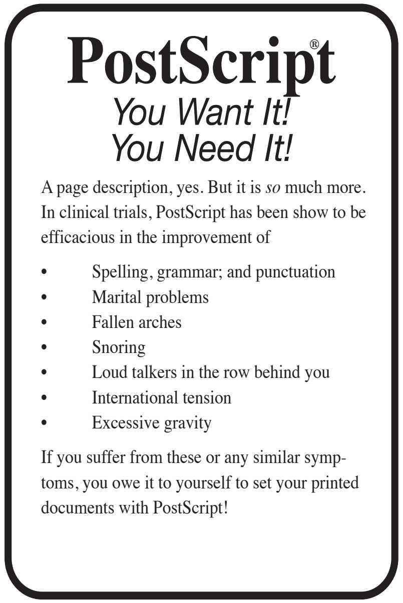
Table of Contents

The Acrobat User	**Changing Calculation Order in Acrobat Forms** By default, form fields in Acrobat are calculated in the order they were created. Sometimes you need to change this order. This month we see how to do this.
PostScript Tech	**The *arcto* and *arct* Operators** PostScript programmers often overlook these useful operators.
PDF & XPS Nuggets	Introducing a new feature: brief-but-true informational nuggets about the XPS and PDF file formats.
Class Schedule	March-April-May
What's New?	**The *Journal* is back after a hiatus** The Acumen Journal is back again and about time!
Contacting Acumen	Telephone number, email address, postal address

Journal feedback: suggestions for articles, questions, etc.

The *arcto* and *arct* operators

One of the very first programming assignments I was given when I went to work at Adobe (back in 1984!) was to produce a one-page informational handout for our marketing and sales department (consisting at the time of only of one person; heck, the entire company at the time was only 19 people). The handout looked roughly like the illustration at right and was done with hand-written PostScript, since there were no PostScript-producing graphics applications at the time.



PostScript®

You Want It!
You Need It!

A page description, yes. But it is *so* much more. In clinical trials, PostScript has been show to be efficacious in the improvement of

- Spelling, grammar; and punctuation
- Marital problems
- Fallen arches
- Snoring
- Loud talkers in the row behind you
- International tension
- Excessive gravity

If you suffer from these or any similar symptoms, you owe it to yourself to set your printed documents with PostScript!

It’s a simple layout and a relatively easy programming task. The only tedious part was doing the rounded border. Being new to PostScript, I created this with a combination of *lineto* and *arc* operators, so I had to calculate the position of several tangent points. Not hard, but, as I say, a bit tedious.

I could have saved myself some trouble if I’d known about a convenient pair of PostScript operators: *arcto* and *arct*. These make it extremely easy to create rounded-corner line art. Being new to PostScript, I hadn’t yet noticed them in the PostScript Language Reference Manual, so I did it the hard way.

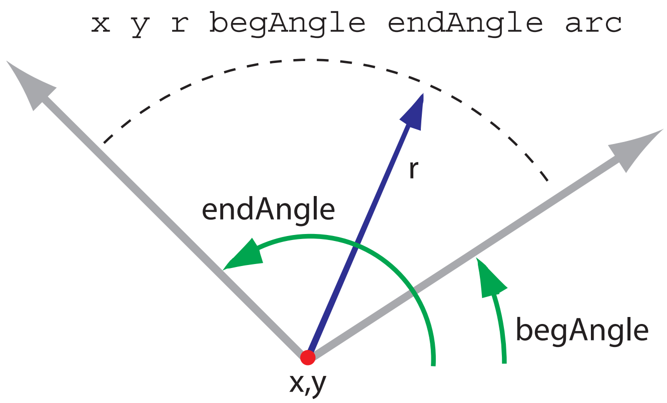
Looking at the code produced by many PostScript sources, I suspect that a lot of people *still* don’t know about these operators.

Well, we can fix that. This month, let’s see how to use *arcto* and *arct*.

Review: the *arc* Operator

We’ll start by reminding ourselves how the common *arc* operator works. You should all remember this operator from your *PostScript Foundations* or *Troubleshooting PostScript* classes (take them both!), but for completeness, we’ll start here.

The *arc* operator adds a circular arc—a part of a circle—to the current path. The operator takes the following arguments from the operand stack:

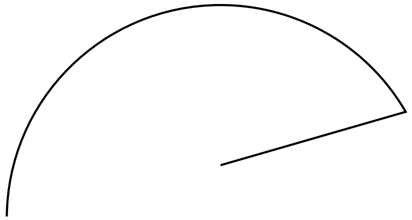


x y	The coordinates of the center of the circle of which the desired arc is a part.
r	The radius of the circle.
beginAngle	The position of the beginning of the arc, expressed as an angle measured along the circle; *0°* is taken to be the direction of the positive *x* axis.
endAngle	The position of the end of the arc, expressed as an angle measured along the circle; *0°* is, again, taken to be the direction of the positive *x* axis.

The operator adds the specified arc to the current path, setting the current point to the end of the arc. As a side benefit, if there already exists a current point, *arc* will do an implicit initial *lineto,* adding to the current path a line segment from the current point to the beginning of the arc.

Thus, the following code would produce the results at right.

```
300 400 moveto    % This’ll be the start of the line segment
300 375 100 30 180 arc
stroke
```



Note the line segment (extending from *300,400* to the beginning of the arc) was added by the *arc* operator.

The *arcto* & *arct* Operators

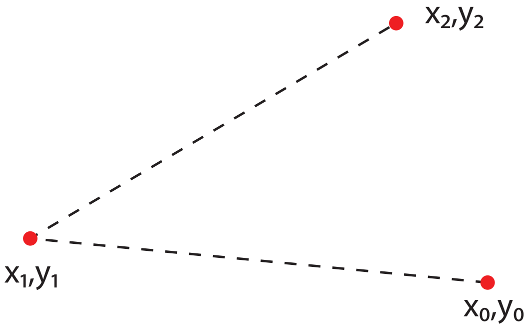
arcto

The *arcto* operator adds an arc to the current path, usually preceded by a line segment; this sounds identical to the *arc* operator, but it's not; *arcto* exists explicitly for the creation of round-corner line art. It is a bit tricky to explain verbally, though how it works is actually quite easy to understand.

The *arcto* operator takes five numbers from the operand stack, representing two *x,y* pairs and a radius.

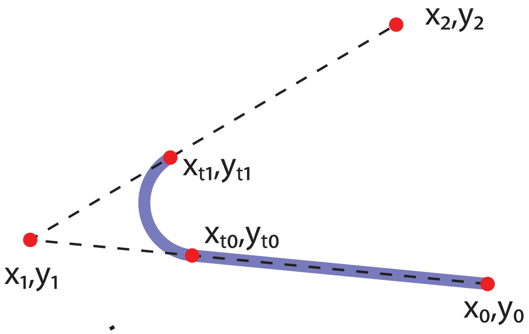
```
x1  y1  x2  y2  r  arcto  =>  xt1  yt1  xt2  yt2
```

It returns four numbers, representing a two coordinate pairs; we'll talk about these in a moment.



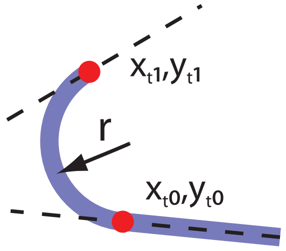
The operator uses a pair of line segments, defined by the current point (x_0,y_0 in the diagrams at right), x_1,y_1, and x_2,y_2.

Arcto adds to the current path (here's the part that's hard to describe) a line segment and a circular arc of radius *r*. The arc occupies the interior of the angle formed by the two line segments and is tangent to those line segments; the line segment extends from the current point to the beginning tangent point of the arc.



Oh, it's easy; look at the middle diagram at right.

The argument *r* is the radius of the circular arc that occupies the angle, as in the lower figure.



The current point is left at the second tangent point, x_{t1},y_{t1}.

Finally, as a service, *arcto* leaves on the stack the *x* and *y* coordinates of the two tangent points. This isn't often useful, but in the rare cases where you need these numbers, you generally *really* need them.

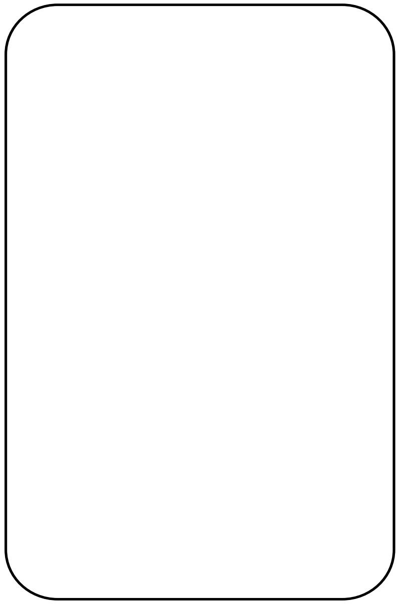
arct

On the other hand, it *is* very unusual that you need to know the coordinates of those tangent points and it gets to be a mild nuisance to do a set of *pops* after each *arcto;* I generally would forget to do so in the initial version of my PostScript programs.

In PostScript Level 2, Adobe added the *arct* operator to the language. This operator is identical to *arcto* except that it doesn't have any return values; the tangents' coordinates are not left on the operand stack.

This is the operator you will most often use in your page descriptions.

The rounded rectangle for the leaflet ended up looking like this:



```
/radius 20 def
/ht 240 def
/wid 150 def

250 200 translate

0 radius moveto
0 ht wid ht radius arct
wid ht wid 0 radius arct
wid 0 0 0 radius arct
0 0 0 ht radius arct

stroke
```

Pretty much the same as the classic *moveto, lineto, lineto, lineto, closepath* for a standard rectangle.

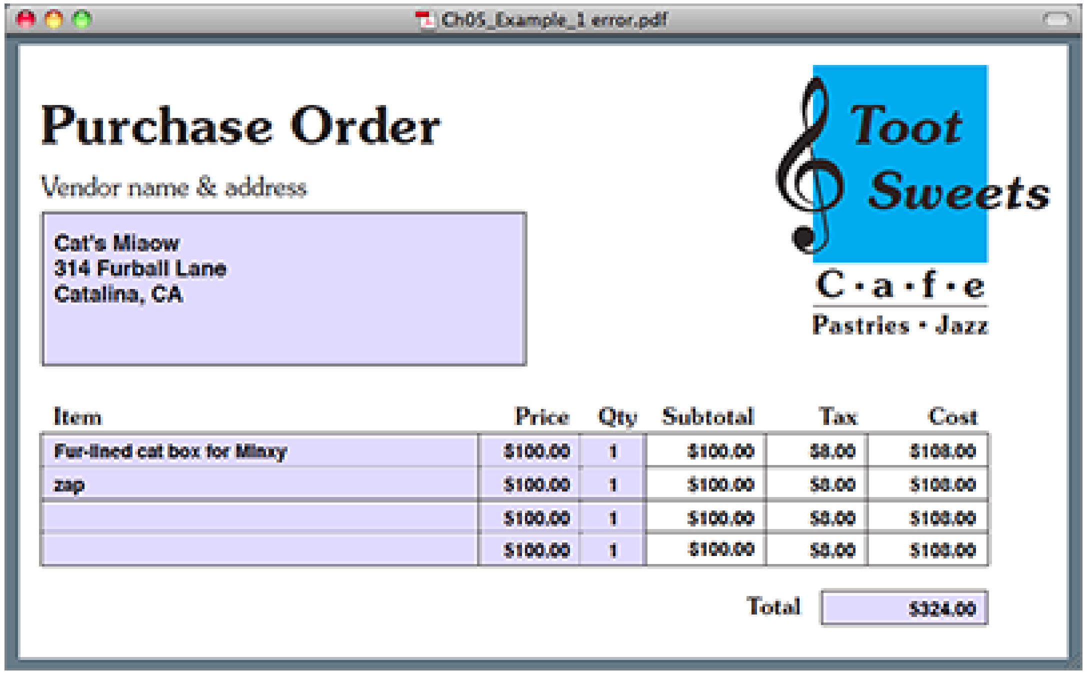
Now, if they'd only Adobe had added a rounded equivalent of *rectfill* and *rectstroke,* life would be grand.

There's no satisfying some of us.

Changing Calculation Order in Acrobat Forms

I got an emailed question recently regarding one of the examples in my old *Extending Acrobat Forms With JavaScript* book. (Out of print several years now, alas; where *are* people getting their copies?)

The example is the form pictured at right: a simple order form for a fictional bakery.



Purchase Order

Vendor name & address

Cat's Miaow
314 Furball Lane
Catalina, CA

Item	Price	Qty	Subtotal	Tax	Cost
Fur-lined cat box for Minxy	$100.00	1	$100.00	$8.00	$108.00
zap	$100.00	1	$100.00	$8.00	$108.00
	$100.00	1	$100.00	$8.00	$108.00
	$100.00	1	$100.00	$8.00	$108.00

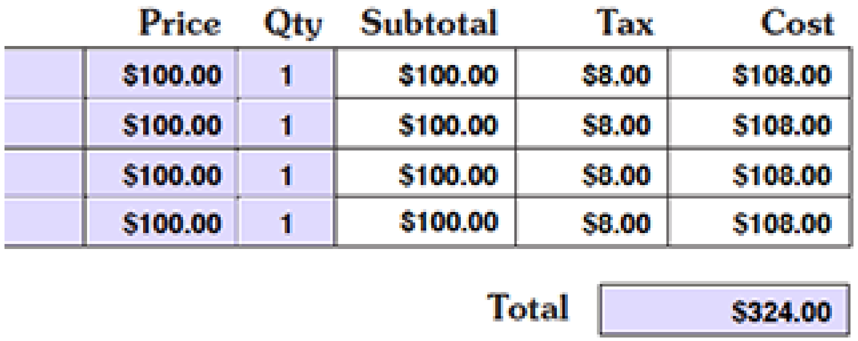
Total $324.00

The problem with the example is that the *Total* field doesn't immediately update when you change the price or quantity fields. To be specific, *Total* always lags one change behind; if you make two changes to *Price* or *Quantity,* the total will update to reflect the first change only when you make the second change.

The source of this problem is relatively subtle: the fields in the form were being calculated in the wrong order.

Let's look at this problem in more detail and see how to fix it.

Calculation Order



	Price	Qty	Subtotal	Tax	Cost
	$100.00	1	$100.00	$8.00	$108.00
	$100.00	1	$100.00	$8.00	$108.00
	$100.00	1	$100.00	$8.00	$108.00
	$100.00	1	$100.00	$8.00	$108.00

Total $324.00

Consider the problematic form; the values in most of the text fields are calculated from the values in other fields. In detail:

- Each *Subtotal* field is calculated from the *Quantity* and *Price* values.
- The *Tax* field is calculated from the *Subtotal* value (using a constant, hardwired tax rate; I got lazy).
- The *Cost* fields are calculated from their corresponding *Subtotal* and *Tax* fields.
- Finally, the *Total* field is the sum of the four *Cost* fields.

In considering these fields, it becomes evident that the order in which the fields are calculated is very important:

- Each *Subtotal* field must be calculated before its corresponding *Tax* field.
- The *Tax* field must be calculated before the *Cost* field.
- All four *Cost* fields must be calculated before the *Total* field can be updated.

By default, Acrobat calculates form field values in the order in which the fields were originally added to the form. This was the source of the example form's trouble: when I created the form, I added the *Total* field before I added the *Tax* or *Cost* fields. (An earlier version of the example didn't calculate taxes, so the *Tax* and *Subtotal* fields didn't exist.)

As a result, when a change was made to one of the price or quantity fields, the total cost was calculated *before* the tax or subtotal.

Bummer.

Changing the Order

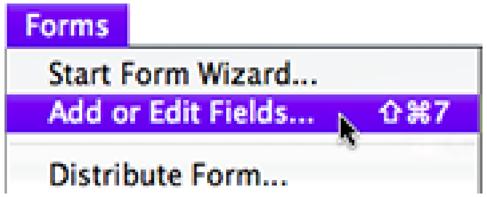
We could reconstruct the form from scratch, adding the fields in the correct order, but that would be a horribly clunky way of correcting what is, at root, a fairly minor problem.

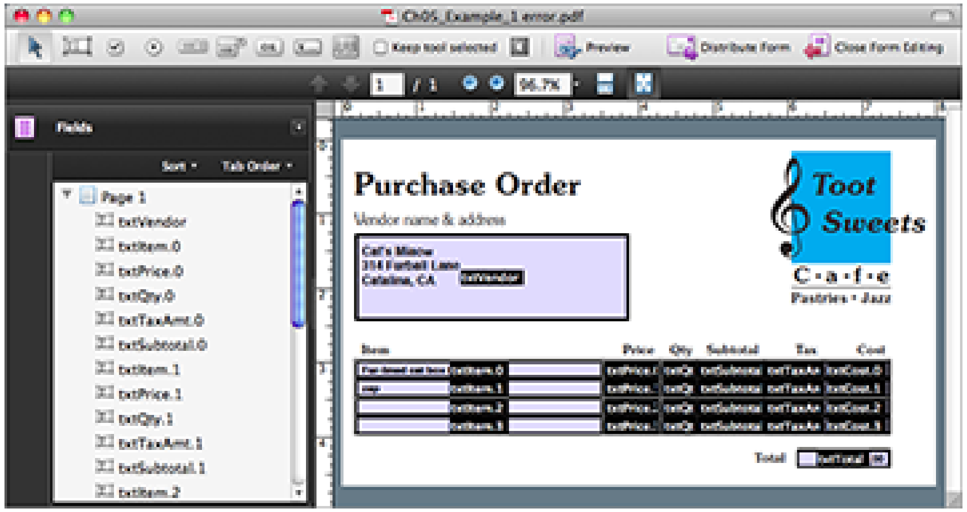
Happily, Acrobat Pro lets you change the order in which form fields are calculated.

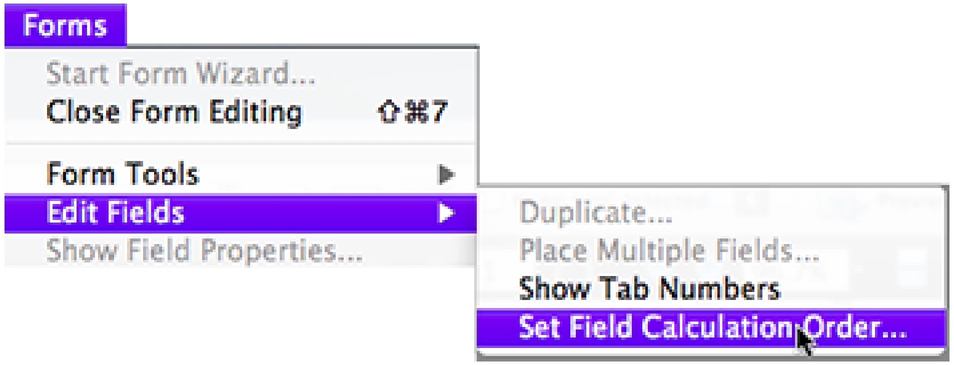
Here’s the procedure:

1. With the form document open, select *Forms > Add or Edit Fields.*

   Acrobat will open the *Edit Fields* window, showing the numbered form fields an a toolbar full of useful tools that we shall completely ignore.



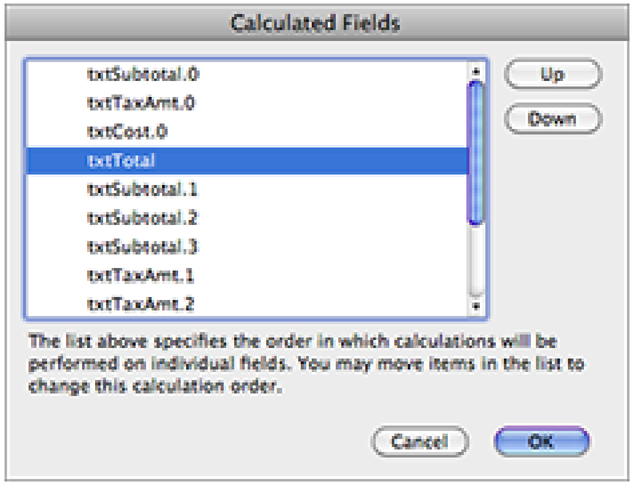




2. Select *Forms* > *Edit Fields* > *Set Field Calculation Order.*

   Acrobat will display the *Calculated Fields* dialog box. This displays a list of all of the calculated fields in the form, presented in calculation order.

3. Select an out-of-order field and click the *Up* and *Down* buttons until the field is in the proper position in the list.



txtSubtotal.0
txtTaxAmt.0
txtCost.0
txtTotal
txtSubtotal.1
txtSubtotal.2
txtSubtotal.3
txtTaxAmt.1
txtTaxAmt.2

The list above specifies the order in which calculations will be performed on individual fields. You may move items in the list to change this calculation order.

4. Click the *OK* button to dismiss the *Calculated Fields* dialog box, returning to the *Edit Fields* window.

5. Select *Forms* > *Close Form Editing* to close the *Edit Form* window.

That's it. Simplicity itself, though it can be surprisingly hard to find the appropriate menu selections when you first go looking for them.

I'm pleased to say that the problem example works perfectly well once the fields are calculating in the proper order.

All of life's problems should be so easy, no?

By the way, I decided not to fix the version of the example form that's posted on the Acumen Training website. You can use it for practice.

(The book examples are all available at www.acumentraining.com/acrobatjs.html.)

PDF & XPS Nuggets

This issue introduces a new feature to the *Acumen Journal*. Every issue will present two very short, but useful nuggets of information, one each for PDF and XPS.

This is better demonstrated than described, so without further ado…

PDF Nugget

PDF is a Text Format

PDF has a reputation for being an unreadable file format. And with some justification; if you open a PDF file in a text editor (as we do in the Acumen Training PDF classes), you will usually find yourself staring at a lot of very-hard-to-pronounce binary:

```
HâúUÎn”0~Ç°É°<?>≠ÆÌ8NÇ™J[ª¢!Möh$~¿Ñ™6eÅˆFí2ÿ”s|O[ZötÊcüÀ˜}q˚w≥∫EÉAˇvt3F<?>
áW„Q= ∞î°4g®.Éh£∂‰fl7XU(Ëø/W≥∂˙Qé∂´m]≠À∂ÆÊ®ÆÇ˛€)A_ö eHpú!&8§ ±J¥|mÚP»ëTF
ˆAø†üÂf±ÑÛb≈10T4à˘ˇYn’‡Ë€<?>…Ù©¿     *÷¡«.¢$K3ëEΩ,§rıΔπì(¶a%›ï4•;10E4
∑“›H”FîÎSÎöΠWn5uµ4;iÊ◊[;≥/ …}ÒNçFıhî,å   9…zÜT‘t‘ò‘‡œ¸r ±X7fs&aO÷_≥WƒÛ
```

It comes as a surprise to most people that the PDF file format is, in fact, made up of ASCII text, perfectly human readable. The reason for the binary gobbledegook is that the contents of most PDF files are compressed, to reduce file size. If you uncompress the binary (it's usually Flate compressed in Acrobat-produced files), you will be looking at something that appears to be slightly aberrant PostScript:

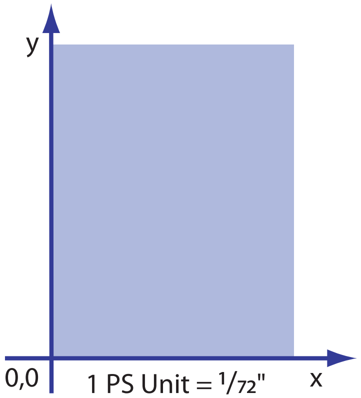
```
100 500 m
200 500 l
200 600 l
100 600 l
8 w
s
```

The drawing commands in a PDF file aren't required to be compressed, it's just by far the most common case. We do hand-written PDF in the Acumen Training classes by simply typing in the PDF code and handing it to the reader without compressing it.

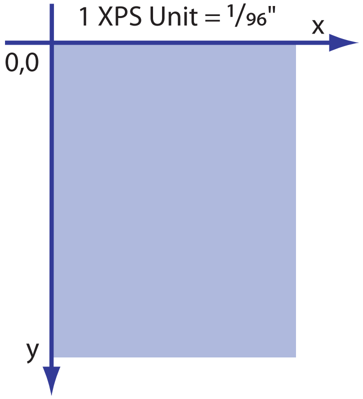
XPS Nugget:

XPS Coordinate System

I came to XPS from a PostScript and PDF background. PostScript and PDF look at the world using a blend of mathematical convention and traditional, pre-computer-age graphics and typography. This is evident in the coordinate system used by both PS and PDF: units are 1/72-inch (typographic *points,* close enough), the *x* and *y* axes increase to the left and upward, respectively, as they did in your grade school math classes; as a consequence of the axes' directions, the coordinate origin is placed at the lower left corner of the page, as at right.



XPS, on the other hand, derives its graphical basis from computer graphics and, in particular, the default coordinate system used by *Microsoft Windows'* graphics. In common with most computer graphics applications, the *x* and *y* axes increase to the right and downward, respectively; the coordinate origin is therefore in the upper left corner of the page. The units used by XPS are 1/96–inch, a unit adopted by Microsoft for its *Windows Presentation Foundation* environment.



A Question:

Can anyone in Readerland tell me *why* Microsoft chose 1/96" as their standard unit of measure? This is not a contest; I don't know the answer and am curious.

I'll post the answer in the next *Journal* issue.

Of course, there's no intrinsic virtue attached to either of these coordinate systems; anything you do in one you can do in the other.

I do find I'm still more comfortable with the 1/72-inch units, since I've worked with PostScript and PDF for a *very* long time, while WPS is still a relative newcomer in the Acumen Training Stable of Classes; also, an awful lot of the projects I work on are spec'd out in points and picas, making the 1/72-inch units a (very minor) convenience.

Schedule of Classes, March–May 2010

At right are the dates of Acumen Training's upcoming classes. Clicking on a class name will take you to the description of that class on the Acumen Training website.

PDF Classes

PDF 1: File Content and Structure	Mar 29–Apr 1		May 3–6
PDF 2: Advanced File Content			
Support Engineers' PDF		Apr 29–30	

PostScript Classes

PostScript Foundations	Mar 22–26		May 10–14
Advanced PostScript			
Variable Data PostScript			May 17–21
Troubleshooting PostScript		Apr 26–28	

XPS Classes (New!)

XPS File Content and Structure	Mar 9–11		May 25–27

O.C. and On-Site

These classes are taught in Orange County, California and on-site at corporate sites world-wide.

Please see the Acumen Training web site for more information, including an up-to-date schedule.

Class Fee

Classes cost $2,000 per student, with the following exceptions:

- XPS class $1,500
- *Troubleshooting PostScript* $1,500
- *Support Engineers' PDF* $1,000

There is a 10% discount for signing up three or more students.

Note that if you have four or more students that need to take a class, it will almost certainly be cheaper to arrange an on-site class.

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: 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: www.acumentraining.com/register.html

email: john@acumentraining.com

telephone: 949-248-1241

mail: 24996 Danamaple, Dana Point, CA 92629

On-Site Classes

Information regarding classes on corporate sites is available at www.acumentraining.com/Onsite.html. These courses are taught throughout the world; for additional information on classes outside the United States, go to www.acumentraining.com/OnsitesWorldWide.html.

Back issues

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

What’s New at Acumen Training?

The *Journal* is Back

After a hiatus, I’m back to doing the *Acumen Journal*.

The *Journal* sports a new feature, *PDF & XPS Nuggets,* which gives me a chance to say something about those file formats. Neither format lends itself to longer articles, so I’ll just be satisfied with passing along something short and interesting.

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 at life's strange combination of beauty and futility?

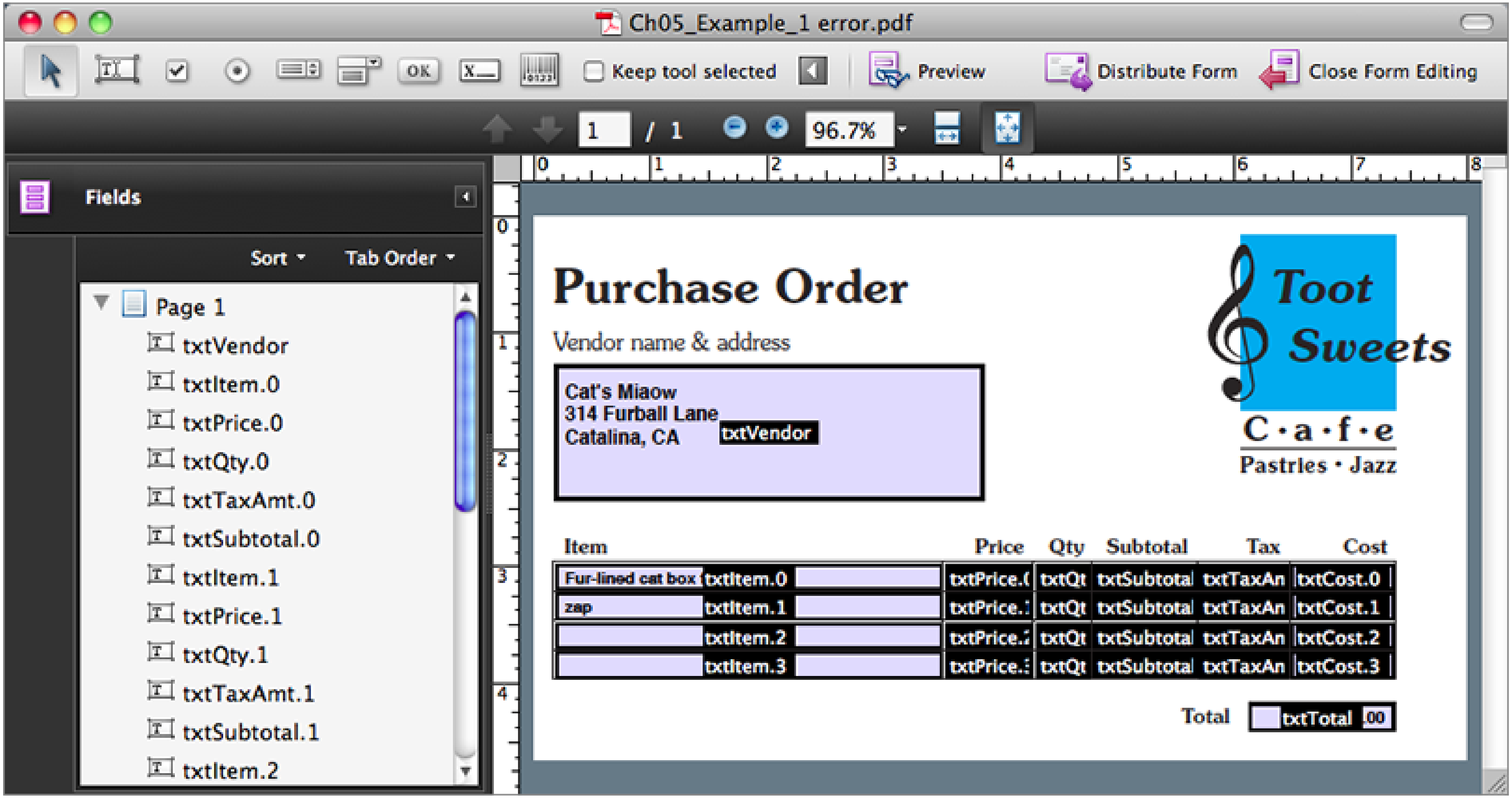
Suggestions for articles. Each Journal issue contains one article each on PostScript and Acrobat. What topics would you like me to write about?

Questions and Answers. Do you have any questions about Acrobat, PDF, XPS, 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:

john@acumentraining.com

Edit Fields Dialog Box



Page 1
- txtVendor
- txtItem.0
- txtPrice.0
- txtQty.0
- txtTaxAmt.0
- txtSubtotal.0
- txtItem.1
- txtPrice.1
- txtQty.1
- txtTaxAmt.1
- txtSubtotal.1
- txtItem.2

Purchase Order

Vendor name & address

Cat's Miaow
314 Furball Lane
Catalina, CA
txtVendor

Item			Price	Qty	Subtotal	Tax	Cost
Fur-lined cat box	txtItem.0		txtPrice.(	txtQt	txtSubtotal	txtTaxAn	txtCost.0
zap	txtItem.1		txtPrice.1	txtQt	txtSubtotal	txtTaxAn	txtCost.1
	txtItem.2		txtPrice.2	txtQt	txtSubtotal	txtTaxAn	txtCost.2
	txtItem.3		txtPrice.3	txtQt	txtSubtotal	txtTaxAn	txtCost.3

Total txtTotal .00