

Table of Contents

[The Acrobat User](#)

Creating Your Own Rubber Stamp Annotations

Among the Acrobat's annotations types is "Rubber Stamp," which lets you place a picture on the page as an annotation. This month we'll see how to add our own pictures to the predefined stamps supplied by Adobe.

[PostScript Tech](#)

Adding Characters to a Type 1 Font

This month we answer a common question: how do you add a character to a Type 1 font?

[Class Schedule](#)

June-July-August

Where and when are we teaching our Acrobat and PostScript classes? See here!

[What's New?](#)

More details on the Technical PDF class

News about the Technical PDF class currently in development.

[Contacting Acumen](#)

Telephone number, email address, postal address, all the ways of getting to Acumen.

[Journal feedback: suggestions for articles, questions, etc.](#)

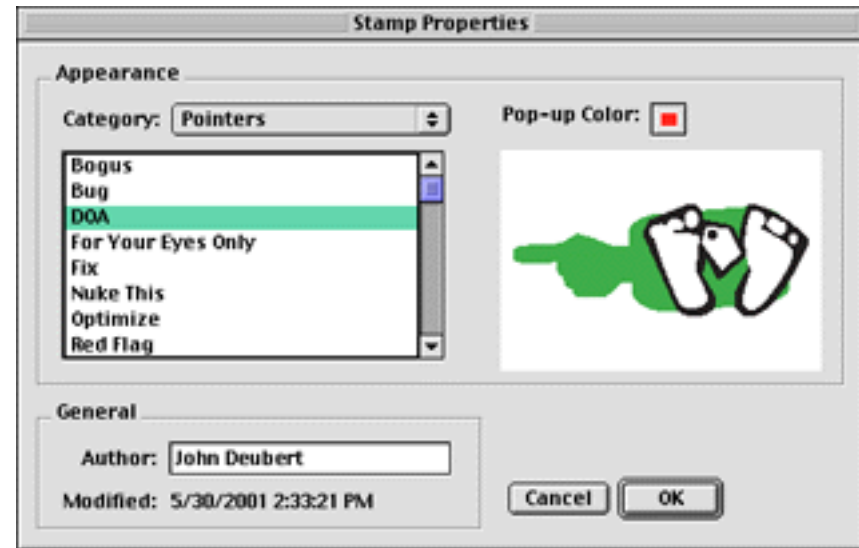
Creating Your Own Rubber Stamp Annotations

FINAL

One of the more fun annotations available in Adobe Acrobat is the "Rubber Stamp." A rubber stamp annotation consists of a resizable picture to which you may attach a note. (The picture at left is one of these; double click on it and you can see an attached note.)

You choose the picture you want to use from a list presented to you by Acrobat. Adobe supplies a fair number of predefined pictures, nicely categorized into names, pointers, etc.

It is fairly easy to add your own pictures to this list. This is what we'll be discussing this month: how to add your own custom rubber stamps to Acrobat.



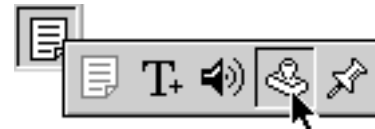
[Next Page ->](#)

A Review: Placing a Rubber Stamp

Before we discuss adding our own rubber stamp annotation to Acrobat, let's review how you place such an annotation on the page.

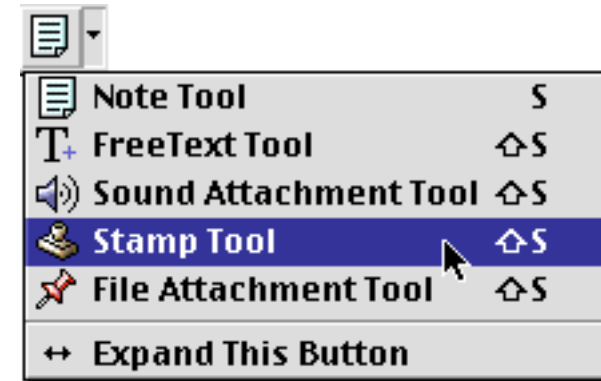
The Rubber Stamp Tool

The Rubber Stamp tool is one of the "Special" annotations, grouped in the Acrobat toolbar with the sticky note and the sound annotation. This is the topmost annotation group in Acrobat 4 and the leftmost in Acrobat 5.



Rubber Stamp, Acrobat 4

When you select the Rubber Stamp tool, your cursor turns into a crosshair with which you may drag out a rectangular area where your annotation will go.



Rubber Stamp, Acrobat 5

[Next Page ->](#)

Rubber Stamp Properties

Once you have dragged out the location for your annotation, Acrobat 4 and 5 do different things.

Acrobat 4 takes you immediately to the Stamp Properties dialog box (below).

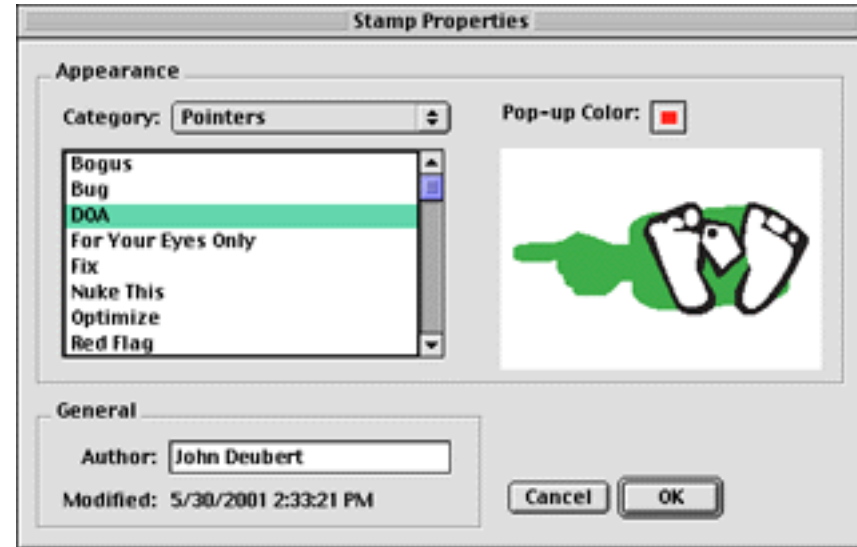
Acrobat 5 simply places the currently-selected rubber stamp immediately on the page. To get to the Rubber Stamp properties dialog box, you will need to click once on the new annotation (which selects it) and then select *Properties* in the *Edit* menu.



Either way, you will now be staring at the Stamp Properties dialog box.

This dialog box allows you to select a picture for your rubber stamp. The pictures are grouped into categories selectable from the *Category* pop-up menu.

We're going to add a couple of pictures to this list and place them in their own category.



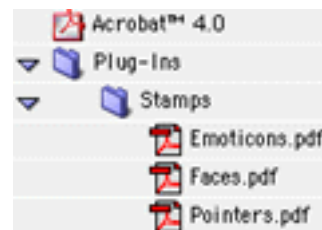
[Next Page ->](#)

Where does Acrobat get these stamps?

The stamps that Acrobat offers us actually reside in a set of PDF files. Each PDF file becomes a category in the pop-up menu; each page within the PDF file becomes a stamp in that category.

These PDF “source files” reside in slightly different places in Acrobat 4 and 5.

In Acrobat 4, the files reside in a “Stamps” folder in the Plug-ins folder in the Acrobat folder. (That is, *Acrobat>Plug-ins>Stamps*.)



In Acrobat 5, they reside in *Acrobat>Plug-ins>Annotations>Stamps*.



The “Title” property of each these PDF files (among the Acrobat Document Properties) becomes the name of the Category in the Acrobat pop-up menu.

Each page in these PDF files is marked as a “template page.” Template pages are a mechanism by which JavaScripts can add pages to a PDF file on the fly. The Rubber Stamp annotation uses page templates to provide names for the individual stamps in this category.

Let’s see how this works.

[Next Page ->](#)

Creating a Rubber Stamp

To create our own rubber stamp and category, we need to do the following:

1. Prepare the artwork for our stamps.
2. Convert each stamp artwork into a PDF file.
3. Assemble all of the stamp PDFs into a single file.
4. Set the Title property of the file to the name we want for our new category.
5. Convert each page in the PDF file into a page template whose name becomes the stamp's name.
6. Move the PDF file containing our custom stamps into the *Stamps* folder.

Our Example

Let's create an "Emoticon" category containing two stamps: a smiley face and a wink (as at left).



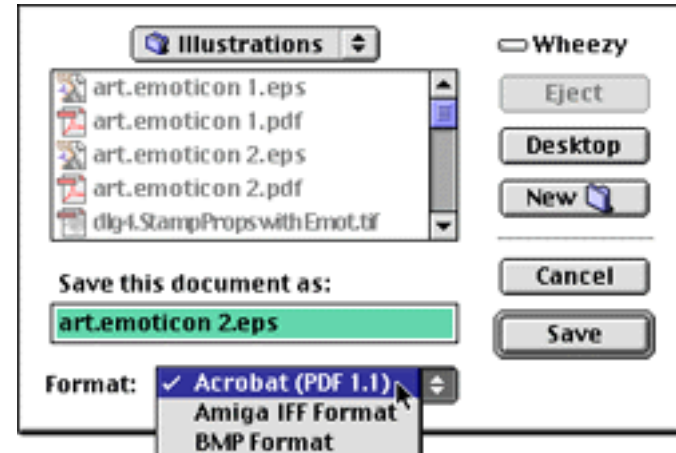
By the way, all of the files involved in preparing these stamps (two Illustrator 6.0 files and a PDF file) are available from the Acumen Training website. Go to <http://www.acumentraining.com/resources.html> and look among the Acrobat samples.

[Next Page ->](#)

1. Prepare artwork In this case, I did the original artwork in Illustrator 6.0. The emoticons are built out of Stone Sans characters converted them to outlines. (Again, these two files are among the Acrobat samples on the Acumen Training website.)

2. Convert to PDF There are a lot of ways to do this, including:

- Print to a PostScript file, then hand the PostScript to Distiller or PDF Creator.
- Print to EZ-PDF (my personal favorite, although Enfocus hasn't yet re-released that product).
- Save the artwork to PDF directly from Illustrator.



In this case, since the artwork is so simple, I just saved each document to a PDF file from within Illustrator itself.

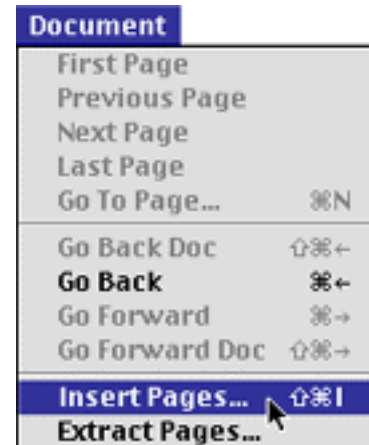
[Next Page ->](#)

3. Assemble a Single PDF file

Having made a series of PDF files, one for each of our stamps, we need to assemble them into a single PDF file, each stamp as an individual page.

The most straightforward way of doing this is to open one of the stamp files in Acrobat and then add the other files, one at a time, using the Acrobat *Include Pages* command. A bit tedious, but not at all hard.

This new PDF file, containing our individual stamps as separate pages, will represent our new rubber stamp category to Acrobat. Eventually, we will need to eventually move it into the *Stamps* folder so that Acrobat can find it.



A shortcut If you have more than a few PDF files to concatenate, you might find it a lot easier to save your original Illustrator (or whatever) artwork as a series of EPS files and then import them, one to a page, into a QuarkXpress/PageMaker/InDesign document. You can then convert the page layout document to PDF.

This will let you create your multi-stamp PDF file in one go without concatenating a series of individual PDF files.

[Next Page ->](#)

- 4. Set the Title Property** Now we want to set the *Title* property of the composite PDF document to the name of our new rubber stamp category.

Open the newly-made PDF file in Acrobat and select either:

- In Acrobat 4:
File>Document Info>General...
- In Acrobat 5:
File>Document Properties>Summary...



In either case, you will be looking at a dialog box similar to that at right. (This is the Acrobat 4 version; the Acrobat 5 dialog box is functionally the same.)

Enter the name you want for your new Stamp category into the *Title* box. Here I've entered "Emoticon" for my new category.

Click the *OK* button to dismiss this dialog box.

[Next Page ->](#)



5. Name Each Stamp

Now we need to assign a name to each of the pages in our PDF file; these will become the names of the individual stamps in our category.

In Acrobat, for each page in the PDF file, do the following:

1. Select *Tools>Forms>Page Templates...*



This yields the *Document Templates* dialog box.

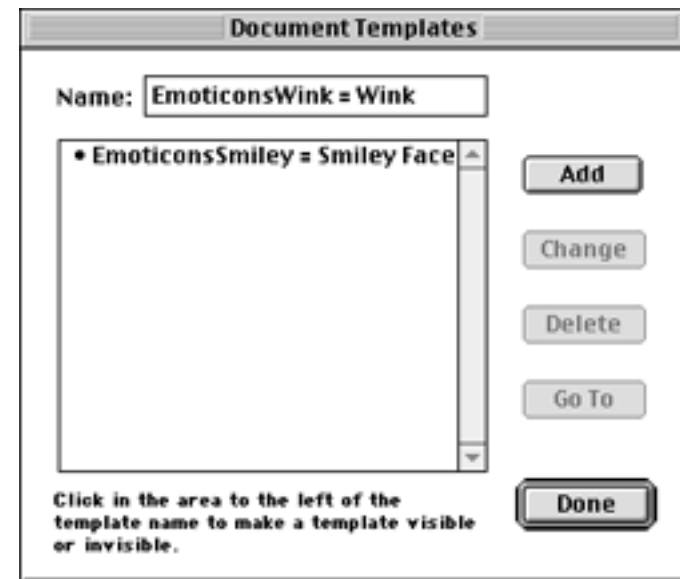
The Rubber Stamp mechanism uses page templates to assign names to the individual stamps in a category.

2. In the *Name* field, we shall enter text that defines the name for the current page's stamp. This odd-looking text has the form:

CategoryNameStampName = VisibleName

The text to the left of the equal sign is the concatenated category name (Emoticons) and stamp name (Wink).

To the right of the equal sign is the name of the stamp as it should appear in the *Category* pop-up menu. (The stamp's internal name does not need to be the same as the name presented to the user. This allows the stamp's name to be in a localized language.)



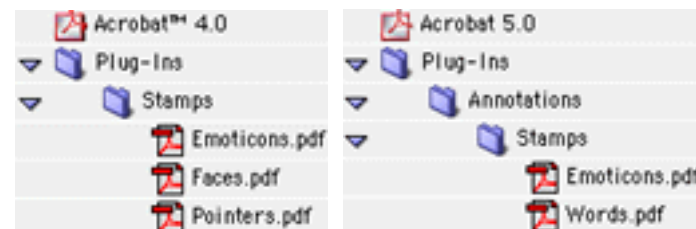
[Next Page ->](#)

6. Put the PDF File into *Stamps*

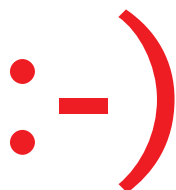
We're done with the PDF file. Save it to disk and then drag it into the *Stamps* folder.

Remember that this will be one of two places, depending on your version of Acrobat:

- Acrobat 4: *Acrobat>Plug-ins>Stamps*
- Acrobat 5:
Acrobat>Plug-ins>Annotations>Stamps



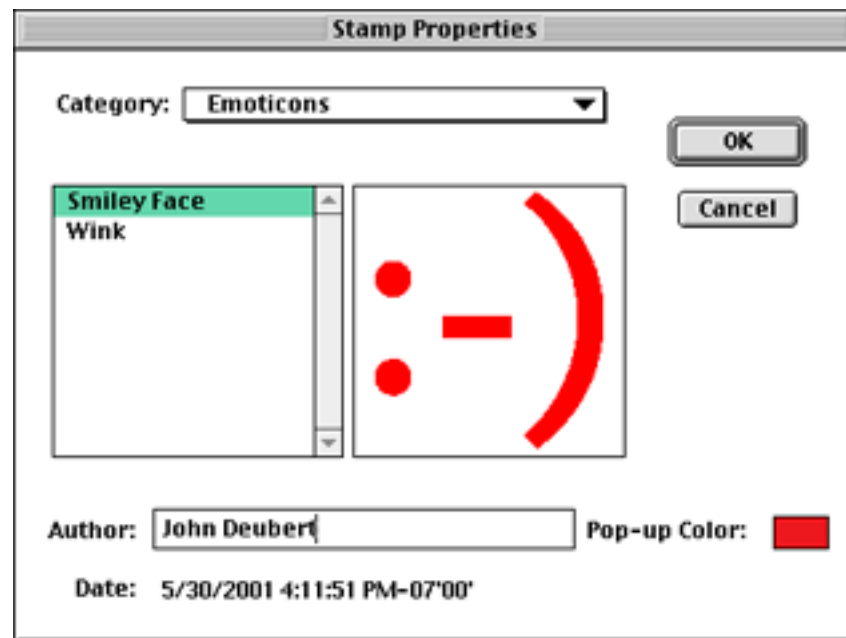
Done!



That's all there is to it. Launch Acrobat, select the Rubber Stamp tool and look at the *Stamp Properties*. You will find your new category and stamps added those supplied by Adobe.

By the way...

Other people don't need to have your stamp installed in their Acrobat in order to see it correctly. The PDF drawing commands for your stamp are embedded in the PDF file. It will display just fine.



[Return to Main Menu](#)

Adding Characters to a Type 1 Font

A question I see on the PostScript newsgroup periodically is “How do I add a character to a Type 1 font?” Often, the person asking the question is needing to add the Euro character to a font they must use.

This turns out to be a relatively simple task, if you know just a little PostScript.

In this month’s PostScript Tech article, we’re going to add a smiley face character to Helvetica, assigning it the character code normally occupied by upper-case “O.”

Thus, “O Me, O My, O” will print like this:

☺ me, ☺ my, ☺!

[Next Page ->](#)

Font Structure

Let's start by reviewing how the *show* operator draws characters in a Type 1 font. This will be review if you've taken any of the Acumen Training PostScript classes.

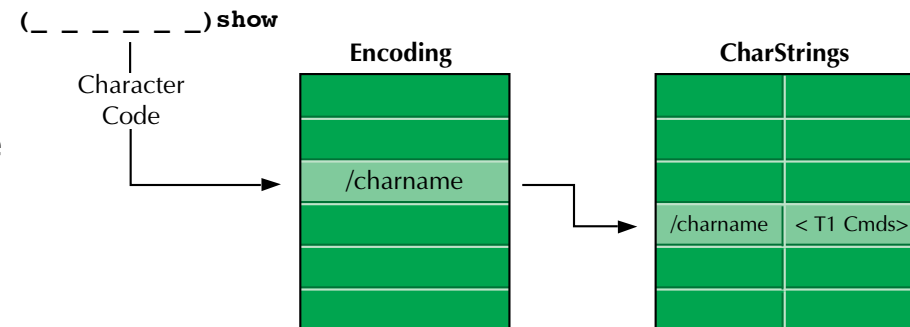
Encoding and CharStrings

There are two entries in a Type 1 font dictionary that together map character codes in a string into a set of drawing instructions in the font:

- **Encoding** is an array of character names that establishes the correspondence between character codes and characters. Each entry is the name of a character within the font. The position of each character name in this array determines the character code of that character.
- **CharStrings** is a dictionary that pairs character names with drawing instructions. Each key in this dictionary is a character name; associated with each name is a string whose bytes are the Type 1 drawing instructions for that character.

The *show* operator uses each byte in its string argument as an index into the current font's *Encoding* array; this gives it the name of the character that should be printed.

Show then uses this name as a key into *CharStrings*. The associated string full of drawing instructions is handed to the Type 1 rasterizer.



[Next Page ->](#)

Adding A Character

Surprisingly, a character name in *CharStrings* may be paired with a PostScript procedure, rather than with a Type 1 string. The *show* operator simply hands this procedure to the PostScript interpreter when it needs to print that character.

This makes it relatively easy to add your own character shapes to a Type 1 font. All you need to do is:

1. Write a PostScript procedure that draws your character.
2. Insert the modified procedure into the *CharStrings* dictionary, giving it whatever name you wish.
3. Insert the character's name into the *Encoding* array in the position corresponding to the character code you want for the new addition.

CharStrings

/A	< T1 Cmds>
/exclam	< T1 Cmds>
/mychar	{ PS Cmds }
/leftparen	< T1 Cmds>

[Next Page ->](#)

Changing read-only fonts

The only complication is that PostScript font dictionaries are read-only; you can't change them. What you *can* do is create an entirely new font identical to the original, but incorporating the changes you wanted to make to the original font.

In our case, we'll make a copy of Helvetica that contains a smiley face character.



If you have taken any of the PostScript classes, you will remember that there are four steps to changing a font:

1. Create a new dictionary the same size as your original font plus room for the changes you want to make.
2. Copy everything from the original font into the new dictionary. In PostScript Level 1 you need to *not* copy the */FID* entry, but we'll ignore this detail here, since Levels 2 and 3 don't need to worry about it.
3. Make whatever changes you want to your new dictionary.
4. Turn the dictionary (with your changes) into a font dictionary using the PostScript *definefont* operator.

The *definefont* operator takes a name and a dictionary from the operand stack, converting the dictionary into a font dictionary with the specified name. It returns a copy of the new font dictionary on the operand stack:

```
/FontName  << dict >>  definefont  ⇒  << fontdict >>
```

[Next Page ->](#)

The PostScript Code

Here is the PostScript code that installs the SmileyFace character into Helvetica. The new character is associated with ASCII code for "O." (This file is available among the PostScript samples at www.acumentraining.com/resources.html.)

```
/Helvetica findfont dup length dict copy begin      % Copy Helvetica

/CharStrings CharStrings dup length dict copy def  % Copy CharStrings

CharStrings /HappyFace                             % Insert the char. def.
{   1000 0 0 0 1000 1000 setcachedevice
    500 400 translate
    0 0 450 0 360 arc
    50 setlinewidth stroke
    -200 250 50 0 360 arc fill
    200 250 50 0 360 arc fill
    0 200 400 225 315 arc stroke
} bind put

/Encoding Encoding dup length array copy def        % Copy Encoding
Encoding 79 /HappyFace put                          % Insert char. name

/HelvHappy currentdict definefont pop               % Create the new font
end

/HelvHappy 30 selectfont                            % Use the new font
72 600 moveto
(O me, O my, O!) show
```

[Next Page ->](#)

Stepping Through the

Code Let's look at this PostScript code in detail...

Make a new dictionary `/Helvetica findfont dup length dict copy begin`

We start by making a new dictionary that is the same size as the Helvetica font dictionary. We copy everything from Helvetica into this new dictionary, which we then move to the top of the dictionary stack with *begin*. (This way we can get to its contents by simply referring to their names; we can also put things into it with a simple *def*.)

Make a writable CharStrings `/CharStrings CharStrings dup length dict copy def`

We want to add our character definition to *CharStrings*; unfortunately, this dictionary (which we copied from Helvetica) is read-only. So, we create a new dictionary and copy everything from our original *CharStrings* into it.

Finally, *def* places this new, writable *CharStrings* into our new font dictionary, replacing the read-only original.

[Next Page ->](#)

Insert the character

```
definition CharStrings /HappyFace
{      1000 0 0 0 1000 1000 setcachedevice
      500 400 translate
      ...
} bind put
```

We put our smiley face PostScript procedure into the new *CharStrings* dictionary, giving it the name "HappyFace."

A character procedure should draw a 1-point character at the origin in the Type 1 coordinate system, in which there are 1000 units to the point. (Our smiley face character is drawn 900 units in diameter.)

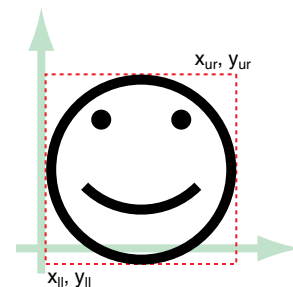
In addition to drawing a smiley face, our character procedure makes a call to the *setcachedevice* operator.

w_x w_y x_{ll} y_{ll} x_{ur} y_{ur} **setcachedevice** \Rightarrow ---

This is required; it defines the character's metrics.

w_x w_y are the distance *show* should move the current point after the character is printed. For us, w_y is zero, since we print horizontally.

x_{ll} y_{ll} x_{ur} y_{ur} are the coordinates of the lower left and upper right corner of the bounding box of the character. This bounding box should exactly enclose the painted character.



[Next Page ->](#)

Assign a character code `/Encoding Encoding dup length array copy def
Encoding 79 /HappyFace put`

To assign a character code to our character, we must place its name into the appropriate position of the font's *Encoding* array. Unfortunately, our *Encoding* (again, copied from Helvetica) is also read-only. So, as with *CharProcs*, we replace the original, read-only *Encoding* with a new, writable version, copying everything from the original array into the new.

Then we put the name "HappyFace" into position 79 of the new *Encoding*. Upper case O will print as a smiley face.

Create the font dictionary `/HelvHappy currentdict definefont pop
end`

Finally, we turn our dictionary into a font dictionary with *definefont*. In your PostScript class, we went into detail as to what *definefont* does. Here, we'll just say that the operator turns our dictionary into a font dictionary with the name *HelvHappy*. This is the name we'll need to hand to *findfont* or *selectfont*.

The *end* removes the dictionary from the dict stack; we don't need it there anymore.

[Next Page ->](#)

Use the font `/HelvHappy 30 selectfont
72 600 moveto
(O me, O my, O!) show`

☺ me, ☺ my, ☺!

We can now use the *HelvHappy* font like any other.

It's just like Helvetica, only annoyingly cheerful.

Caveats There are a couple of warnings associated with this new character:

Character Proc Restrictions The PostScript in your character procedure is subject to two important limitations:

Well-behaved Your PostScript procedure must be well-behaved, in the EPS meaning of the term. No use of *erasepage*, *quit*, *showpage*, or *init*-anything. See the EPS file specification for a complete description of "well-behaved." (You can get this from Adobe's web site or from [Acumen website's Resources page](#).)

No color operators allowed Your character procedure may not change the color. This means you may not call *setgray*, *setcolor*, *setrgbcolor*, *image*, or any other operator that changes the color or colorspace. If your character prints a bitmap, it must use *imagemask*, rather than *image*. This is a limit imposed by *setcachedevice*.

[Next Page ->](#)

If you absolutely must change color within your character definition, you will have to replace *setcachedevice* with a call to *setcharwidth*. This operator is similar to *setcachedevice*, but it takes only the current point offset:

w_x w_y **setcharwidth** \Rightarrow ---

Look in the PostScript Language Reference Manual or the PostScript Foundations student notes for more details on *setcachedevice* and *setcharwidth*.

ATM Won't Like This Note that *Adobe Type Manager* will not work with your inserted character. ATM only works with characters defined using Type 1 drawing commands.

[Return to Main Menu](#)

Schedule of Classes, June 2001 - August 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.

The PostScript classes are taught in Orange County, California, near the Orange County airport, and in London at Adobe Systems' offices near Heathrow.

PostScript Classes

<u>PostScript Foundations</u>	Orange Co., CA	June 4 - 8	Orange Co., CA	August 6 - 10
---	----------------	------------	----------------	---------------

<u>Advanced PostScript</u>	Orange Co., CA	July 16 - 19
--	----------------	--------------

<u>PostScript for Support Engineers</u>	Orange Co., CA	July 23 -27
---	----------------	-------------

<u>Jaws Development</u>	Orange Co., CA	July 30 - August 2
---	----------------	--------------------

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

PostScript Course Fees PostScript classes cost \$1,750 per student. The Jaws class is \$2,000 per student. These classes may also be taught on your organization's site.

[Registration →](#)

[Acrobat Classes →](#)

Acrobat Class Schedule

Acumen training teaches three users' classes in Adobe Acrobat (the links below will take you to the Acumen website's complete description). These are all taught with Acrobat 5, although Acrobat 4 versions may be taught if this is what your site uses.

[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.

[Troubleshooting with Enfocus' PitStop](#)

This class shows the student how to use all of the capabilities of this popular editing and preflight software.

On-site Only

The Acrobat 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.

[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](#)

What's New at Acumen Training?

PDF Class Info Here is some new information about the Technical PDF class currently in development.

Duration The class will be a four day, hands-on class.

Audience The course will be primarily aimed at printer engineers and technical support personnel in companies that are developing printers that consume PDF directly. The emphasis will be on PDF file structure and content with an eye toward what a printer needs to pay attention to when printing a PDF file.

The course is designed to be also very useful for people who will be producing PDF directly in their software.

Completion Data I'm currently looking to conduct the first class in October.

Course Topics The current list of topics is on the Acumen Training website (www.acumentraining.com/Descr_PDFTech.html). I am looking for comments on this list: what should be added, removed. Anything you wish to say.

[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?

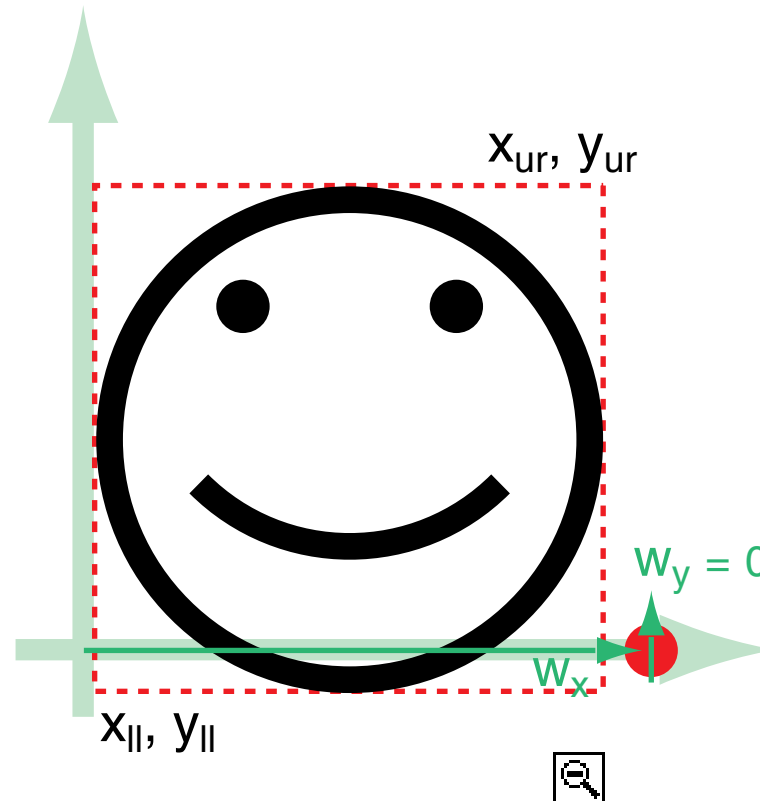
Questions and Answers. We are planning a Q&A section for future issues. Do you have any questions about Acrobat, PDF or PostScript?

Please send any comments, questions, or problems to:

journal@acumentraining.com

[Return to Menu](#)

w_x w_y x_{ll} y_{ll} x_{ur} y_{ur} setcachedevice \Rightarrow ---



Sorry this illustration is a bit busy.

Note that w_x and w_y are the distances *show* should move the currentpoint after the character is printed.

w_x will be the character's width plus side bearings to the left and right.

w_y will be zero, presuming your font prints horizontally.

Everything here is expressed in the Type 1 coordinate system wherein there are 1000 units to the point.

[Return](#)

General Info


Filename: Wheezy:Acumen Training:Acumen
Journal:2001.06:Illustrations:art.emoticon 1.pdf

Title:

Subject:

Author:

Keywords:

Binding: 

Creator: Adobe Illustrator 6.0

Producer: Acrobat PDF File Format 1.1 for Macintosh

PDF Version: 1.0

Created: 5/30/2001 3:06:59 PM

Modified: 5/30/2001 3:06:59 PM

Optimized: No

File Size: 2353 Bytes

