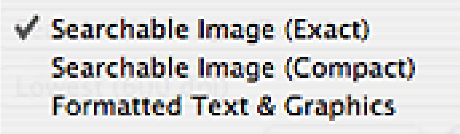
Recognize Text – Settings

Primary OCR Language: English (US)

PDF Output Style: Searchable Image (Exact)

Downsample Images: Lowest (600 dpi)

   - *Searchable Image (Exact)* - Acrobat creates a searchable image using a very accurate algorithm to convert the text.


✓ Searchable Image (Exact)
Searchable Image (Compact)
Formatted Text & Graphics

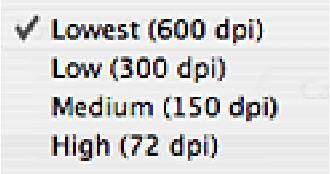
   - *Searchable Image (Compact)* - Acrobat creates a searchable image, but uses a faster and slightly less-accurate OCR method.
   - *Formatted Text & Graphics* - Acrobat replaces each full-page image with real text and line art, doing its best to match the original font.

5. Select a language to be used in the OCR conversion.

   Acrobat 7 provides a list of a dozen or so languages from which you may choose.

6. Pick a resolution to which Acrobat can reduce the image when doing the OCR.

   When Acrobat performs OCR, it reduces the resolution of the image to speed up the processing. The lower the downsampled resolution, the faster the processing, but the less-accurate will be the character recognition. I'd leave this at 600 dpi, myself.



6. Click the OK button to return to the Recognize Text dialog box.

Next Page ->

7. Click the OK button to start the OCR process.

Acrobat will process the bitmapped pages.

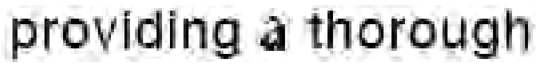
If you chose to make a searchable image, the resulting PDF pages will look the same as they did before, but now all of the text tools will work on the apparently bitmapped text. You can search the text, select it with the *Select* tool, and copy the selection. Of course, what the text tools are really operating on is the underlying text layer.

If you chose to convert the document to text and graphics, you will be looking at a document with real text and line art.

OCR Suspects

It often happens, when analyzing a document, that Acrobat encounters character bitmaps that it can't identify; that character's image might be malformed or the font is too unusual for Acrobat to recognize its character shapes. A section of the page image that Acrobat cannot analyze is called an *OCR suspect*.

Acrobat leaves OCR suspects as bitmaps in-line with the OCR-generated text; in the illustration at right, the letter "a" is a suspect.



providing a thorough

Once you have had Acrobat convert a scanned page to text (whether to a searchable image or to text and graphics), you should examine all of the page's OCR suspects and correct Acrobat's identification of the characters.

Acrobat makes it easy to examine and correct all the document's OCR suspects.

Next Page ->

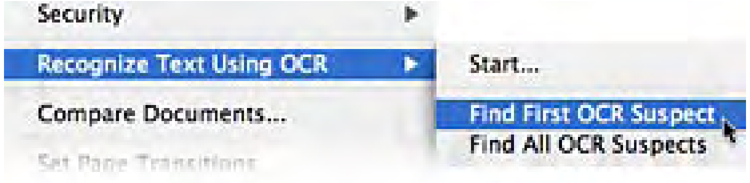
Correcting OCR Suspects

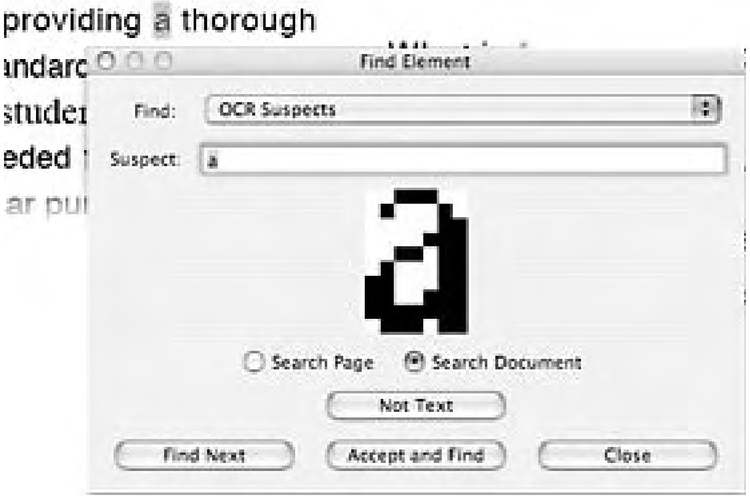
To correct the OCR suspects do the following:

1. Select *Document>Recognize Text using OCR>Find First OCR Suspect.*

   Acrobat will higlight the first suspect in the document and displays the *Find Element* dialog box (below, right).

   This dialog box presents you with a close-up of the unidentified bitmap, a *Suspect* text field, showing Acrobat's best guess as to what characters the bitmap represents, and four buttons that let you tell Acrobat what to do with the OCR suspect.





Find: OCR Suspects

Suspect: a

○ Search Page ● Search Document

2. Correct Acrobat's interpretation of the the suspect in the *Suspect* field (if appropriate) and click on one of the four buttons:

   - *Not Text* tells Acrobat that the bitmap doesn't represent text (perhaps it's an icon) and Acrobat should leave it unchanged.

Next Page ->

- *Find Next* tells Acrobat to move on to the next suspect. The bitmap will remain on Acrobat's list of suspects and will be displayed the next time you look through the OCR suspects.
- *Accept and Find* tells Acrobat to replace the bitmap with whatever text is in the *Suspect* field and then move to the next suspect. You can change the text in the *Suspect* field before clicking this button.
- *Close* finishes the dialog box, returning you to the PDF page.

Repeat the above steps for each OCR suspect. When you have finished, Acrobat will present you with a dialog box telling you there are no more suspects to be found.

Batch Converting Files

Most people who are converting paper documents to PDF have a *lot* of such documents to convert. Doing them by hand, one-by-one gets boring after the first thousand pages, which makes it all the sweeter that you can use the ICR feature in an Acrobat batch sequence; this would let you convert a whole folder of scanned files at once.

This is too useful to not do, so let's see how we go about it.

Next Page ->

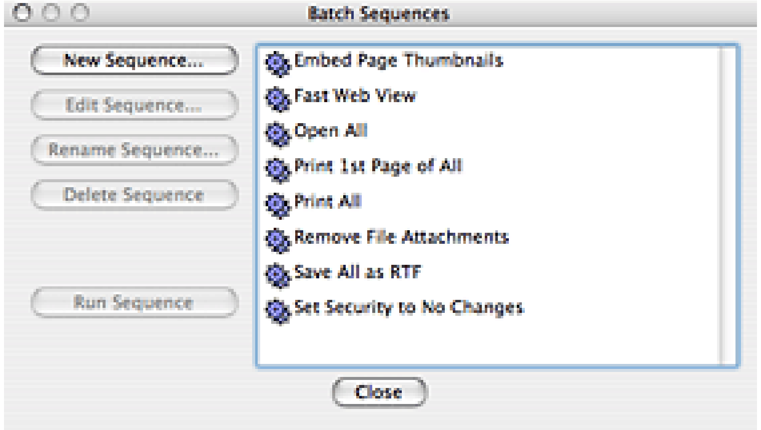
Creating the OCR Sequence

Acrobat does not ship with Page Capture batch sequence, so we need to create one. Do the following:

1. Select *Advanced>Batch Processing.*

   At least, that's where *Batch Processing* is located in Acrobat 7. Adobe playfully changes its location in each new version of Acrobat, so you may need to look around for it. It's there some where if you have Acrobat 5 or later.

   Once you find and select the menu item, Acrobat will display the *Batch Sequences* dialog box. This lists all of the batches available to your copy of Acrobat.

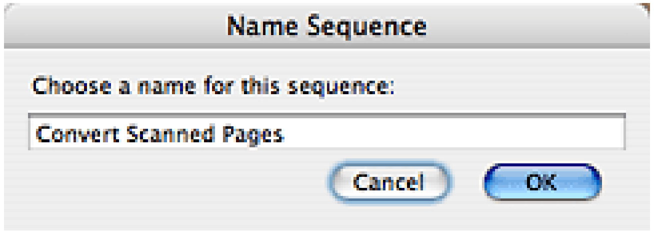


2. Click the *New Sequence* button.

   Acrobat will ask you for a name for the new batch (lower right).

3. Type a name into the text field and click OK.

   Acrobat will display the *Edit Batch Sequence* dialog box (next page).



Choose a name for this sequence:

Convert Scanned Pages

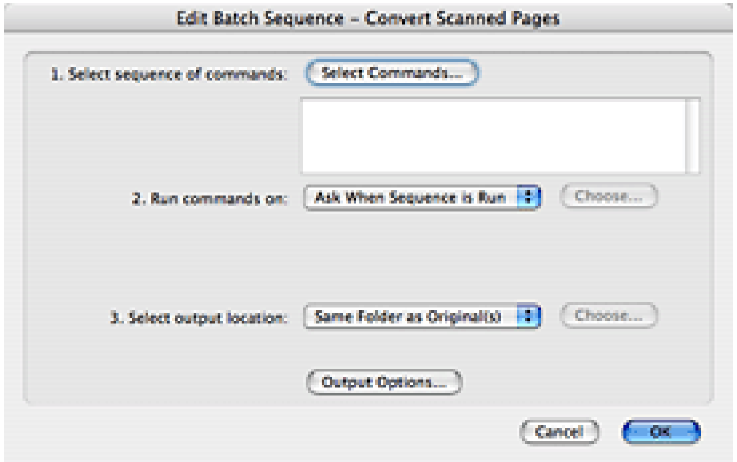
Next Page ->

This dialog box lets you specify the details of your sequence: the actions it should perform, on what files it should operate, and where it should put the result.

4. Click the *Select Commands* button.

   Acrobat will display the *Edit Sequence* dialog box (lower right).

5. In the left pane, select *Recognize Text Using OCR* and click the *Add* button.

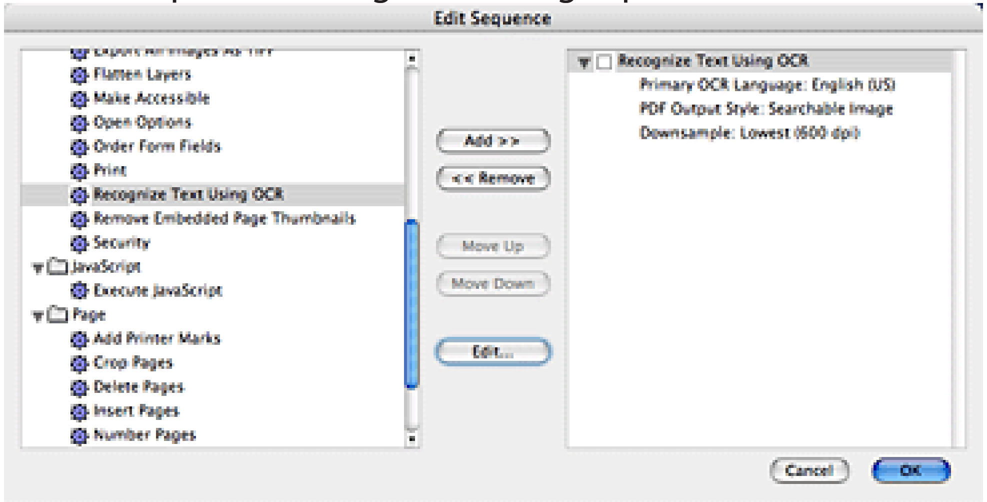


Edit Batch Sequence – Convert Scanned Pages

1. Select sequence of commands:
2. Run commands on: Ask When Sequence is Run
3. Select output location: Same Folder as Original(s)

Acrobat will add this step to the sequence, listing it in the right pane.

6. Click the *Edit* button to get the Recognize Text dialog box we described earlier (next page) and specify the type of conversion and other settings you want for this batch's conversion.



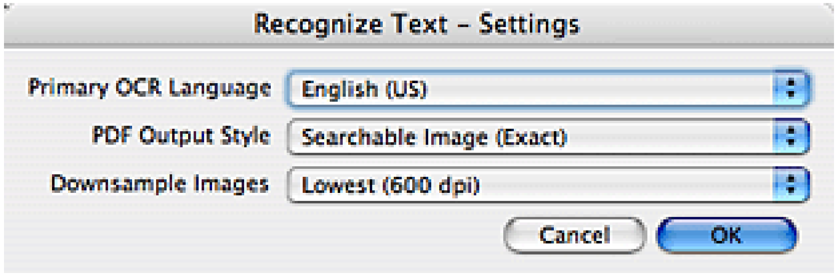
Edit Sequence

Recognize Text Using OCR
Primary OCR Language: English (US)
PDF Output Style: Searchable Image
Downsample: Lowest (600 dpi)

Next Page ->

Click the OK button to return to the *Edit Sequence* dialog box.

7. Click that dialog box's OK button to return to the *Edit Batch Sequence* dialog box, which now displays our new sequence.


Recognize Text – Settings

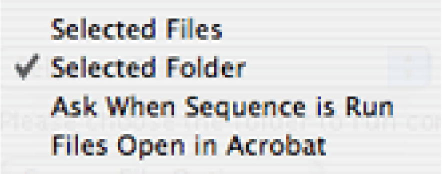
Primary OCR Language: English (US)

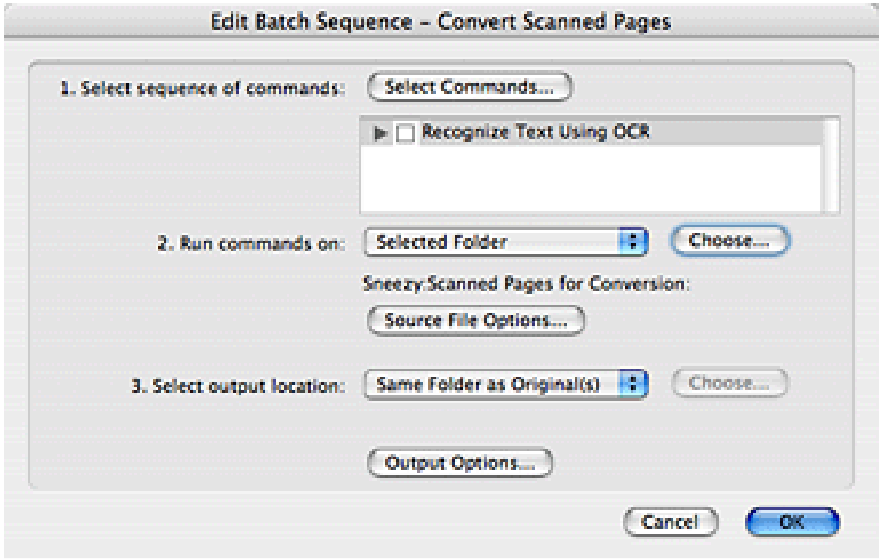
PDF Output Style: Searchable Image (Exact)

Downsample Images: Lowest (600 dpi)

8. In the *Run Commands On* pop-up menu, tell Acrobat where to look for the files the sequence should process.

   This menu gives you several choices:




Edit Batch Sequence – Convert Scanned Pages

1. Select sequence of commands: Recognize Text Using OCR

2. Run commands on: Selected Folder

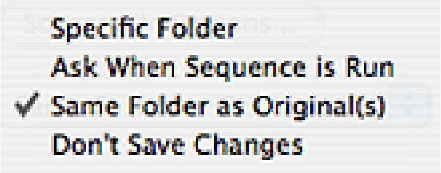
Sneezy:Scanned Pages for Conversion:

3. Select output location: Same Folder as Original(s)

If you have a "hot folder" reserved for scanned documents needing conversion, you would choose *Selected folder* and then click the *Choose* button to select a folder. When you later trigger the sequence, the OCR action will be applied to all image files in that folder.

Next Page ->

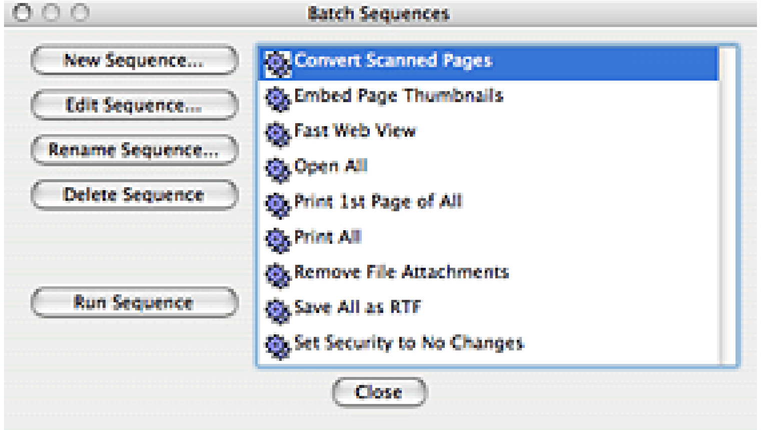
9. In the *Select output location* pop-up menu, specify where Acrobat should put the converted PDF files. The choices, at right, are reasonably self-explanatory.



10 Click OK to return to the *Batch Sequences* dialog box, which now displays your new sequence.

Running the Sequence

Having defined your sequence, you can run it any time you wish by selecting *Advanced>Batch Sequences* and then double-clicking your sequence in the resulting *Batch Sequences* dialog box.

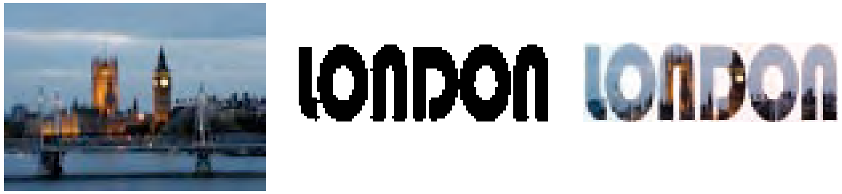


(You can also select the sequence in the list and click the *Run Sequence* button, of course.)

This batch sequence makes it much less tedious to convert a large number of scanned documents to text. Just place the scan files into your sequence's source folder, run the sequence in Acrobat, and stand back!

Return to Main Menu

Explicitly Masked Images, Part 2



Last June, we started a discussion of *explicitly masked images*, a PostScript Level 3 feature that allows an image dictionary to be accompanied by a mask dictionary, whose data indicate what part of the image should be printed.

In the first half of this topic, we discussed the basics of how explicitly masked images work in PostScript. We examined ImageType 3 image dictionaries and the contents of the image and mask dictionaries they contain.

The previous article used techniques appropriate only to small images and masks; in particular, the previous article's examples provided image and mask data as strings, which is completely inappropriate to real-world images.

This month, we fix this, seeing how to use actual scan data in an explicitly masked image.

Reading Assignment

This article assumes that you are have read three other Journal articles: the previous part of this series, in the June 2006 *Journal* and the two-part series on the SubFileDecode filter, in the June and July 2002 issues of the *Journal*. Read these articles before continuing here, since we have some relatively chewy stuff to discuss this week.

Next Page ->

ReusableStream Decode Filter

The problem we were left with last time was how to get the data for a real image to the *image* operator. In a regular image, we would usually read the data directly from the input stream, usually through a decoding filter of some flavor:

```
<<
     /ImageType 1
     ...
     /DataSource currentfile /ASCII85Decode filter
     ...
>> image
J/OJ!UDc!rBgo(r$KG#1>&d!4%M9@`.9pCQ.ue...
```

This technique won’t work for a masked image, because our ImageType 3 image needs to process two sets of data—the image and the mask—simultaneously. In the earlier issue’s examples, we solved this by supplying the image and mask data as strings, which can be processed as needed by the image mechanism. However, PostScript strings have a limit of 64K, which isn’t nearly large enough for a real-world image.

The solution to the problem is to make use of a filter introduced in Level 3: ReusableStreamDecode.

Next Page ->

How it works

ReusableStreamDecode is an interesting little guy. When you attached the filter to a file,

```
myFileObj /ReusableStreamDecode filter
```

the *filter* operator reads the entire contents of the file into VM and returns a filtered fileobject that represents that data in memory. This fileobject behaves in every way as though it were associated with a real, read-only file on the RIP's hard disk; in particular, you can read data from it and then reposition the file pointer so you can read the data again.

ReusableStreamDecode can take an optional *parameters* dictionary as an argument; this dictionary specifies another filter through which the source file's data should be passed before being placed in VM. For example, the following would pass the data taken from *myFileObj* through the ASCII85Decode filter before storing it in VM:

```
myFileObj
<< /Filter /ASCII85Decode >>
/ReusableStreamDecode filter
```

The parameter dictionary may also have a *DecodeParms* dictionary that holds parameters for the internally-applied filter. (In the above example, *ASCII85Decode* takes no parameters, so we don't need a *DecodeParms* entry.)

We are going to use the ReusableStreamDecode filter to store the image and mask data in VM as a pair of "virtual fileobjects." We will then hand these virtual file objects to the *image* operator as the data sources it needs.

Next Page ->

Reading from currentfile

We want ReusableStreamDecode read data from *currentfile* and place that data into VM. The problem is that the filter will read into memory the *entire* contents of the file to which it is attached; if we let it, the filter will consume the entire input stream, leaving no PostScript code for execution.

To prevent this happening—to ensure the ReusableStreamDecode filter reads only the image data—we will limit the scope of the filter with the SubFileDecode filter. (You *did* read the June and July 2002 *Journal* issues, didn't you? You 'll need it for the following.)

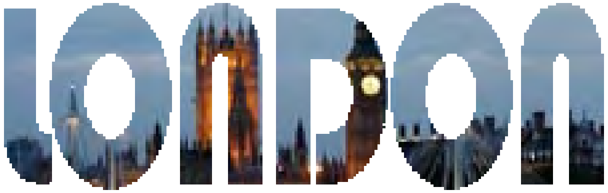
We shall have ReusableStreamDecode read its data through a SubFileDecode filter that defines logical end-of-file to be the stream of characters "**EOD**". ReusableStreamDecode will stop reading from the stream when those characters pass through it. The code will look something like this:

```
currentfile                          % Read data from currentfile
<<   /Filter /SubFileDecode          % Pass the data thru SubFileDecode
     /DecodeParms <<                 % ...with the following parameters:
          /EODString (**EOD**)       % ...EOF is marked by "**EOD**"
          /EODCount 0                % ...Ignore zero instances of **EOD**
     >>
>>
/ReusableStreamDecode filter         % The filter starts reading data
This is data that is being read into VM. This will be
image data in our final code.
**EOD**                              % Here's the logical end-of-file
% And now we are back to executable PostScript
showpage
```

Next Page ->

The Code

Here’s the annotated code for the “London” masked image. Refer to the May 2006 article for the details on each of the entries in the various image dictionaries.



Sample Files

As usual, this issue’s sample files are on the Acumen Training Resources page. Look for the file LondonMasked.zip.

```
/LondonImage  % Image data virtual file
     currentfile << /Filter /ASCII85Decode >> /ReusableStreamDecode
     filter
J/OJ!UDc!rBgo(r$KG#1>&d!4%M9@`.9pCQ.ue5=Hr`*?2T...
...

/LondonMask   % Mask data virtual file; this is the 1-bit text image
     currentfile << /Filter /ASCII85Decode >> /ReusableStreamDecode
     filter
J3Vsg3$]7K#D>EOLfdV7*=m^sU(KA\&\f’>6kSsVj=,d’O?o...
...

/TheImage     <<        % The image data’s image dictionary
     /ImageType    1         % Type 1 image
     /Width        393       % Width & height of the image in samples
     /Height       281
     /BitsPerComponent 8     % 8 bits each of r, g, & b
     /Decode [ 0 1 0 1 0 1 ]     % Map data into color
     /ImageMatrix [ 393 0 0 -281 0 281 ] % Map into 1-unit square
     /DataSource LondonImage /LZWDecode filter   % Data source
>> def
```

Next Page ->

```
/TheMask                    % The Mask data's image dictionary
<<   /ImageType 1
     /Width         393
     /Height        281
     /BitsPerComponent 1
     /Decode  [ 0 1 ]
     /ImageMatrix [ 393 0 0 -281 0 281 ]
     /DataSource LondonText /LZWDecode filter
>>def

/DeviceRGB setcolorspace    % We are printing a color image
100 100 translate           % Location of the image
393 281 scale               % Size of the image
<<                          % Now we print the masked image
     /ImageType 3           % Explicitly masked image
     /InterleaveType 3      % The mask is a separate image
     /DataDict TheImage     % The image dictionary
     /MaskDict TheMask      % The mask dictionary
>>
image                       % Call the image operator

showpage
```

Most of this we saw last time. What's new is the use of the ReusableStreamDecode "virtual file objects" as our sources of the image and mask data.

Next Page ->

One point of possible confusion when you read the code: the original image data was LZW compressed and then converted to ASCII85. ReusableStreamDecode read the data through the ASCII85Decode filter, which undid the ASCII85 encoding; the data written to memory was LZW compressed image data.

To use this compressed image data, we needed to attach the LZWDecode filter to our virtual file objects. Thus, for the image dictionary, we had

```
/DataSource LondonImage /LZWDecode filter
```

Using Less VM

One of the nice thing about the way PostScript does standard images is that the amount of VM has no bearing on the printing of the image. Since the image data is read directly from the input stream and is never stored wholly in VM, any PostScript RIP can render any image, regardless of the size of that image.

This is not true of explicitly masked images. Since the RIP needs to have both the image and the mask data available simultaneously, at least one of the sets of data need to be stored in VM, in our case as a virtual file created with ReusableStreamDecode.

In our previous example, we stored *both* sets of data in VM, mostly for the sake of clarity in the code. However, this means we consumed enough VM to store both sets of data.

We can be more efficient in our memory use by storing only one of the sets of data in VM and feeding the other set of data in-line with the PostScript code, as we usually do. For choice, we'd store the mask data in VM, since it's much smaller than the image data.

Next Page ->

Explicitly Masked Images, Part 2

Now our PostScript code looks like this (I’m going to abbreviate the parts that haven’t changed from last time):

The New Code

```
% We have removed the “LondonImage” file, since we’ll read the image
% data directly from the input stream.
/LondonMask        % Mask virtual file; this is the 1-bit text image
     currentfile << /Filter /ASCII85Decode >> /ReusableStreamDecode
     filter
J3Vsg3$]7K#D>EOLfdV7*=m^sU(KA\&\f’>6kSsVj=,d’O?o...
...

/TheImage     <<            % The image data’s image dictionary
     /ImageType    1        % Type 1 image
     /Width        393      % Width & height of the image in samples
     /Height       281
     /BitsPerComponent 8         % 8 bits each of r, g, & b
     /Decode [ 0 1 0 1 0 1 ]     % Map data into color
     /ImageMatrix [ 393 0 0 -281 0 281 ] % Map into 1-unit square
     /DataSource currentfile     % Our data source is now currentfile
          /ASCII85Decode filter /LZWDecode filter
>> def

/TheMask                    % The unchanged Mask image dictionary
<<   /ImageType 1
     ...
     /DataSource LondonMask /LZWDecode filter
>>def
```

Next Page ->

```
/DeviceRGB setcolorspace      % We are printing a color image
100 100 translate             % Location of the image
393 281 scale                 % Size of the image
<<                            % Now we print the masked image
     /ImageType 3             % Explicitly masked image
     /InterleaveType 3        % The mask is a separate image
     /DataDict TheImage       % The image dictionary
     /MaskDict TheMask        % The mask dictionary
>>
image                         % Call image, followed by image data
J/OJ!UDc!rBgo(r$KG#1>&d!4%M9...
...
...D5N-(L^3~>

showpage
```

Now, when *image* needs image data, it will read it one input-buffer-full at a time, with no impact on VM.

Masked images: Cool

Masked images are under-used in PostScript, which is a pity, since they are a common effect in graphic arts. The alternative to using ImageType 3 is to do the masking ahead of time—in Photoshop, perhaps—and then print the result as a standard ImageType 1 image. That has the benefit that the image will print on any PostScript Level 2 printer, where ImageType 3 is restricted to PostScript Level 3.

But it wouldn't impress your friends half as much.

Return to Main Menu

Schedule of Classes, October 2006-January 2007

Following are the dates of Acumen Training's upcoming PostScript and PDF Technical classes. Clicking on a class name below will take you to the description of that class on the Acumen training website.

These classes are taught in Orange County, California and on corporate sites world-wide. See the Acumen Training web site for more information.

Technical Classes

NEW-ISH!

PDF File Content and Structure 1		Nov 13-16	Jan 22-25
PDF File Content and Structure 2		Nov 28-Dec 1	
PostScript Foundations	Oct 30 - Nov 2		Sept 4–8
Variable Data PostScript			
Advanced PostScript			
PostScript for Support Engineers			Jan 8-12

Course Fee The PostScript and PDF classes cost $2,000 per student.

Registration Info

Acrobat Class Schedule

Regretfully, I have suspended teaching Acrobat classes.

However, watch this space!

Return to Main Menu

Contacting John Deubert at Acumen Training

For more information

For class descriptions, on-site arrangements or any other information about Acumen's classes:

Web site: http://www.acumentraining.com **email:** john@acumentraining.com

telephone: 949-248-1241

mail: 24996 Danamaple, Dana Point, CA 92629

Registering for Classes

To register for an Acumen Training class, contact John any of the following ways:

Register On-line: http://www.acumentraining.com/registration.html

email: registration@acumentraining.com

telephone: 949-248-1241

mail: 24996 Danamaple, Dana Point, CA 92629

Back issues

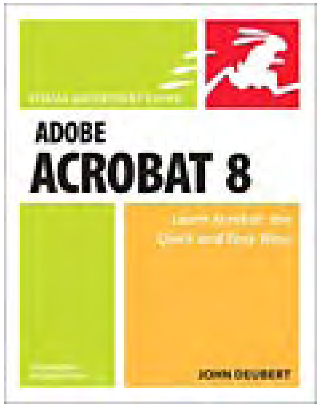
All issues of the *Acumen Journal* are available at the Acumen Training website:
http://www.acumenjournal.com/AcumenJournal.html

Return to First Page

What's New at Acumen Training?

Acrobat 8 Visual Quickstart

I'm finishing a new book: *Acrobat 8 Visual Quickstart Guide,* published by Peachpit Press, due to be on store shelves in November. It's a beginner's book on how to do all the things you most commonly will want to do with Acrobat 8, covering the gamut from opening and navigating PDF files through digitally signing documents to converting paper documents to PDF. (Well, alright; we covered that last in this issue of the *Journal.)*



Here are the chapter titles:

1. Starting Acrobat
2. Viewing PDF Documents
3. Saving and Printing
4. Making PDF Files
5. Adding Comments
6. Reading Commented Docs
7. Conducting Group Reviews
8. Manipulating PDF Pages
9. Adding & Changing Text & Graphics
10. Adding Basic Navigation Features
11. Acrobat Presentations
12. Orgainizing Sets of Documents
13. Creating Acrobat Forms
14. Password Protection
15. Digital Signatures
16. Converting Paper Documents

You will want to buy several of these, just so you can admire their spines on your bookshelf.

Return to First Page

Journal Feedback

If you have any comments regarding the *Acumen Journal,* please let me know. In particular, I am looking for three types of information:

Comments on usefulness. Does the Journal provide you with worthwhile information? Was it well written and understandable? Do you like it, hate it? Did it make you wonder if something was lost in the translation into English?

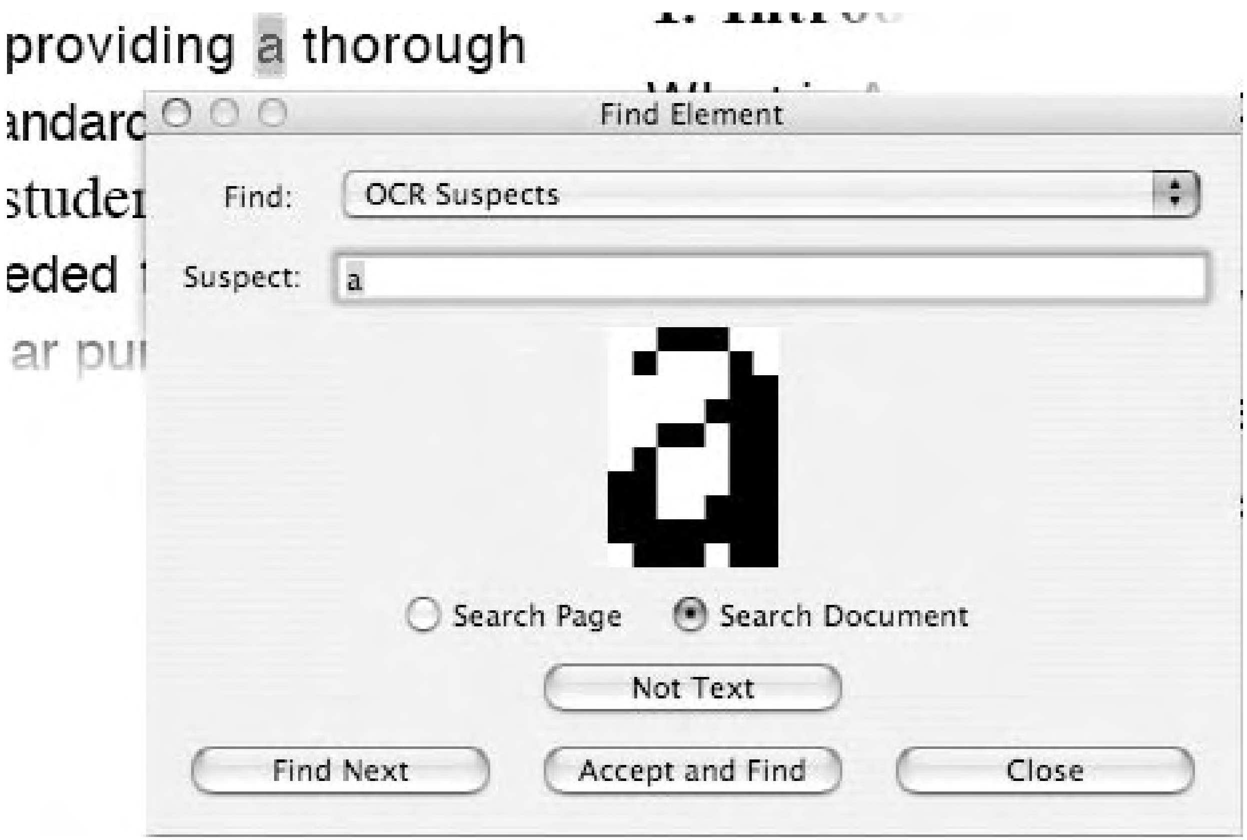
Suggestions for articles. Each Journal issue contains one article each on PostScript and Acrobat. What topics would you like me to write about?

Questions and Answers. Do you have any questions about Acrobat, PDF, or PostScript? Feel free to email me about. I'll answer your question if I can. (If enough people ask the same question, I can turn it into a Journal article.)

Please send any comments, questions, or problems to:

journal@acumentraining.com

Return to Menu

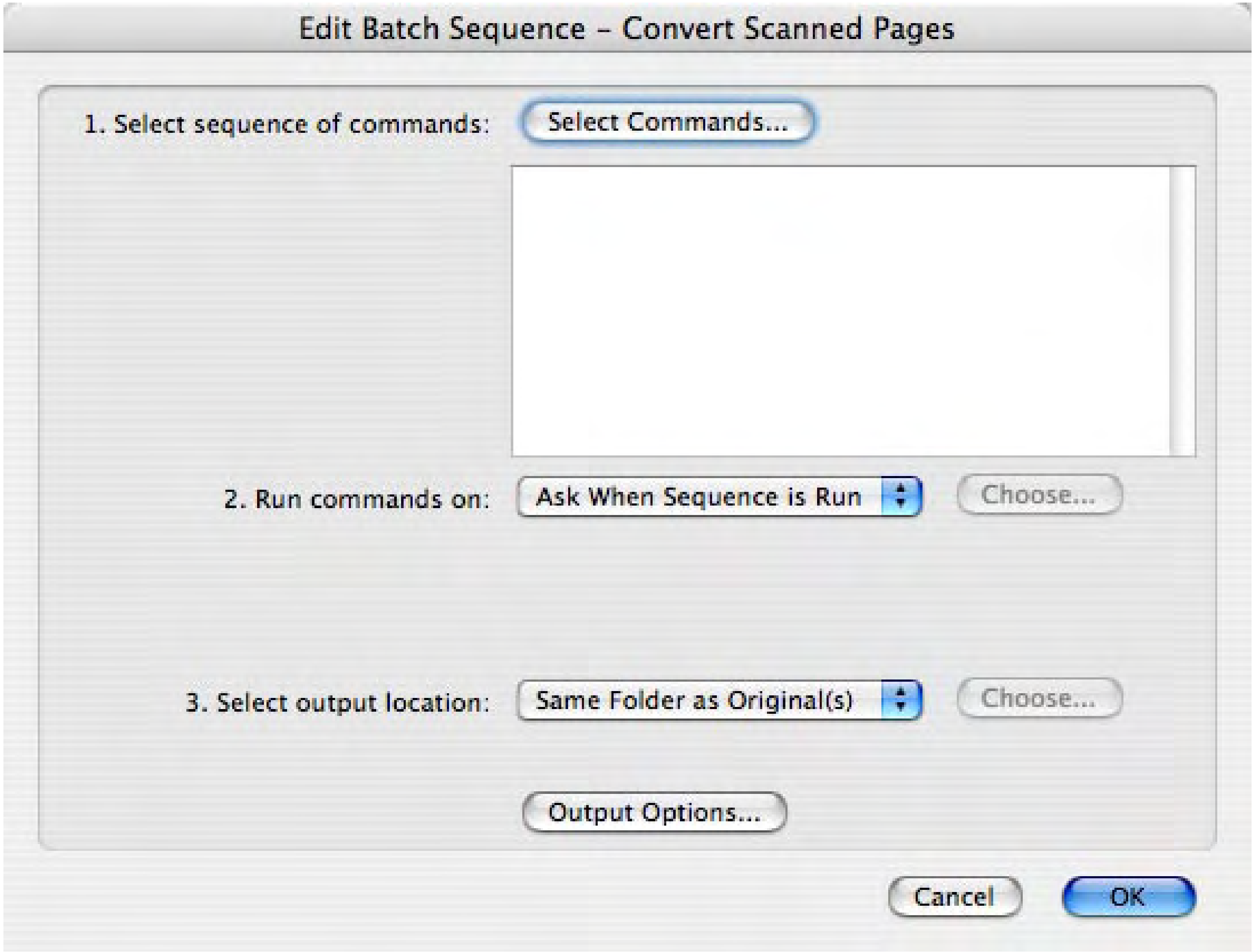


Find: OCR Suspects

Suspect: a

○ Search Page ● Search Document

Edit Batch Sequence Dialog Box



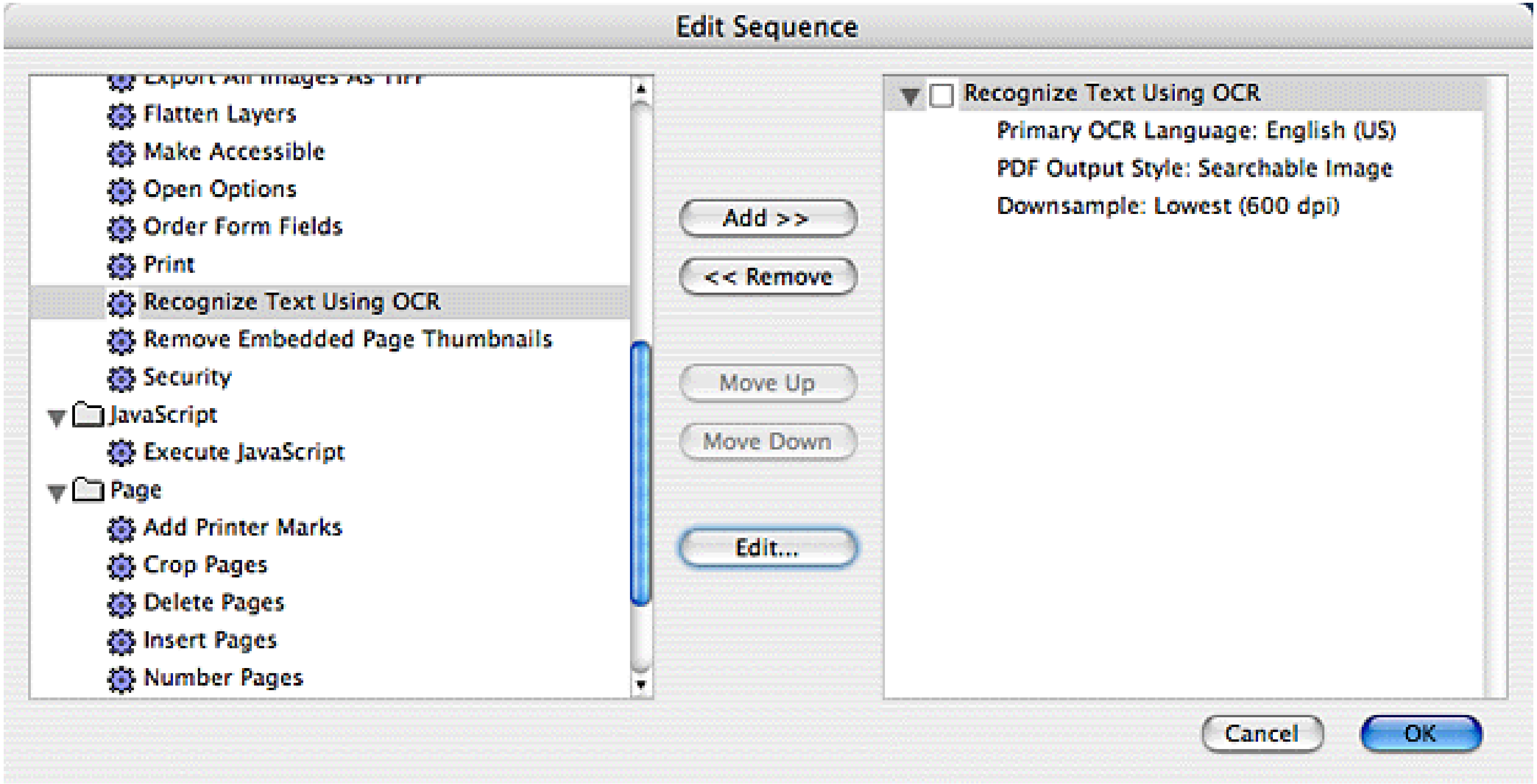
Edit Batch Sequence – Convert Scanned Pages

1. Select sequence of commands:

2. Run commands on: Ask When Sequence is Run

3. Select output location: Same Folder as Original(s)

Edit Sequence Dialog Box



☐ Recognize Text Using OCR

Primary OCR Language: English (US)

PDF Output Style: Searchable Image

Downsample: Lowest (600 dpi)