Table of Contents

Self-Modifying Procedures in PostScript

This month we'll discuss an aspect of PostScript that is both fun and, as far as I can tell, completely useless. At least, *I've* never found a use for it.

PostScript procedures are implemented as executable arrays; we can do anything to a procedure body that we can to any other array. In particular, we can modify PostScript procedures on the fly using the *put* operator. In fact, it is relatively easy to write a PostScript procedure that changes its own definition.

There is generally no good reason to do this; in 25 years of PostScript programming, I've never had occasion to use this technique for anything other than, well, demonstrating the technique.

Still, I think it's an interesting example of the flexibility of PostScript as a programming language; also, it *is* fun in a demented sort of way.

Let's look at how to do this.

Background: The Nature of Procedures

A PostScript procedure is actually an array of objects; it is like any other array, except that it is marked executable.

Thus, the PostScript line

```
{ 72 mul }
```

constructs an array with two objects in it: the number object *72* and the executable name object *mul.*

Whenever the PostScript interpreter encounters such an array as the result of a name lookup (of the name "inch," perhaps), it traverses the array one object at a time, executing or pushing each object on the stack

as appropriate. (In our example, of course, the *72* goes on the stack and then the executable name *mul* is looked up and its value is executed.)

Modifying a Procedure

However, once again, an executable array is just an array. Given a procedure definition like

```
/inch { 72 mul } bind def
```

we could change the numeric value in it by doing the following:

```
/inch load          % Get the definition of inch; put the {proc} on the stack
0 100 put           % Put the value 100 into position 0 of the proc
```

The next time we use the *inch* procedure

```
3 inch
```

we will multiply the procedure's argument by *100,* rather than the original *72;* if you examined the definition of *inch,* you'd find it is now

```
{ 100 mul }
```

Cool, huh?

Well, okay. Interesting, anyway.

Self-Modifying Procedures

We can take this one step further by writing a procedure that fetches its own definition and then modifies that.

Thusly:

```
/inch {  72 mul
         /inch load
         0 100 put
       } bind def
2 inch =
3 inch =
```

The above piece of PostScript code writes the numbers 144 and 300 to the output stream.

The *inch* procedure here multiplies its argument by 72 (initially), then fetches its own executable-array definition and inserts *100* into position *0,* replacing the number *72* that was originally in that position.

Thus, after the first execution of the procedure, *inch* is defined to do a *100 mul*.

So, What Can We Do With This?

Not much, actually. Most of the circumstances I can imagine that would prompt me to use this would be better served with a variable or other technique.

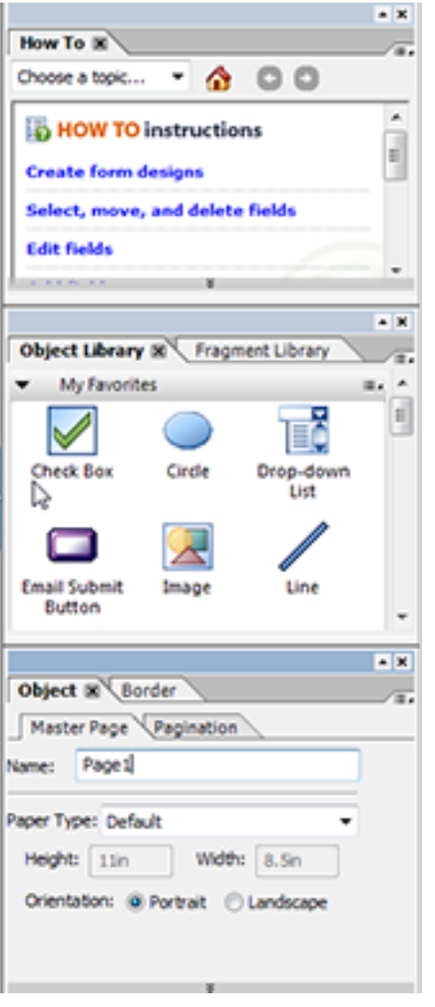
So, here's my challenge: can you come up with a situation in which a self-modifying procedure is—at least arguably—the best solution?

Send it to me, if so, and I'll share it next time.

Comparing the Acrobat Form Toolbar and *LiveCycle Designer*

Acrobat has long had support for interactive form fields. Since primeval times, Acrobat has provided tools you could use to create interactive form fields that collect data from end users and send it off somewhere for processing.

Acrobat 7 (I think it was) introduced a new, Windows-only form creator called *LiveCycle Designer*. When invoked, this editor presents a palette of form fields that you can create on the page. *LiveCycle Designer* gives you a form field designer that has much the flavor of a drawing package, letting you easily create labelled form fields of any PDF type.



LiveCycle Designer (hereinafter known as "LCD") was an instant hit in the Acrobat world; people ooh-ed and ahh-ed over its graphics-program-like interface.

And yet, for myself, I've never seen how *LCD* is a practical improvement over the long-standing Acrobat form field tools that reside in the Form Fields toolbar. (The so-called "AcroForms" tools.) LCD is flashier, but, as far as I can see, a preference for one over the other is a matter of personal taste.

Let's look at how we go about making a form field with each of them and then you can decide which you prefer.

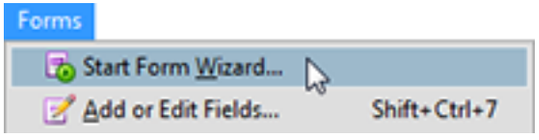
LiveCycle Designer

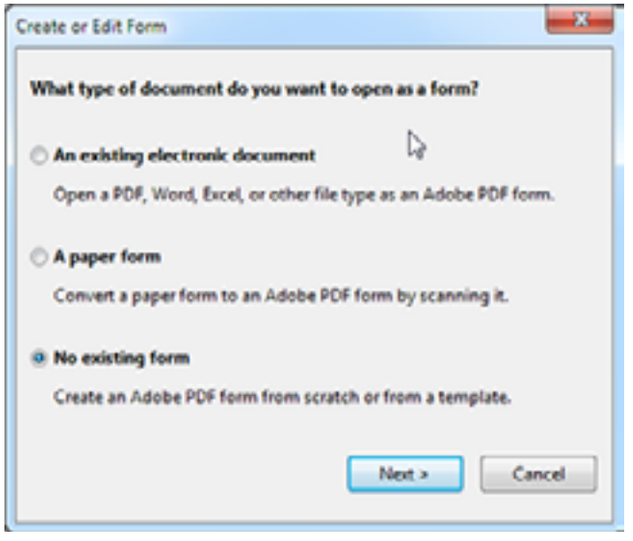
Let's start with *LCD*. We'll create a new Acrobat form and add a single pushbutton to that form.

Launching *LCD*

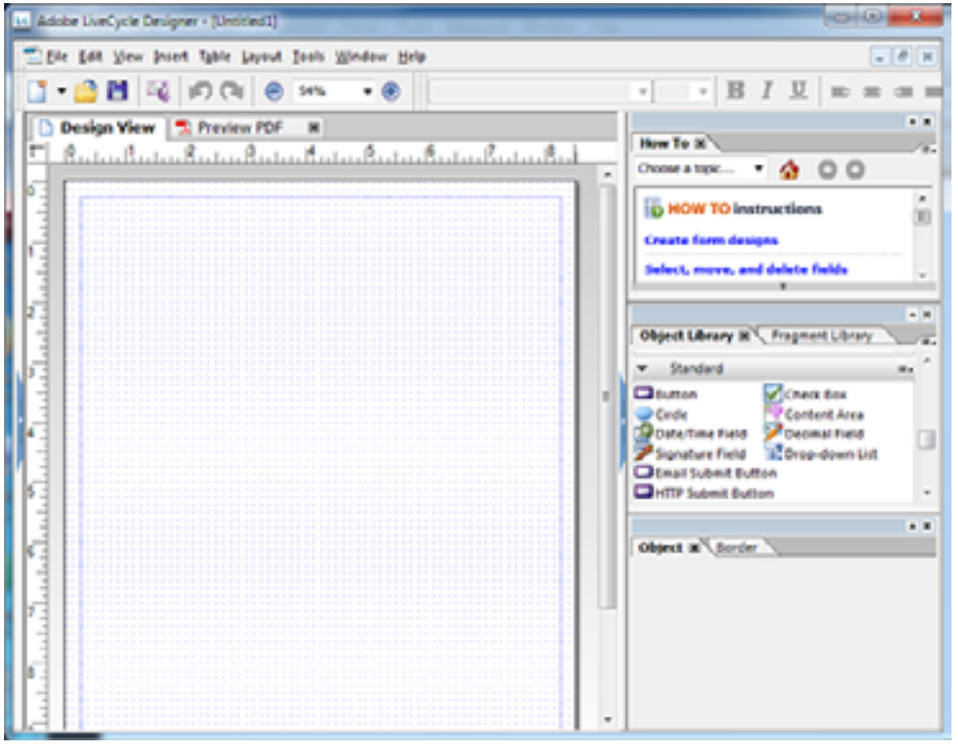
You launch *LCD* by selecting *Start Form Wizard* from the *Forms* menu.

Acrobat presents you with a dialog box (far right) that asks what you want to use as the starting point for your form.



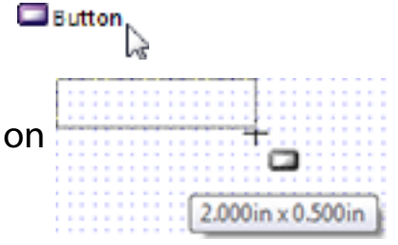


If you select *No Existing Form,* Acrobat launches *LCD* and presents you with its design window (below).



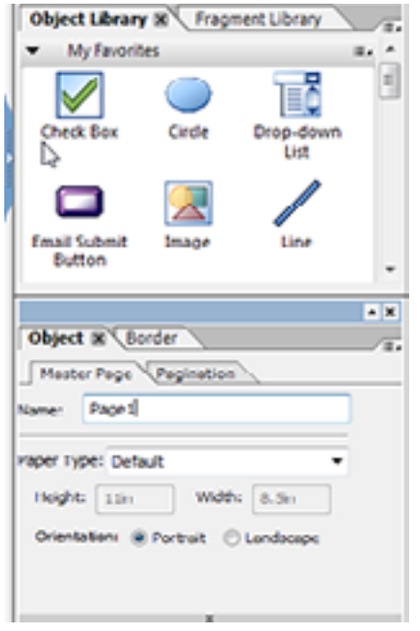
Create the Pushbutton

The right-hand side of the *LCD* design window presents an *Object Library*, a panel of tools you can use to create a variety of Acrobat form fields. If you scroll down this panel, you will eventually find a *Button* tool.



Click on the button tool and the cursor will turn into a crosshair you can use to drag out a rectangular button on the PDF page.

LiveCycle Designer will add the button to the page. The panels on the right will now include several tabs with properties you can set for the newly-created pushbutton (color, label, border type, etc.), as below right.

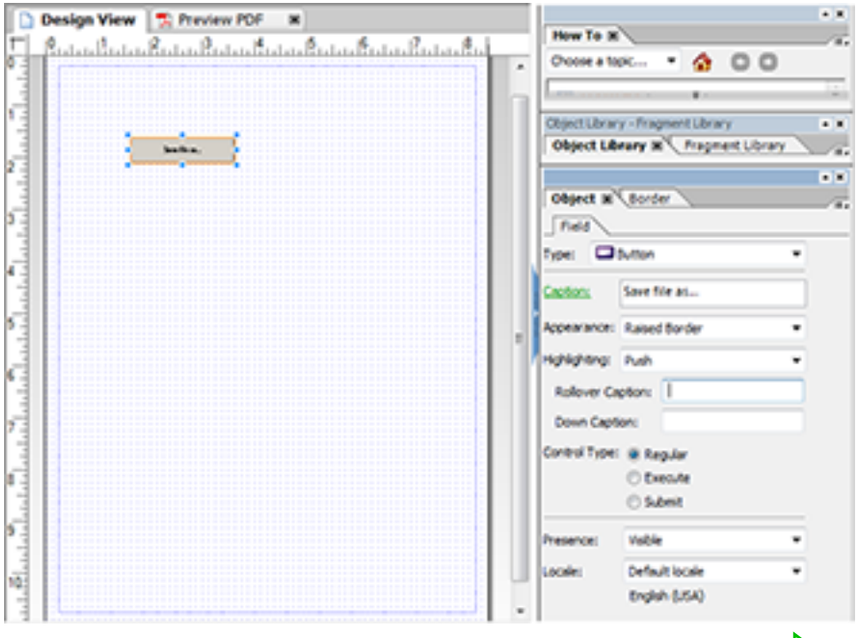


So, Summarizing…

That's the drill:

- Select a field type from the palette.
- Drag out the location you want for the form field.
- Specify the properties you want to determine the button's appearance and (to an extent) behavior.
- Repeat.

Now let's see how we do this with the AcroForms toolbar.

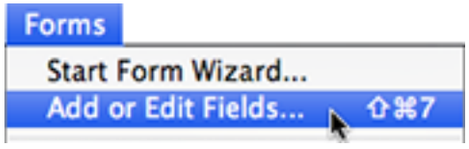


AcroForms Toolbar

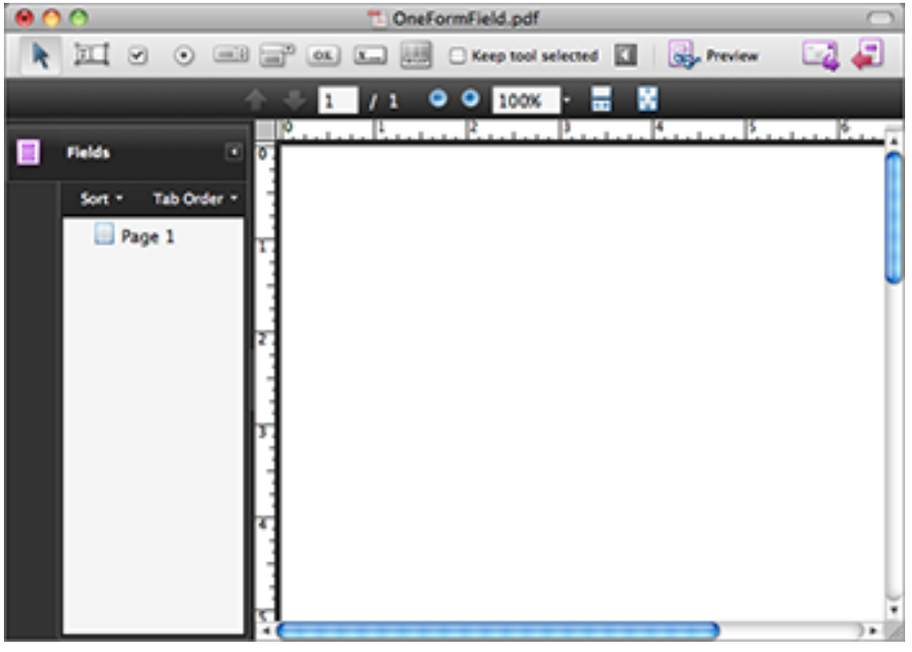
Acrobat's intrinsic form field tools are grouped in a toolbar called, for historical reasons, the AcroForms toolbar.

Add Fields with the Toolbar

To add form fields to an existing PDF file, start with the document open in Acrobat and select *Forms>Add or Edit Fields*. Acrobat will open the document in a form editor window, below right.

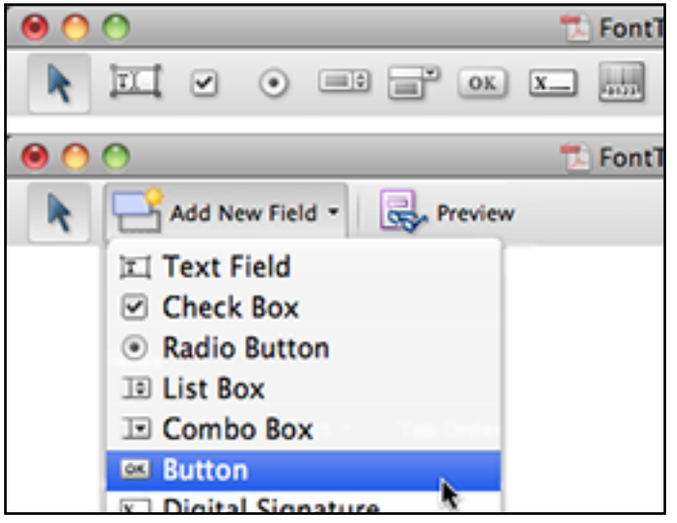


(Mac, like Windows, allows you to select *Forms>Start Form Wizard,* but since LCD doesn't exist for the Mac, you just end up staring at this same *Edit Fields* window.)



Adding a Button

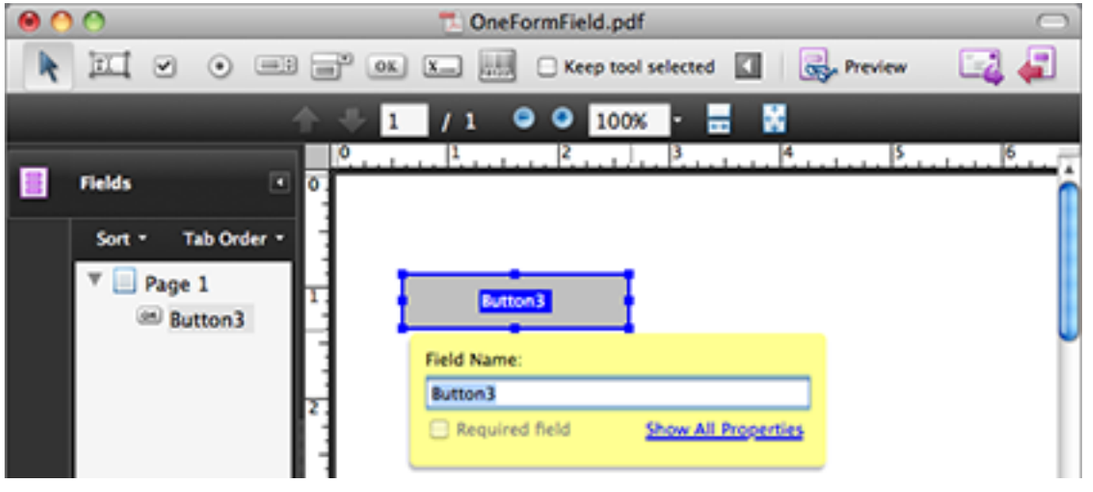
Depending on your Acrobat settings, the *Edit Fields* window will contain either a set of buttons, one for each type of form field you can add to a page, or an *Add New Field* drop-down menu that lets you select a specific type of form field.



To add a button to the PDF page, you either select *Button* from the drop-down menu or click on the button tool in the toolbar.

Either way, the cursor will turn into a crosshairs, which you may use to drag out a rectangle on the PDF page; this will be the location of your new button.

Having placed your form field, you will be looking at a page something like the one at right; the new button is represented by a blue rectangle and there is a yellow pop-up window immediately adjacent.



Drag the blue rectangle to the exact position you want for the final button; you may also drag the rectangle's handles to tweak the button's final size.

Click on the *Show All Properties* link to set the color, label, and other properties of the button; here you can also specify what should happen when you click the final button.

So, Summarizing... That's how you make a button with the Form Field toolbar:

- Select a field type from the AcroForms toolbar or drop-down menu.
- Drag out the location you want for the form field.
- Specify the properties you want for the button's appearance and behavior.
- Repeat.

And how did we create a button with LiveCycle Designer, a few pages back?

- Select a field type from the LCD palette.
- Drag out the location you want for the form field.
- Specify the properties you want to determine the button's appearance and (to an extent) behavior.
- Repeat.

Pretty darn similar, no?

Comparison

So what do either of these buy you, given that the effort to use them is much the same in both cases?

LiveCycle Designer: Aesthetics, mostly

The aesthetics of LCD are very different from the AcroForms toolbar. Using LCD has much the flavor of using a graphics package.

Also, LCD does offer some shortcuts for creating certain common items (such as a *Submit* button).

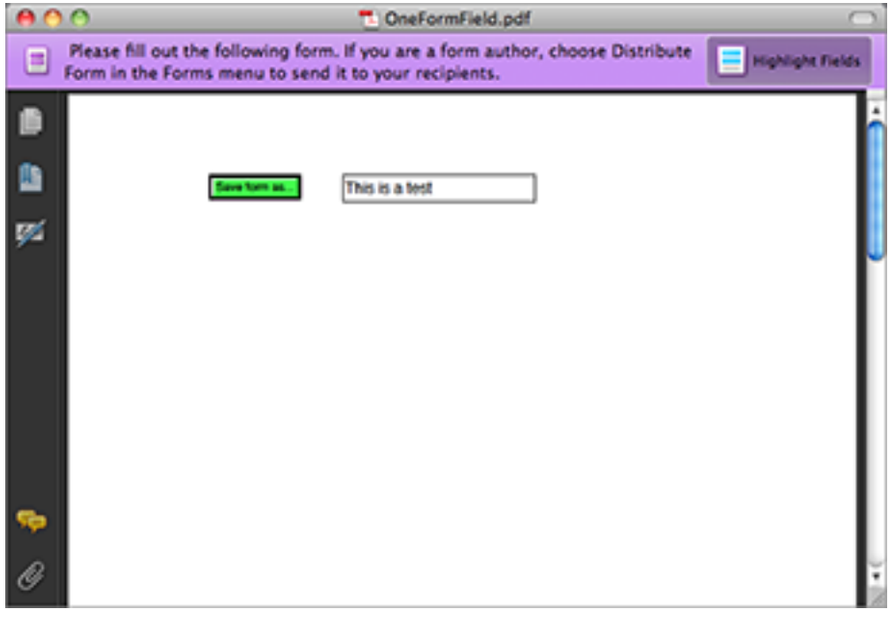
Form Field Toolbar: FIle Size

One great benefit of creating form fields using the AcroForm toolbar is file size. Forms created with the form field toolbar are smaller—often substantially smaller—than those created with LiveCycle Designer.

How much smaller we talkin' about here?

LiveCycle Designer: 23k

Consider the very simple form at right, consisting of a pushbutton and a text field. Creating this form with *LiveCycle Designer,* it occupies about 23k on the disk.



Traditional Tools: 14k

The version created with the traditional tools takes up a bit less than 14k, roughly 60% the size of the *LiveCycle* version.

Of course, this file has nothing in it *but* form fields (plus quite a lot of Acrobat-generated structural information), so this rather exaggerates the file size reduction you would get from a real form file.

Still, the point is still valid: a form created with the traditional tools will always be smaller than the *LiveCycle Designer* version.

Handwritten Form: 2½k

By the way, note that a lot of the contents of both of these PDF files is structural information inserted by Acrobat when it made the file. To give you an idea, in the *PDF 2* class, we construct a form file similar to this with hand-written PDF. That version of the file occupies about 2½k. Now, *that's* compact.

But, I digress.

So, Which is Better?

Ultimately, neither of these methods is overwhelmingly better than the other. I prefer using the AcroForms toolbar, rather than LiveCycle Designer, because of the smaller file size and because I've always found the LCD interface rather unappealing. The latter point is purely subjective, however, and other people are perfectly happy with LCD.

Also, LCD is Windows-only, so if you are doing your form design on a Mac, you are constrained to using the Form Field toolbar.

However, since there's no great benefit either way, use whichever of the two makes you happier.

PDF & XPS Nuggets

PDF Nugget

PDF & PS Drawing

PDF implements pretty much the same graphic model as PostScript: you're placing colored ink on paper to draw line art, text, and raster images. Because there's an underlying common graphic model, PDF drawing commands are similar to PostScript's.

For example, here's how you draw a filled red triangle in both languages:

PostScript

```
100 100 moveto
300 100 lineto
200 200 lineto
closepath
1 0 0 setrgbcolor
fill
```

PDF

```
100 100 m
300 100 l
200 200 l
h
1 0 0 rg
f
```

Pretty darn similar, yes?

The two sets of drawing commands start to diverge when you dig deeper. PDF, for example, maintains two separate colors at any given time, one for stroking and one for filling. (The *rg* command we used above specifically sets the fill color.)

Still, if you know PostScript and ever take a PDF class, you will find much that is familiar.

XPS Nugget:
XPS On-Disk Packaging

Last Issue's Question:

Last month I posed the question of why Microsoft chose 1/96" as their standard unit of measure?

Yolanda Palomo of Xerox forwarded me a link with the answer. Evidently, Microsoft was trying to roughly compensate for the larger distance at which most people read their computer screens, compared to printed text.

To give text roughly the same clarity on a computer screen—lower resolution than paper and held farther away—Microsoft increased the assumed pixel count from 72 dpi (standard on the Mac) to 96 dpi.

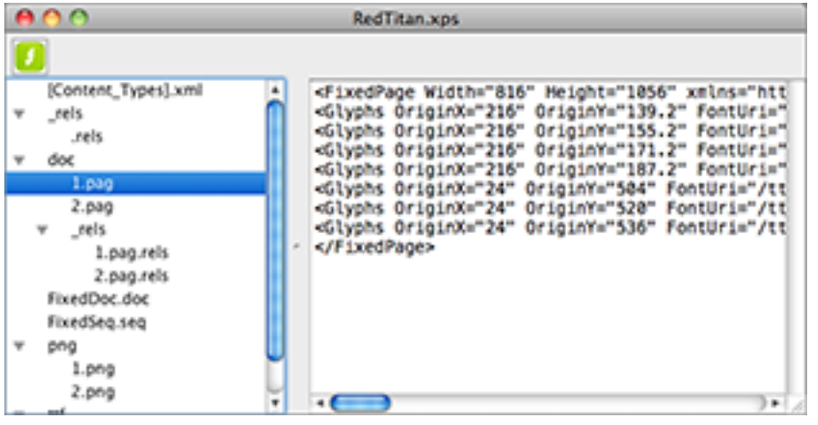
Click here to go to the online article.

Thanks, Yolanda!

When you initially start working with them, XPS files seem extraordinarily opaque when viewed with a text editor. *Somewhere* in there, you just know, are pages, fonts, drawing commands, and everything else needed for the document, but it sure isn't obvious how it's being stored among the binary gobbledegook.

The key point to understand is that an XPS file is actually a *zip* archive. If you change the suffix of an XPS file to .zip and then open the file with your favorite archive utility, you will be looking at a directory of files that together define the XPS document.

The editor that we use in the *XPS: File Contents and Structure* class does this un-zipping automatically. When you open an XPS file, the editor presents the student with a clickable list of the file components, as at right. When you make changes to the page contents and then save the file, the editor re-zips the set of component parts back into a single XPS file.



This is what makes it possible for us to do hands-on exercises in the XPS class.

By the way, the editor, *TextXPS* (for both Mac and Windows) is available, free for the downloading, from

www.acumentraining.com/software.html

It has some weak points and I'm still in the process of writing a manual for it, but as a classroom tool it works pretty well.

Schedule of Classes, August–October 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.

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

PDF Classes

PDF 1: File Content and Structure		Sep 6–9	Oct 25–28
PDF 2: Advanced File Content	Aug 9–12		
Support Engineers' PDF			Oct 7–8

PostScript Classes

PostScript Foundations		Sep 20–24	Nov 8–12
Advanced PostScript			Oct 11–14
Variable Data PostScript	Aug 16–20		
Troubleshooting PostScript			Oct 4–6

XPS Classes

XPS File Content and Structure		Sep 14–16	

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.

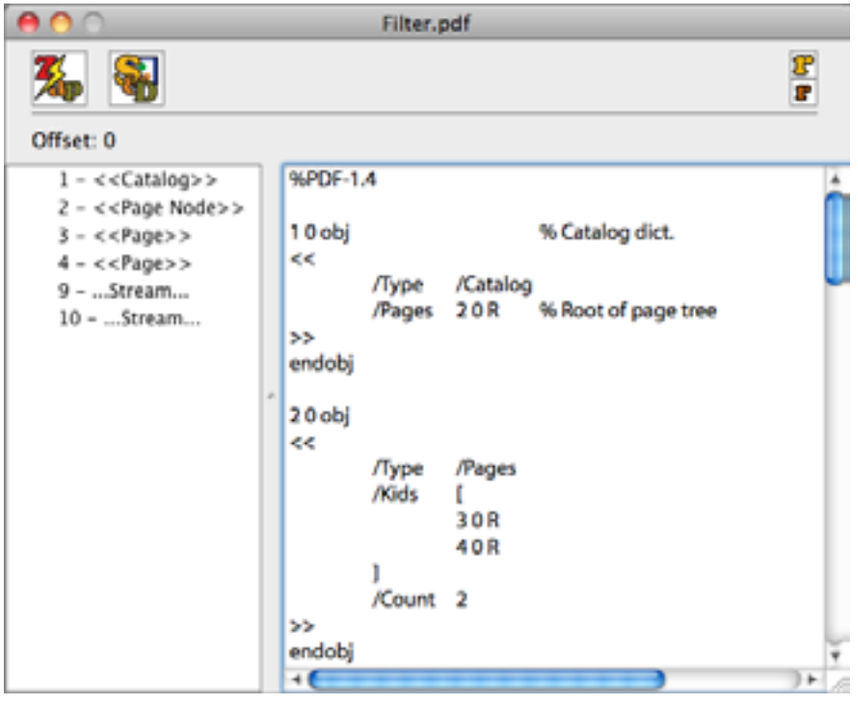
Back issues

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

What's New at Acumen Training?

Introducing *TextPDF 1.0*

At long last, the PDF classes get an editor all to themselves, *TextPDF*.



TextPDF is, at root, a text editor specific to working with PDF file. Replacing the long-in-the-tooth *AcumenEdit*, *TextPDF* lets you hand-edit PDF code and then regenerates the stream lengths and xref table so the file can be viewed in Acrobat. It also presents a clickable list of the objects in the PDF file, allowing for easy navigation among the objects making up the file.

TextPDF is free for the downloading at www.acumentraining.com/software.html.

It has a few known problems, but is, nonetheless, a useful tool for exploring a PDF file's contents.

Stay tuned for an eventual *TextPS* for the PostScript classes.

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?

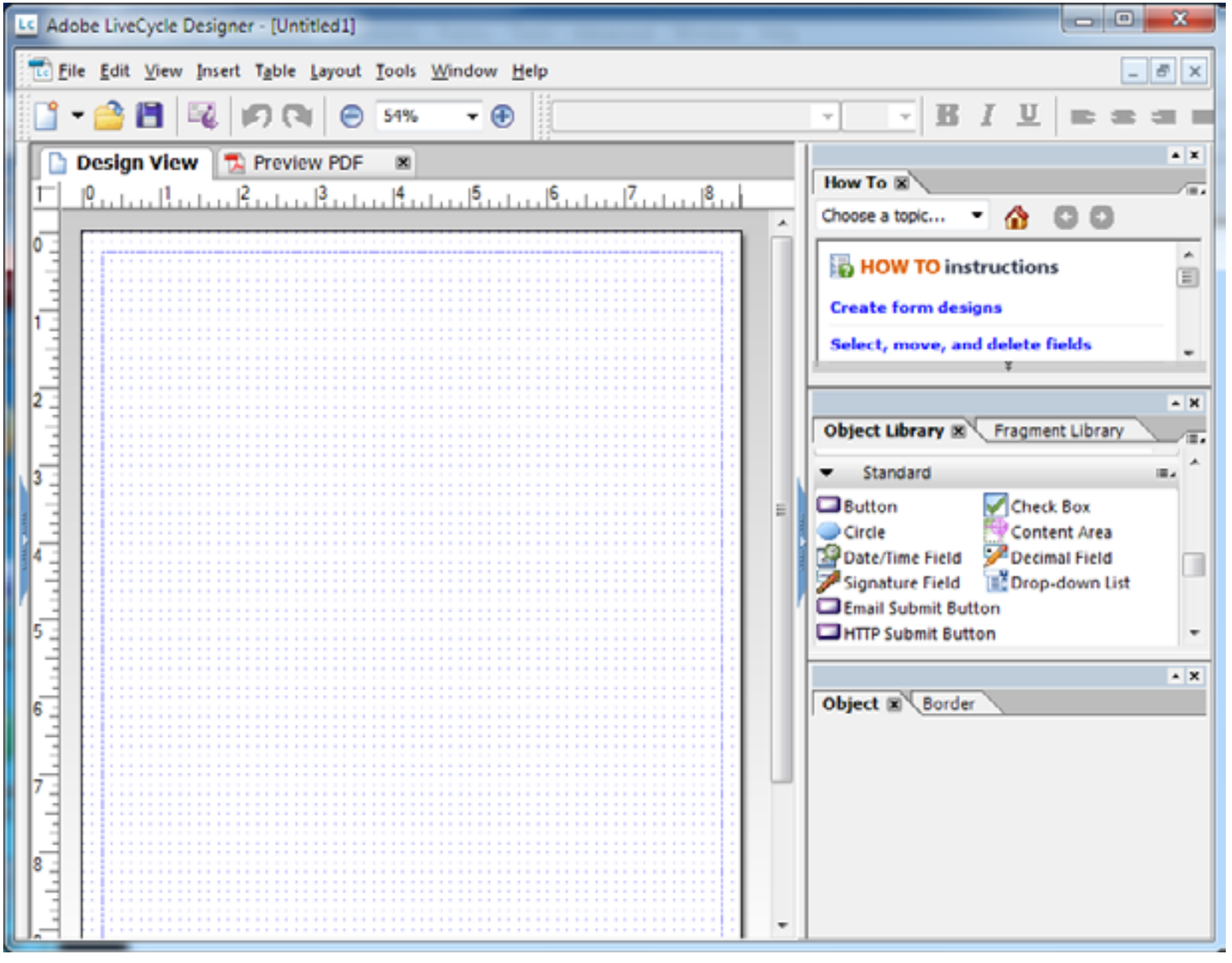
Suggestions for articles. Each Journal issue contains one article each on PostScript and Acrobat. What topics would you like me to write about?

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.)

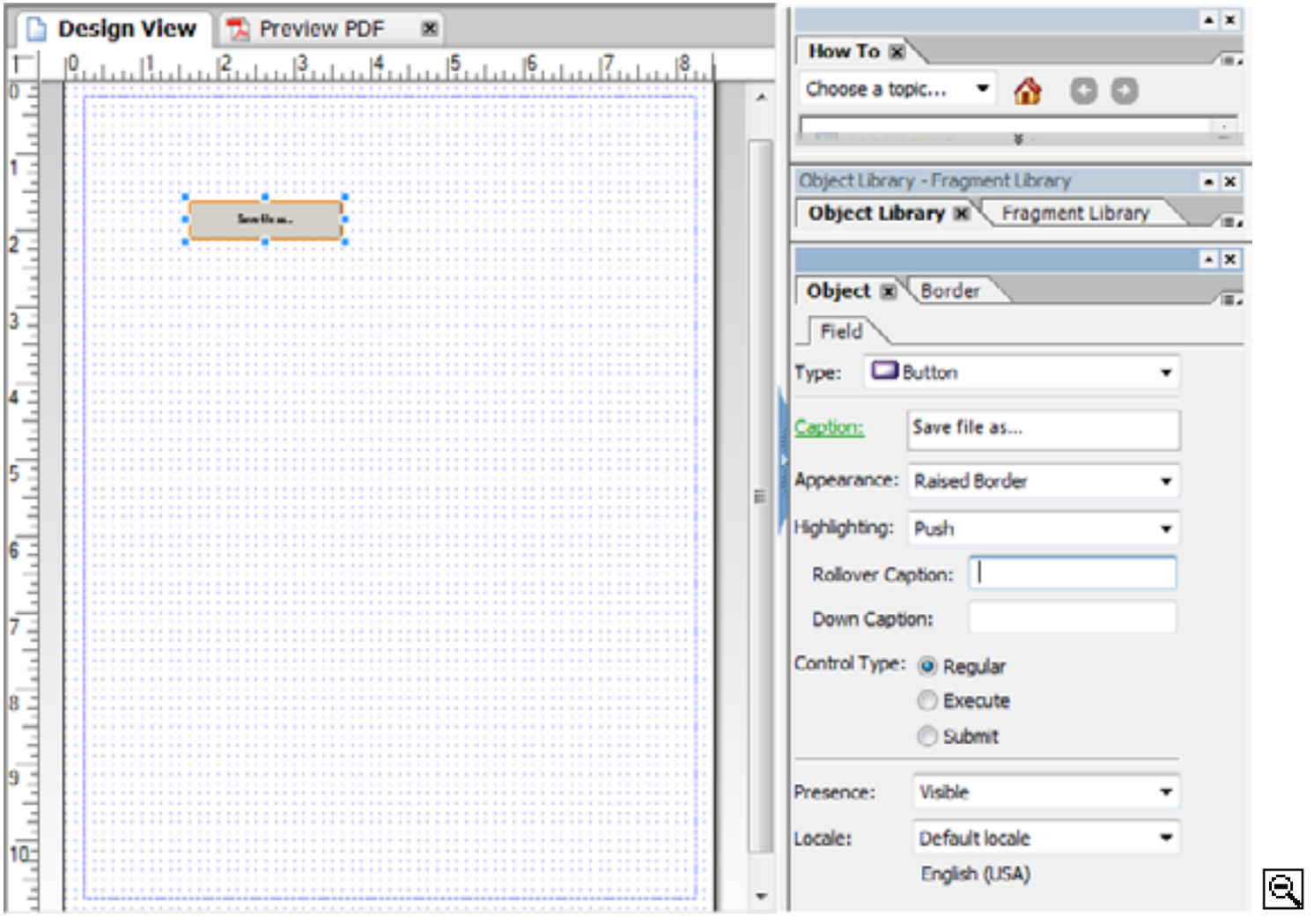
Please send any comments, questions, or problems to:

john@acumentraining.com

LiveCycle Designer Design Page



LiveCycle Designer With Button and Properties Panel



Edit Form Field Window