Table of Contents

Acumen Training

Acrobat Signatures

One of Adobe's goals when they released Acrobat 4, several years ago, was to make it possible for a PDF file to be a legally binding document. To this end, they included support for digital signatures in Acrobat 4.

Eventually, they hoped that contracts and other legal documents could be signed and stored electronically, without a piece of paper signed in ink.

This month and next, we are going to examine the workings of Acrobat's digital signature mechanism. In this issue, we'll look at how to sign a PDF document. Next month, we'll see how to send the signed document to another person and how that person can verify that your signature is authentic.

Next Page ->

Background

What's a Signature?

The purpose of a signature, whether digital or ink, is to signal that a particular person has seen, agreed to, or otherwise acknowledged the contents of a document. For this to be the case, the following must be true of the signature and the document:

- It must be possible to establish that a person's signature was, indeed, placed on the document by that person.
- It must be possible to verify the document has not been changed since it was signed.

Digital Signatures

The default Acrobat signature mechanism, *Adobe Self-Sign Security,* fulfills both of these requirements. A signature placed on a PDF page is a set of encrypted data embedded in the PDF file. This data allows a recipient's copy of Acrobat to determine:

- The signature has a digital "footprint" associated with a particular signer.
- Whether the the document was changed since it was signed. If so, the document can be made to revert to the state it had when it was signed.

The Acrobat signature mechanism is modular; third parties can write Acrobat plug-ins that implement a particular signature mechanism. In this article, we'll describe Adobe's *Self-Sign Security,* the signature mechanism that ships with Acrobat.

Next Page ->

Self-Sign Security: An Overview

In brief, Adobe's Self-Sign Security works as follows:

Create User Profile

Before you can use the Self-Sign Signature mechanism, you must create a password-protected user profile. You will use this password whenever you sign a PDF document.

Signing a Document

There are two steps to signing a PDF document:

Log in

First you must log into the Self-Sign Security system. You supply your password at this point so the system knows it's really you.

Sign the document

You sign the document by clicking on the Acrobat *Signature* tool (the cursor will turn to a crosshair) and then dragging out a rectangle on the page you're signing. Acrobat will ask you some questions and ask you to verify your signature. Your signature will be added to the page in the area you dragged out.

Once signed, a PDF file becomes an "append-only" document. Acrobat will always be able to determine whether the document was modified after you signed it and to revert to the earlier, signed version of the PDF file.

Next Page ->

Sending the Document

Having signed the document, you'll need to send it on to whomever wanted your signature. It is not enough to send the signed PDF file; how is the recipient supposed to know that the signature is validly yours?

Trusted Certificate

You must create and send to the recipient (just once) a certificate file that contains encrypted information identifying your signature. The recipient installs this as a *Trusted Certificate,* "trusted" in the sense that he or she is satisfied that it was, indeed, you who sent the certificate. (Perhaps they'll call you to see if you actually sent them that certificate file.)

Once your certificate file is installed in the recipient's copy of Acrobat, that copy of Acrobat will always be able to recognize your signatures as valid.

Receiving the Document

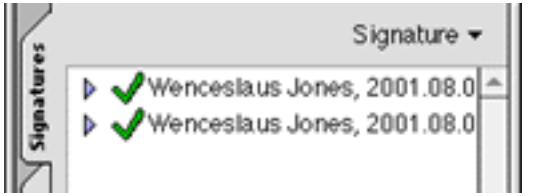
The person who receives the document must do the following:

Install Trusted Certificate

If necessary, install the Certificate file you sent into Acrobat as a Trusted Certificate.

Verify Signature

Upon opening the signed document and exposing the *Signature* panel (at right), select *Verify All Signatures* from the fly-out menu. Acrobat will confirm that your signature is valid.



Next Page ->

Step by Step: Signing a Document

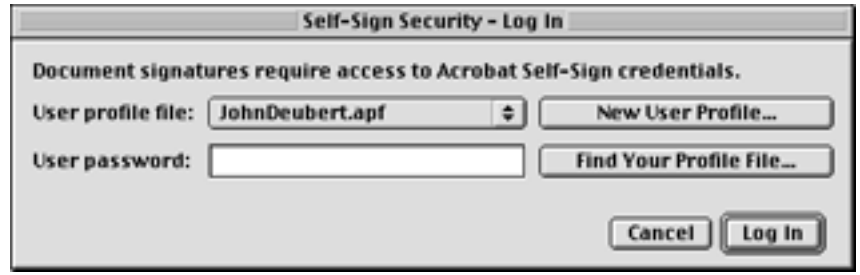
Creating a User Profile

The first step in signing a document is something you need do only once: create a User Profile by which the signature mechanism will know you.

Go to "Log-In"

Start by going to *Tools>Self-Sign Security>Log In…* You will be faced with the User Log-In dialog box.

If you already had a User Profile on this computer, you could simply select it from the pop-up menu.



Click "New User Profile…"

In this case, we shall click on the *New User Profile…* button.

We are now looking at the *Create New User* dialog box, that collects information about us for the signature mechanism.

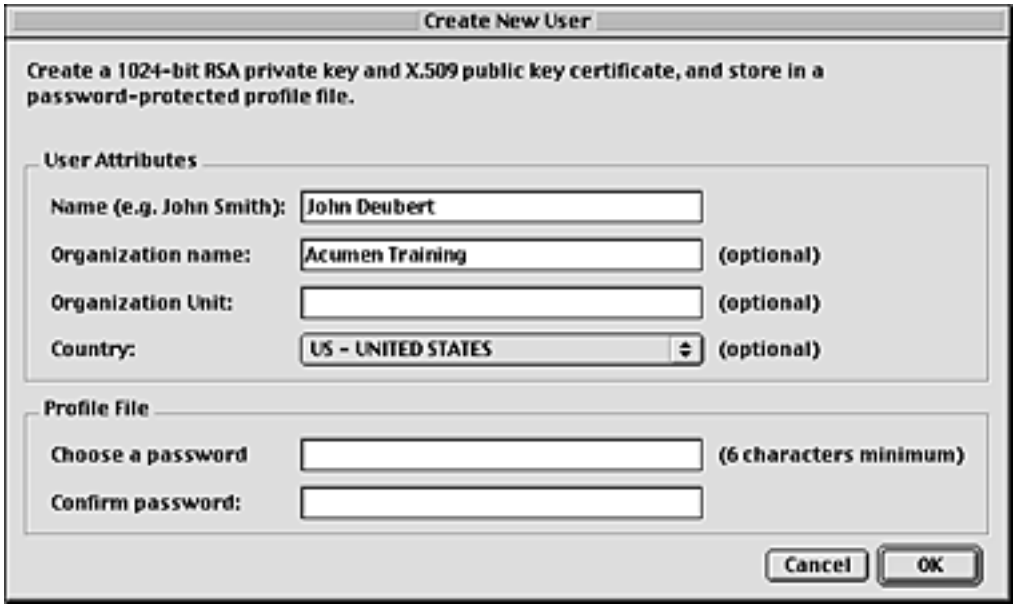
Next Page ->

Fill out the form

In the *Create New User* dialog box, you tell Acrobat who you are and choose a password.

Don't forget this password. You'll need it every time you log in and every time you sign a PDF document.

That's it. Click the *OK* button and you will be asked to save your User Profile file on your computer's hard disk.



You will now have created a user profile and be logged in.

Logging In

From now on, when you select the *Log In…* menu item, you can simply select your User Profile from the pop-up menu.

Next Page ->

Acumen Journal: Acrobat User

Signing the Document

To sign the document, you may either click on the Signature tool or select *Tools>Digital Signatures>Sign Document…*

Drag a rectangle

The cursor will turn to a crosshair. Click and drag out a rectangular area that will be occupied by your signature.

Reenter your password

You will then be faced with the *Sign Document* dialog box.

Here you need to supply your password again. Note that you can also supply other information, such as why you are signing the document, your phone number, etc.

Next Page ->

Save the document

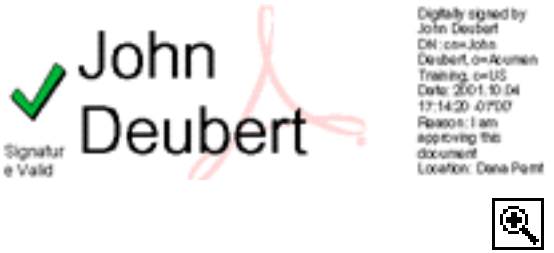
To finish your signature, you must click on either the *Save* or *Save As* button.

The document will be saved in a special "Append only" state. Any future changes to the document will be internally appended to the document's present contents.

It will be possible for Acrobat to determine that the document has been changed since you signed it.

It will also be possible for Acrobat to return the document to the state it had when you signed it.

When you return from this dialog box, your signature will appear on the page within the rectangle that you specified with the signature tool.

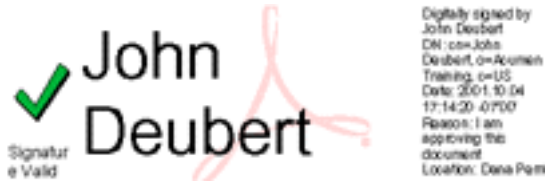


Next Page ->

Signature Appearance

The actual signature embedded in the PDF document is a little wad of data placed into the PDF file. The representation you see on the PDF page is a just visual indication that the page has been signed.

The default appearance of a signature consists of your name in large print and some other information in a smaller point size.



You can change the appearance of your signature by modifying some of the Self-Sign Security User Settings:

- You can change the information that is reported in the small type.
- You can replace your name with a picture of your devising.

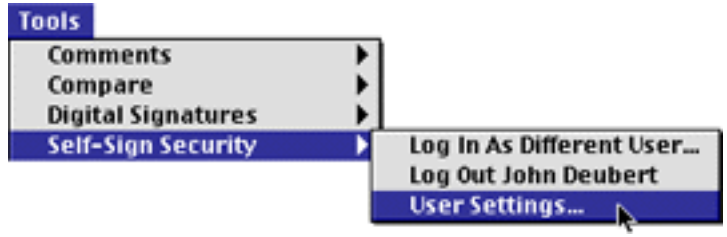
  At right, for example, is my signature placed on the page with a vaguely Kilroy-was-here drawing instead of my name. (No, I'm *not* much of an artist.)

Let's see how we do this.

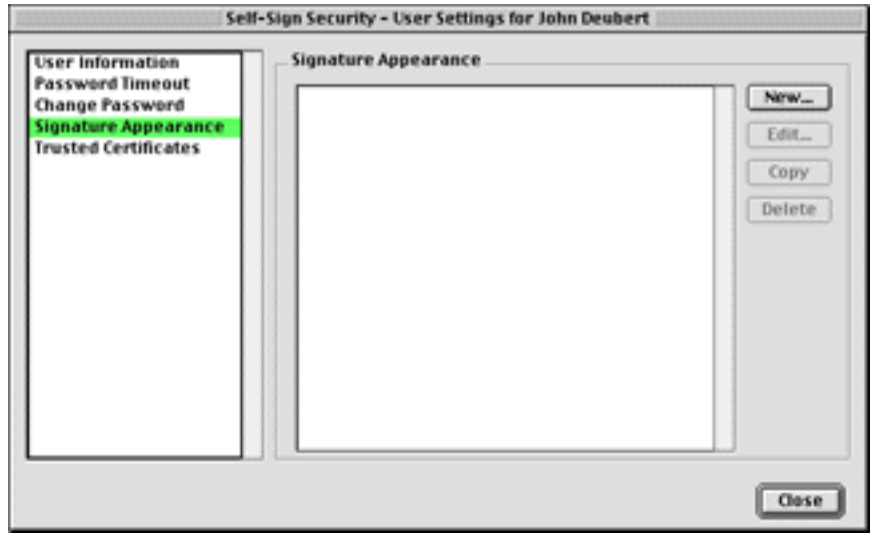
Next Page ->

User Settings

You change the appearance of your signature by selecting *Tools>Self-Sign Security>User Settings…* Acrobat will present you with the *User Settings* dialog box.



This dialog box allows you to specify a number of settings that affect the behavior of the Self-Sign Security mechanism.



Click on Signature Appearance

Clicking on *Signature Appearance* in the left panel gives you access to controls that let you define a named set of appearance settings.

Click on the "New…" Button

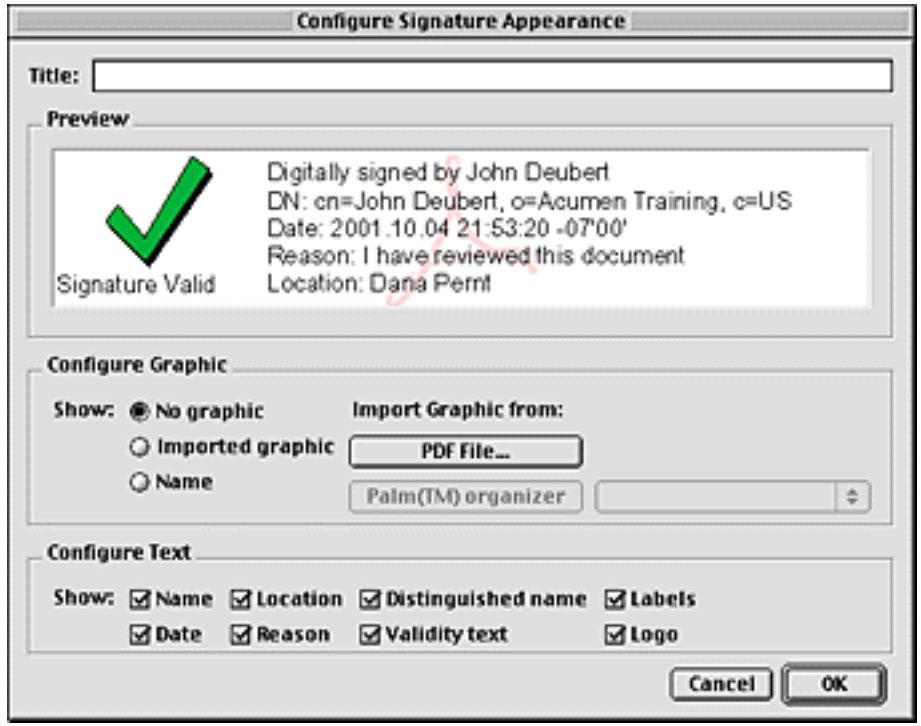
Clicking on the *New…* button gives you access to the *Configure Signature Appearance* dialog box (next page).

Next Page ->

Configure Appearance

This dialog box allows you to specify how your signature is displayed on the PDF page.

You are actually creating a named collection of settings that indicate what graphic, if any, and what text information, if any, should represent your signature on the page.



Title

Type into the *Title* field the name you want to give this set.

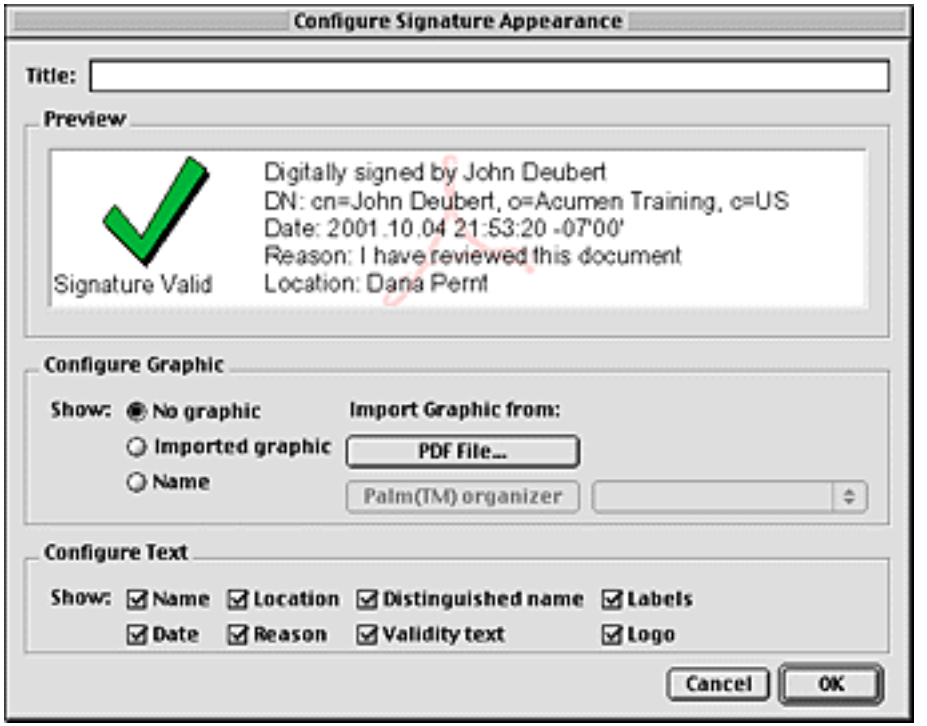
Configure Text

At the bottom of the dialog box is a set of check boxes indicating what information should be included on the page. Select whichever ones you want.

The large Preview will change to reflect your selection.

Next Page ->

Configure Graphic



In the middle of the dialog box are a set of radio buttons that specify what graphic, if any, should be included in the signature.

Clicking the *PDF File...* button allows you to pick any single-page PDF file whose contents will be used as the picture for your signature.

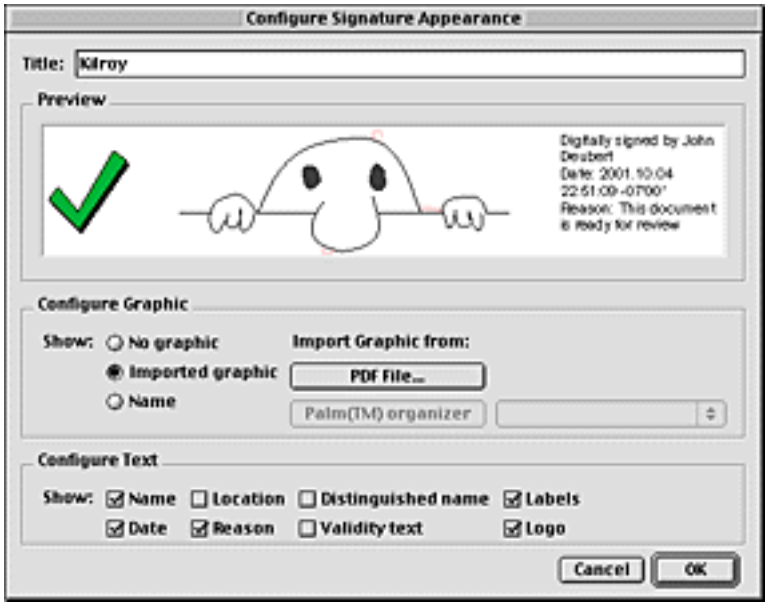
This graphic may be line art, a scanned image, or anything you wish, as long as it has been saved as a PDF file. Don't worry about cropping the page to the border of your artwork; when you select the PDF file, Acrobat will ignore white space surrounding the graphical content.

You may also create this graphic with any software tool you wish. Just save the final art as a PDF file and you can select it as a signature graphic.

Next Page ->

Kilroy, for example

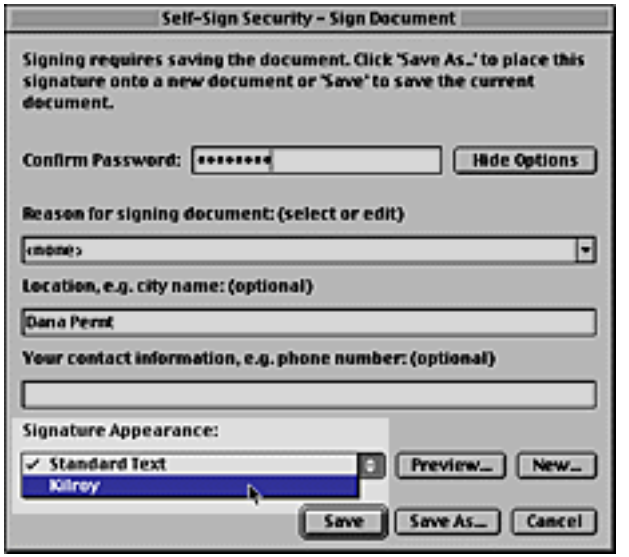
I created Kilroy in Adobe Illustrator® and saved it as a PDF file. I then imported it as a signature graphic in Acrobat.



Now, when I sign a PDF document, I can select *Kilroy* from the *Signature Appearance* menu in the *Sign Document* dialog box. (This is the dialog box I get when I'm signing a PDF file.)

For verisimilatude, you could scan a sample of your own pen-on-paper signature and use that as the graphic for your digital signature.

Fun is where you find it!



Next Page ->>

Next Month...

So now we know how to sign an Acrobat document. As you can see, it's really pretty easy.

Next month, we'll look at the other half of the picture: how to send a signed document to someone else and what they have to do to verify your signature is valid.

Return to Main Menu

Centering Text Vertically

It comes up all the time in PostScript programming: you want to print text exactly centered in a rectangle.

Horizontal centering is pretty easy; PostScript supplies a *stringwidth* operator that you can use to figure that out. But there is no operator in PostScript that gives you the information you need to center the text vertically.

This month, we'll discuss how to vertically center text. This will give us a chance to use some PostScript operators that otherwise tend to gather dust: *pathbbox and flattenpath.*

Our goal will be to write a *centerxyshow* procedure that takes a string and prints it centered about the current point.

Next Page ->

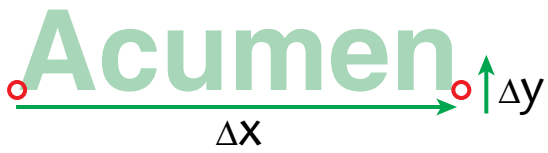
The Horizontal Case

Let's start with the simple case: horizontal centering a string about the current point.

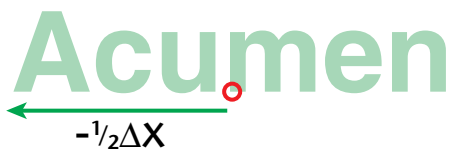
The principle is straightforward: measure the physical width of the string, move back half that distance, and print the string. This is made relatively easy by PostScript *stringwidth* operator. (This will be review, if you've taken the PostScript Foundations class.)

stringwidth `(str) stringwidth => Δx Δy`

The *stringwidth* operator takes a string from the stack and returns the *x* and *y* offset that would be applied to the current point if you were to print the string.



The *y* offset will be zero for horizontally printed text. The *x* offset is the number we need for centering: the width of the text on the page. We want to move the current point to the left by half this amount before printing the string.



Next Page ->

centershow

So here is the definition and use of a *centershow* procedure that takes a string from the stack and prints it centered horizontally about the current point:

```
/centershow
{    dup                  % Save a copy of the string for later
     stringwidth pop      % Get the current point offsets & discard Δy
     -2 div 0             % Halve Δx (& reverse its sign); push 0 on stack
     rmoveto show
} bind def

/Helvetica 24 selectfont
300 600 moveto
(Acumen Training) centershow
```

Nothing too hard here.

Let's see about doing the same thing vertically.

Next Page ->

The Vertical Case

In centering text vertically about the current point, the general procedure is similar to the horizontal case: find the vertical extent of the text, move down by half that distance and print the string. We're hampered in this by the fact that PostScript provides no operator that returns the height of a string. We need to do this ourselves.

The best way to determine a string's height is to convert the character outlines to a path and then find the bounding box of that path. Happily, PostScript *does* provide operators that do this.

We need to use two operators: *charpath* and *pathbbox.* As we'll see, we may also want to use a third operator, *flattenpath.*

charpath `(str) bool charpath => ---`

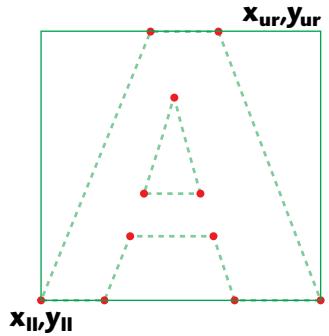
This operator takes a string and a boolean from the stack and adds the outlines of the string's characters to the current path.

You may use either *true* or *false* for the boolean value; it is there for the benefit of stroked fonts, which pretty much don't exist anymore. Use whichever boolean is your favorite.

(If you've taken the PostScript Foundations or PostScript for Support Engineers class, you'll find a more detailed description of the boolean's purpose in your student notes.)

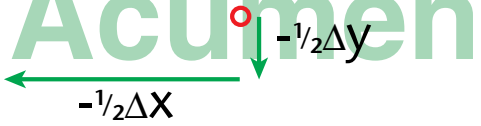
Next Page ->

pathbbox `- - - pathbbox => llx lly urx ury`



This operator returns the coordinates of the lower-left and upper-right corners of the *path bounding box,* that is, the rectangle that exactly encloses the points in the current path.

This gives us the information we need to do our centering calculations. To center a string about the current point, both horizontally and vertically, we need to move the current point to the left and down by half the differences in the *x* and *y* coordinates returned by this operator.



To center our text, we can do a *charpath* on our string, execute *pathbbox,* and use the resulting numbers to calculate the *x* and *y* offsets.

A first pass at our *centerxyshow* procedure is on the next page:

Next Page ->

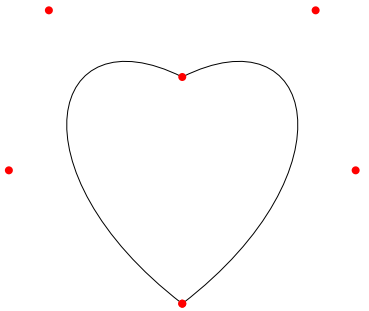
centerxyshow, First Pass

```
/centerxyshow  % (str) => ---
{    dup
     gsave                        % So we can undo the charpath path elements
     false charpath pathbbox      % result:  xll yll xur yur
     3 -1 roll                    % =>  xll xur yur yll
     sub -2 div                   % =>  xll xur -1/2Δy
     3 1 roll                     % =>  -1/2Δy xll xur
     sub 2 div exch               % =>  -1/2Δx -1/2Δy
     grestore                     % Go back to the original current point
     rmoveto show                 % Offset the current point and show
} bind def
```

A Problem

One problem with our *centerxyshow* so far is that it bases its centering calculations on the bounding box of *all* the points in the path, including bezier control points.

Consider the heart at right, constructed from two bezier curves. Bezier control points do not lie on the curve, itself; in this particular case, they lie quite far off the path. The *pathbbox* operator's return values reflect all of these points, which is not what we want.



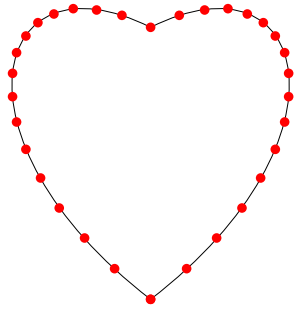
For centering text, we want a path that reflects the outline of the *visible* characters, excluding the bezier control points that went into constructing them.

We can obtain such a path with the *flattenpath* operator.

Next Page ->

flattenpath

As you may remember from your PostScript class, at rendering time (*i.e.,* when we execute *stroke* or *fill),* PostScript replaces all bezier curves in the current path with a series of straight lines that approximate those curves.



The *flattenpath* operator replaces all bezier curves in the current path with a series of *lineto's* that correspond to the straight-line approximation of those curves. For example, when applied to our heart, *flattenpath* converts the two bezier curves to the series of *moveto* points diagrammed at right.

Since character outlines make extensive use of bezier curves, the control points in that outline could, in principle, introduce a significant inaccuracy into our centering calculations. To be very good about it, our *centerxyshow* should do a *flattenpath* before executing *pathbbox.*

Our final definition of *centerxyshow* now looks like this:

Next Page ->

centerxyshow, Final

```
/centerxyshow  % (str) => ---
{    dup
     gsave                      % So we can undo the charpath path elements
     false charpath
     flattenpath pathbbox       % result:  xll yll xur yur
     3 -1 roll                  % =>  xll xur yur yll
     sub -2 div                 % =>  xll xur -1/2Δy
     3 1 roll                   % =>  -1/2Δy xll xur
     sub 2 div exch             % =>  -1/2Δx -1/2Δy
     grestore                   % Go back to the original current point
     rmoveto show               % Offset the current point and show
} bind def
```

This should eliminate the possibility of any inaccuracies due to bezier control points in the character definitions.

Next Page ->

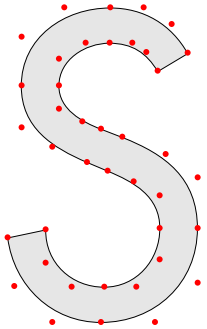
Acumen Journal: PostScript Tech

Two Final Points

A couple of points of interest, in closing:

Control Point Inaccuracies

We introduced *flattenpath* into our *centerxyshow* to ensure that bezier control points don't introduce any inaccuracies into our centering calculations. In practice, bezier control points don't seem to be a problem with character shapes. I've not been able to find any character definitions whose bezier control points extend beyond the actual character outline's bounding box.



I still include *flattenpath* when I do these calculations, but it's fixing a very rare problem.

Homework: *ShowPathPoints*

The illustrations showing the control points for the heart and for the character above are real. That is, these are hand-written EPS files that define a procedure called *ShowPathPoints.* This procedure runs through the current path and draws a little filled circle centered on each vertex and control point in the path.

As a challenge, in all your plentiful spare time, write your own *ShowPathPoints* procedure. I'll show you mine next month.

If you want a hint, move your cursor over **this text**.

See you next month.

Return to Main Menu

Schedule of Classes, Oct 2001 - Dec 2001

Following are the dates and locations of Acumen Training's PostScript and Acrobat classes. Clicking on a class name below will take you to the description of that class on the Acumen training website. (September has been completely consumed by on-site classes, so there are no Orange County classes this month.)

The PostScript classes are taught in Orange County, California.

PostScript Classes

PostScript Foundations	Orange Co., CA	October 15 - 19	Orange Co., CA	December 10 - 14
Advanced PostScript	Orange Co., CA	November 5 - 9		
PostScript for Support Engineers	None until first quarter, 2002			
Jaws Development	Orange Co., CA	November 27 - 30		

For more classes, go to www.acumentraining.com/schedule.html

PostScript Course Fees

PostScript classes cost $2,000 per student.
These classes may also be taught on your organization's site.

Registration →
Acrobat Classes →

Acrobat Class Schedule

On-Site Only These classes are taught only on corporate sites. If you have an interest in any of these classes for your group, please see the Acumen Training website regarding arranging an on-site class.

Acrobat Essentials This class teaches the student how to make perfect PDF files. It includes complete coverage of the meaning and proper settings of all of the Distiller Job Options.

Interactive Acrobat Here we show you how to add bookmarks, links, buttons, sounds, movies, form fields, and other interactive features to an Acrobat file.

Creating Acrobat Forms This class shows you how to make interactive forms in Adobe Acrobat. It steps you through creating the form, posting form contents to a server, and everything else you need to create a working PDF form.

Troubleshooting with Enfocus' PitStop This class shows the student how to use all of the capabilities of this popular editing and preflight software.

Back to PostScript Classes

Return to First Page

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: 25142 Danalaurel, Dana Point, CA 92629

Registering for Classes

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

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

email: registration@acumentraining.com

telephone: 949-248-1241

mail: 25142 Danalaurel, Dana Point, CA 92629

Back issues

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

Return to First Page

Addendum to Last Month

Fixing Acrobat 5 Thin Lines

Several people have pointed out a couple of applications that create Acrobat files whose hairlines cannot be fixed with the methods I described in last month's *Journal.*

FrameMaker

In particular, hairlines in PDF files produced by FrameMaker cannot be fixed. Many of the stroked lines drawn by FrameMaker are actually very thin rectangles. Lines in FrameMaker output cannot be fixed because, properly speaking, they aren't lines at all.

Offhand, I don't know why they do their lines this way. I can see no obvious advantage over just printing a real line.

Unfortunately, I also can see no fix for the inaccurate widths of lines created this way. Sorry.

Any out there have any suggestions?

Return to First Page

Journal Feedback

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

Comments on usefulness. Does the Journal provide you with worthwhile information? Was it well written and understandable? Did you like it, hate it, or did it make you want to eat brussels sprouts? How could we make it better? Do you like the PDF format?

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

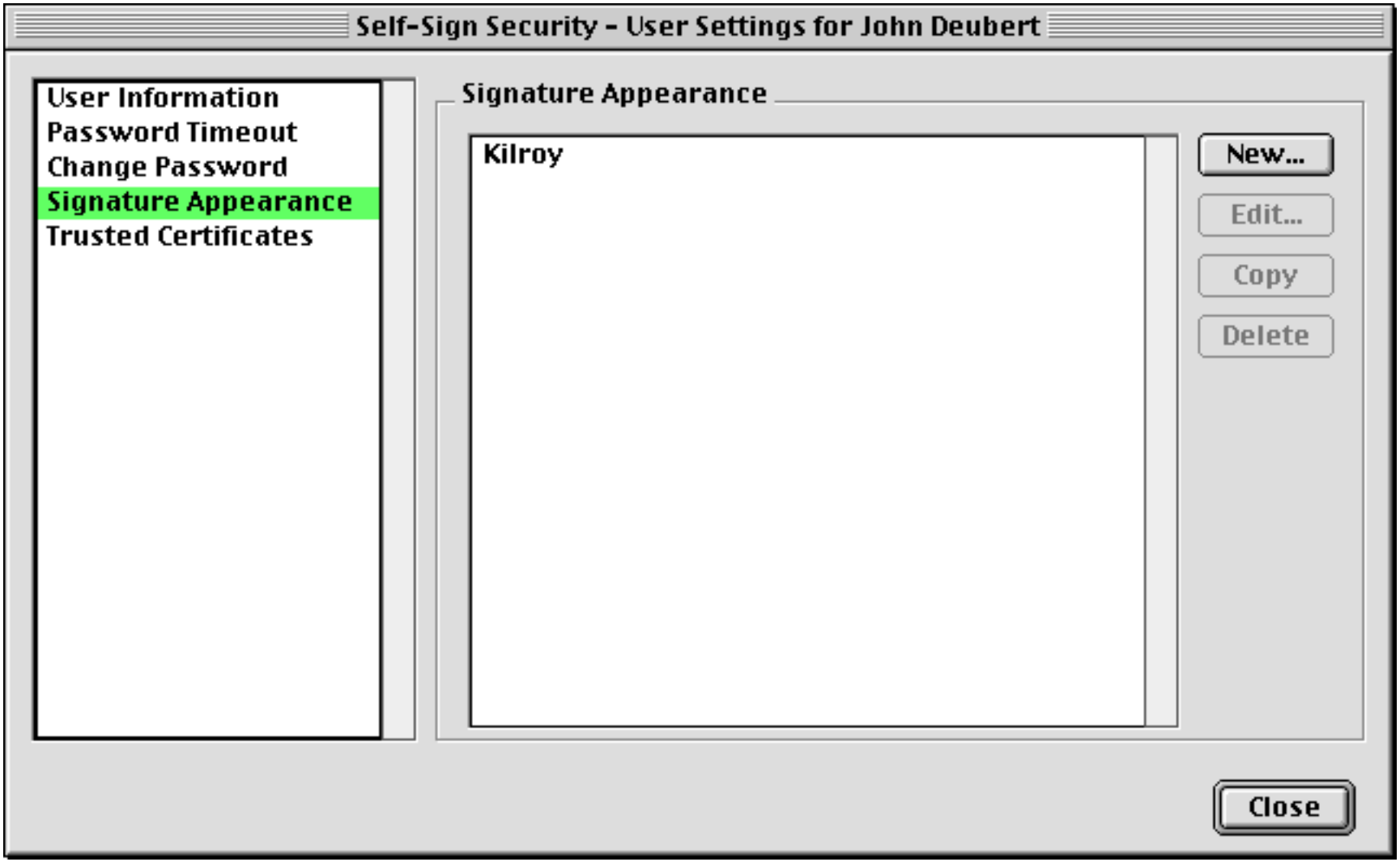
Questions and Answers. We are planning a Q&A section for future issues. Do you have any questions about Acrobat, PDF or PostScript?

Please send any comments, questions, or problems to:

journal@acumentraining.com

Return to Menu

Digitally signed by John Deubert
DN: cn=John Deubert, o=Acumen Training, c=US
Date: 2001.10.04 17:14:20 -07'00'
Reason: I am approving this document
Location: Dana Pemt


Self-Sign Security - User Settings for John Deubert
User Information
Password Timeout
Change Password
Signature Appearance
Trusted Certificates
Signature Appearance
Kilroy
New...
Edit...
Copy
Delete
Close