Table of Contents

Variable Data PostScript Basics

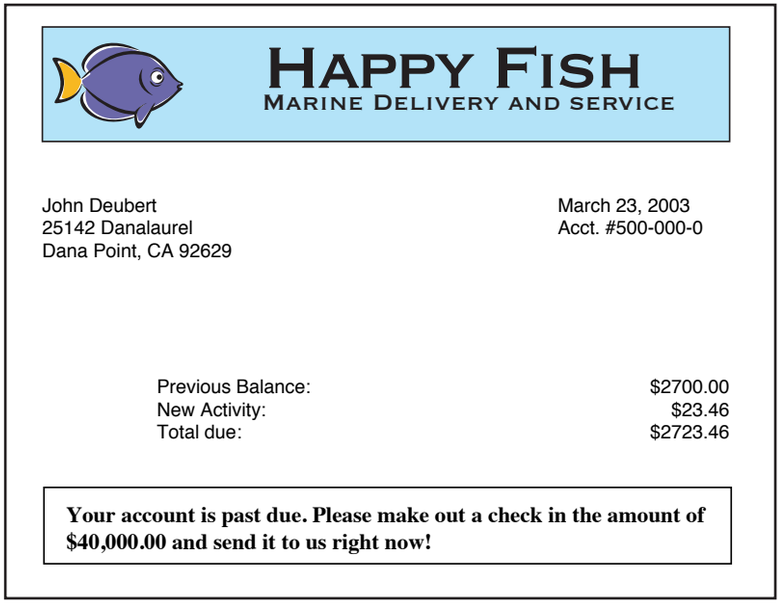
I teach a lot of PostScript classes to software engineers writing in-house programs that generate PostScript code. I enjoy teaching these folks because, since they are actually writing generating working PostScript, they get as excited as I do about such things as forms and the SubFileDecode filter, and other language features that have to do with speeding up or bullet-proofing one's output.

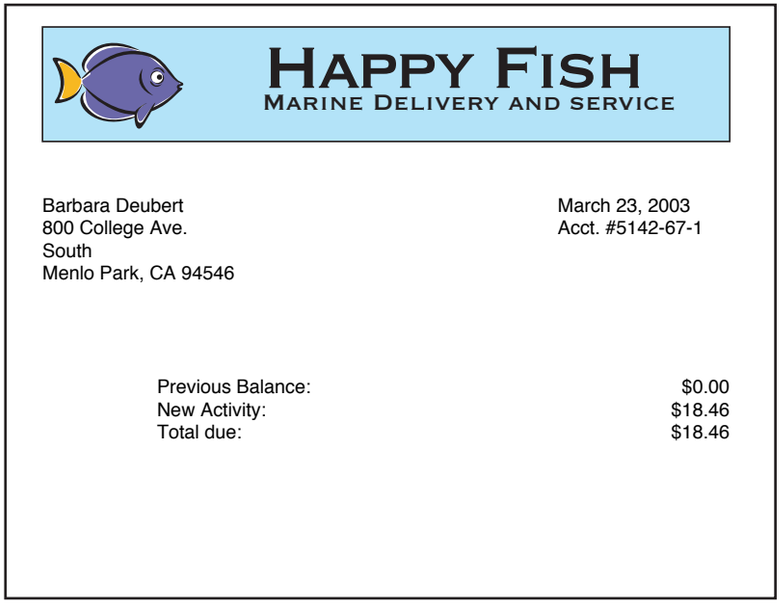
Most of the people who do handwritten PostScript code are doing variable data PostScript, that is, generating PostScript code that incorporates information drawn from a database. The resulting printed documents may be customer bills, newsletters, advertising mailers, shareholder reports, or anything else that entails printing a series of documents that incorporate customer data.

In response to some requests I've had recently, I'd like to discuss the basics of how one approaches such a task. This article can discuss only the basics of how one does this, of course; a full treatment of the topic takes a full five days to cover in the Acumen Training *Variable Data PostScript* class.

Scenario

In a fairly typical situation, a company maintains a database of customer information—name, address, outstanding balance, etc.—and wants to generate monthly bills from that data; pages from the monthly print run might look something like this:





Files on Website

The sample code presented in this article is available on the Acumen Training Resources page. Look for the file *VariableData.zip*.

The examples are taken from the *Variable Data PostScript* class, by the way.

Note the "variable" part of the variable data printing: each bill will be tailored to the data representing that particular customer. In this case, the obvious customer-specific data varies: the each customer's name, address, balance, etc. will be specific to that customer. Note that we also print a banner when there is an overdue balance telling the customer to pay the total amount or face Dire Consequences.

Presumably the customer data is being kept in a database (pick your favorite SQL flavor); when it comes time to print monthly bills, we need to create a (potentially very long) PostScript stream that will print all of the individually tailored invoices.

Let's see what that PostScript will look like.

PostScript Goals

There's no magic to variable data printing (though there often *is* tedium). We are going to use the usual *moveto, lineto, show,* and other operators to draw each page of output, incorporating each customer's data as needed. To print the customer's name, we do a *moveto* and a *show;* to draw a logo, we use whatever combination of drawing operators are needed (although if we're sophisticated, we would turn the logo into a form and draw it with *execform).*

There are three important goals we should have in mind when we design our PostScript stream (beyond putting appropriate marks on the page, that is).

Isolate Documents Within the Stream

Our PostScript stream will consist of a large number of individual documents—individual bills, share-holder reports, etc. These documents within the stream should be isolated from each other. In particular:

- No document within the stream should be dependent upon variables or other definitions created in another document within the stream.
- Documents should reclaim the memory that they allocate.
- Each document should initialize the graphics state to its desired initial value.

Wrapping each document in a *save/restore* pair will suffice for most of these, so this goal is easily met.

Isolate the Impact of PostScript Errors

Variable data print jobs are frequently *very* long; it is not unusual to have a single PostScript stream that prints 200,000 10-page, individually-tailored documents. What you don't want is for a PostScript error in document number 7 to kill the printing of the remaining 199,993. It is important to be able to isolate the effect of PostScript errors within the print stream. Ideally, when the PostScript stream is finished, you should have 198,712 printed documents and a log file indicating what went wrong with the other 1,298.

This is relatively easily accomplished with the *SubFileDecode* filter; see the June and July 2002 issues of the *Acumen Journal* for details.

Maintain a Log File

Variable data output should create a log file in which it stashes diagnostic information, in particular identifying the details of PostScript problems within the stream. Any PostScript RIP that has a hard disk available to it can create files and store information in them; your PS code should use the *stop* & *stopped* operators to catch PostScript errors and report their information in the log file.

We discuss *stop, stopped,* and PostScript file operators in both the *PostScript Foundations* and *Troubleshooting PostScript* classes. (You took one of those, didn't you?)

Drawing on the Page

As I said earlier, our PostScript stream will draw images, text, and line art on the page using the usual PostScript operators. In principle, this is straightforward.

Where things get tricky is when you need to do things like calculate line breaks for paragraphs of text, word wrap around graphic elements, align text (right, center, full justification), and other formatting of the page contents.

For example, to right justify a piece of text against the right margin, you need to calculate the length of that text, move to a point that is that distance to the left of the margin, and then print the text.

In generating your PostScript stream, you will need to decide where the formatting calculations are going to take place. There are two possibilities:

- *Driver-side formatting* - We shall do the formatting calculations in the software that generates the PostScript code.
- *Printer-side formatting* - Formatting calculations are done by procedures defined as part of the PostScript code.

From this decision will spring all of the other decisions regarding how your PostScript (and the application that generates it) looks and behaves.

Let's discuss each of these in turn.

Driver-Side Formatting

In driver-side formatting, all of the calculations necessary to setting text and placing graphics on the page will be done by the application that is generating the PostScript code. This application—written in C++, Java, .Net, or whatever your favorite programming poison is—will need to calculate, for example, the starting position on the page of every snippet of printed text. It can then place into the PostScript code calls to *moveto* and *show* appropriate to each snippet.

The PostScript code might look something like this:

```
/ms           % (str) x y => ---
{ moveto show } bind def
...
%%Page: 1 1
(Mary Antimony) 450 700 ms
(2157 Mousedropping Lane) 450 680 ms
(Dead Owl Crossing, MS) 450 660 ms
(The party of the first part, having) 100 600 ms
(consulted with the party of the second) 100 580 ms
(part, must part company with the party) 100 560 ms
(of which they are a part.) 100 540 ms
...
```

The significant feature of the above code is that it consists of very simple PostScript. It has a simple prolog (in our case consisting of a single procedure) and a script that is very close to Red-Book-simple PostScript code. All of the numbers were calculated in the generating application and simply placed into the PostScript.

This is what characterizes driver-formatted PostScript code:

- A short, simple prolog that defines a minimal set of utility procedures.
- A relatively verbose script that is very close to straight moveto-lineto-show level PostScript code.

Benefits Because the prolog is relatively simple, it will contribute less to the overall size of the print stream; this makes a driver-formatted approach more appropriate for situations where most print jobs are short (one to three pages, perhaps).

Disdvantage The only real disadvantage to driver-formatted PostScript output is that the driver must have all of the information it will need to calculate positions and sizes. In particular, the driver needs width and kerning tables for any fonts that will be used in the output so that it can calculate the width of a particular string of characters.

The Macintosh and Windows PostScript drivers take this approach, since the average print job on these systems is quite short and, furthermore, being system-level drivers, they have access to all of the system's fonts' metric information.

Printer-Side Formatting

The alternative to doing the formatting calculations in the driver is to do those very same calculations in PostScript (it *is* a programming language, after all). In this case, we shall have a fairly long, elaborate prolog; it will contain procedures that do all of the computation associated with laying out the page, incorporating the variable data.

The script, on the other hand, can be very simple. At one extreme, the script may consist solely of text-formatted data appended to the prolog, as below.

(Don't try to follow this code in detail; much has been omitted to save some space and, in any case, this isn't an article exactly how to do this kind of page layout.)

```
/scratchString 100 string def
/lm 72 def
/rm 372 def
/leading 16 def
/theFont /Helvetica findfont 14 scalefont def

/ReadLineOfData             % --- => (data)
{ currentfile //scratchString readline pop } bind def

/PrintWord        % (word) => ---
{    dup stringwidth pop        % Get the string's width
     ... omitted for brevity
} bind def

/newline                    % --- => ---
{ lm currentpoint exch pop leading sub moveto } bind def

/PrintParagraph             % (paragraph) => ---
{    ( )
     ... omitted for brevity
} bind def

/rightshow                  % (str) => ---
{ dup stringwidth pop neg 0 rmoveto show } bind def

/centershow                 % (str) => ---
{ dup stringwidth pop -2 div 0 rmoveto show } bind def

/fixedText
(The party of the first part, having consulted with the \
party of the second part, must part company with the party of \
```

```
which they are a part) def

/DoPage
{    //theFont setfont
     72 700 moveto

     ReadLineOfData          % Read the customer's name
     show
     newline

     ReadLineOfData          % Read the customer's address
     show
     newline

     ReadLineOfData          % Read the customer's city-state-zip
     show
     newline
     newline
     //fixedText PrintParagraph

     showpage
} bind def


/DoRun
{    ReadLineOfData cvi
     { DoPage }
     repeat
} bind def

%%EndProlog
```

```
% Invoke the DoRun procedure, followed by raw data
DoRun
2
Mary Antimony
2157 Mousedropping Lane
Dead Owl Crossing, MS
B. Zelbub
90876 Hotfoot Lane
Bowels of Earth, PA
```

Note that all of the calculations (simple, in this case) placing the text on the page is done in the PostScript procedures. The Prolog here is a fixed wad of PostScript code, the Script is just the raw data dumped from the database.

Benefits

Better for Long Print Runs

Printer-side formatting results in a relatively large, complex prolog, but a correspondingly simple script; at one extreme, as in the above example, the scripts consists of just the raw data appended to the fixed script code. This makes printer-side formatting most appropriate for very long print jobs; when you are printing hundreds of thousands of pages (common in variable data printing), the large prolog is still a vanishingly tiny part of the whole stream, whereas the script is *very* much more compact than results from driver-side formatting. Most people doing variable data printing eventually converge on printer-side formatting.

Easier Text Setting

It is generally easier to write PostScript code to line breaks, text alignment, etc., than it is to write driver code to do the same thing because the PostScript interpreter has guaranteed access to font character widths and other metrics. The text calculations are relatively simple using *stringwidth* and its brethren.

One exception: if you are doing pair kerning, you get no particular benefit from a printer-side approach; PostScript has no direct support from pair kerning, so you'll end up having to consult a kerning table, as you would with driver-side formatting.

Easier to Write the Driver

The PostScript driver for printer-side output is generally very simple; at its simplest, the driver need only concatenate a fixed, pre-written prolog and the records from the database and send the lot to the PostScript device. The can make writing the PostScript output part of the programming task very quick, assuming you have someone who knows PostScript well.

Which brings us to…

Disadvantages

Requires Better PostScript

The only significant disadvantage to printer-side formatting is that you must be a much better PostScript programmer to do it. A simple knowledge of *moveto, lineto, stroke , fill,* etc. will not suffice. As an indication, my *Variable Data PostScript* course requires the *PostScript Foundations* course as a prerequisite; and then it provides about half the topics in the *Advanced PostScript* course in addition. (If you're curious about the list of topics in the *Variable Data PostScript*—and who wouldn't be?—you can find a course description on the Acumen Training website.)

There's a *lot* to discuss

As you might imagine, variable data PostScript printing is a huge topic and we can't even be said to have scratched the surface; rather, we have viewed the surface from a medium distance. If the prospect is a bit daunting from this distance, don't be discouraged; it isn't that hard.

Just, as I said earlier, tedious.

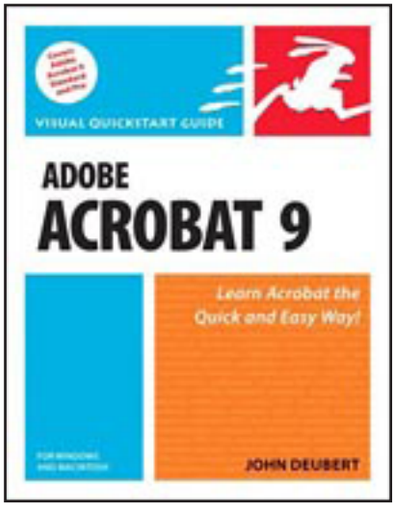
Also, strangely fun.

PDF Portfolios

File on Website

The portfolio used as an example in this article is available on the Acumen Training Resources page. Look for the file *Portfolio.zip*.

Acrobat 9 introduces a new feature called PDF Portfolios that has turned out to be surprisingly (to me, anyway) useful. Their main problem is that upon first hearing of them, they don't seem to solve any real-world problems and so don't really have a reason for existing.

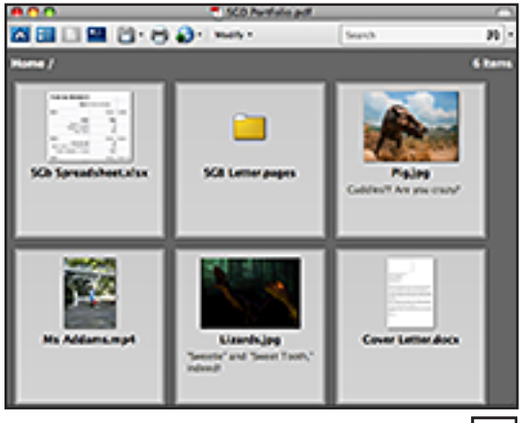


Not at all true! Once properly understood, PDF portfolios are very useful and extremely easy to create.

This month we shall discuss the features and abilities of PDF portfolios, talking only briefly about how to actually create and modify them. For a full discussion, see Acrobat Help or the excellently-written *Acrobat 9 Visual Quickstart Guide*.

E Pluribus Unum

A PDF portfolio packages one or (usually) more files into a single PDF file. The intent is to combine the files into a coherent collection of documents that must conveniently travel around together.



For example, the portfolio illustrated at near right (and available on the Acumen Training web site) contains documents associated with a fictitious lawsuit. (Yes, I know these are not real lawsuit documents; let's pretend, shall we?)

The files—in the example these are a PDF–format cover letter, an Excel spreadsheet, two images, an mp4 movie, and a Macintosh *Pages* file—are presented to the user in a grid. Double-clicking a

file causes Acrobat to display that file, either within the portfolio window or by opening the file with the original application, depending on the file type.

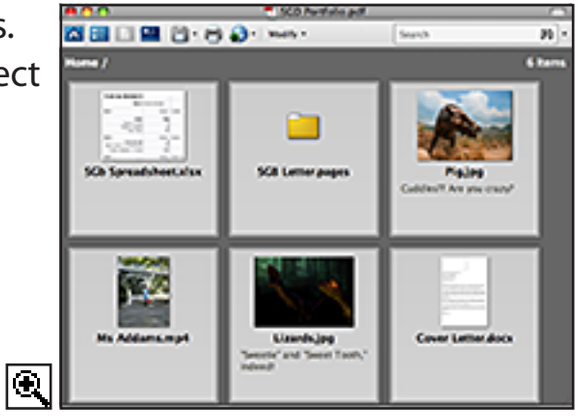
Customization

The primary purpose of a portfolio is the bundling of several documents together and the control of the presentation of those documents. Let's talk a bit about the kind of customization you can apply to the file's appearance.

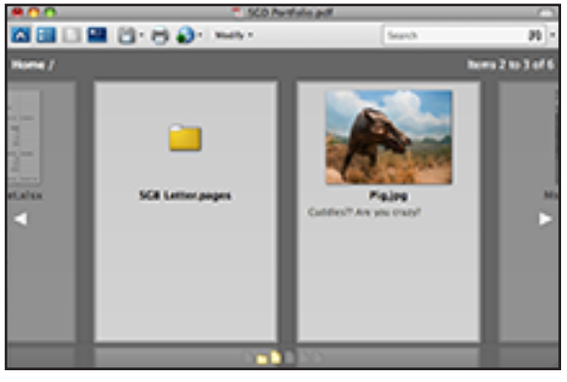
Again, here we are going to discuss what you can do, not how to do it; we'll see how it's all done a little later in the article.

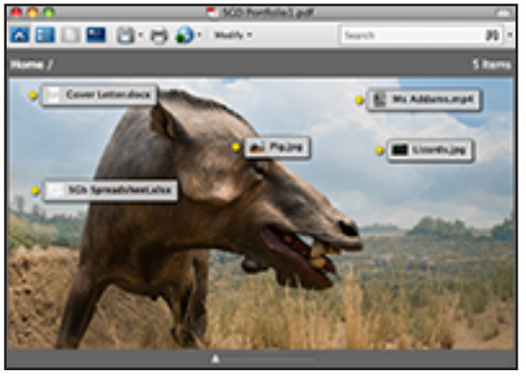
Home Page Appearance

A portfolio's *home page* is the initial presentation of its component files. By default, this is a grid arrangement, as at right. You can, however, select among four different initial arrangements:



- *Grid,* as described
- *Revolve,* in which the documents are presented as though mounted on a revolving wheel; this is similar to the "Cover Flow" presentation used by many Apple products.
- *Slide,* presenting the portfolio's documents side-by-side.





- *Background*, in which the component documents are presented as icons against a background picture (illustrated on the previous page).

For my purposes, the grid is usually the most useful layout, but the others certainly allow you to assemble a polished presentation of the portfolio's contents.

Comments

You may attach a comment to each item in the portfolio; this comment will be displayed with the item's preview in the home page.

Header

You may specify a header for your portfolio. This is a combination of artwork, image, and text that appear at the top of each page displayed in the portfolio, including the home page. At right, the header is the company logo (the dog) and the snippet of text associated with it.

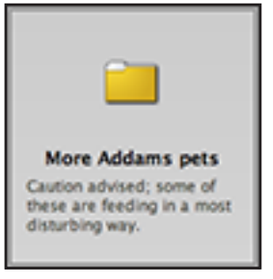
Welcome Page

Portfolios are all about presentation; the Home page allows the user to see all of the files that make up the portfolio, but this may not be the first thing that you want the user to see upon opening the portfolio. Perhaps you want to initially present a summary of the portfolio's purpose; perhaps instructions for how to view the portfolio contents; perhaps you simply want to display an initial splash screen with your company logo.

Acrobat allows this by letting you specify a "welcome page" for the portfolio. This is a combination of graphics, text, and image that will be displayed upon the opening of the portfolio, as at right. You have considerable control over the layout and appearance of the cover page, though the its background is invariably translucent, allowing the home page to vaguely show through.

Folders

A portfolio can contain folders that, when double-clicked, reveal other files and folders. This allows you to organize the contents of your portfolio in whatever manner you please. Just drag a folder into the portfolio as you would a file.



Creating a Portfolio

There are a lot of parameters that dictate the exact appearance and behavior of a portfolio; the procedure for creating them can be involved. Furthermore, in keeping with Adobe's design philosophy, there are many ways of creating a portfolio, allowing you to pick the method that suits you best.

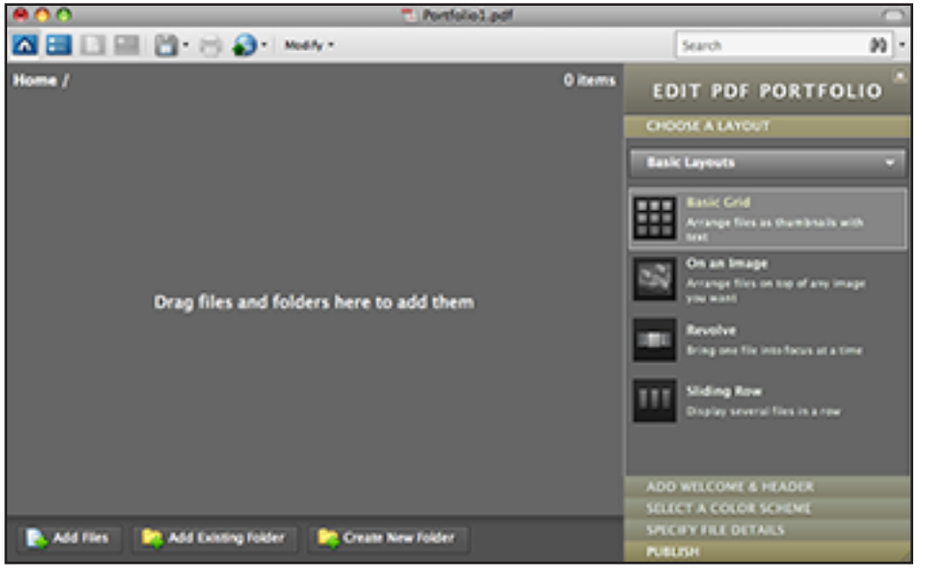
Here we are going to present what seems to me to be the easiest method to create a portfolio and discuss the basics of that technique. I'll leave you to explore the other methods and all of the possible options at your leisure. (You do have leisure, don't you?) The Acrobat 9 Visual Quickstart Guide would be a very good place to start that exploration. (Have I mentioned the Visual Quickstart Guide? Great book.)

The Portfolio Design Window

You start creating a portfolio by selecting *File>Create PDF Portfolio*. Acrobat presents you with the Portfolio Design Window, containing controls and panes specific to the task.

The simplest way to add files and folders to the portfolio is to simply drag them from the desktop into the big, gray section of the window.

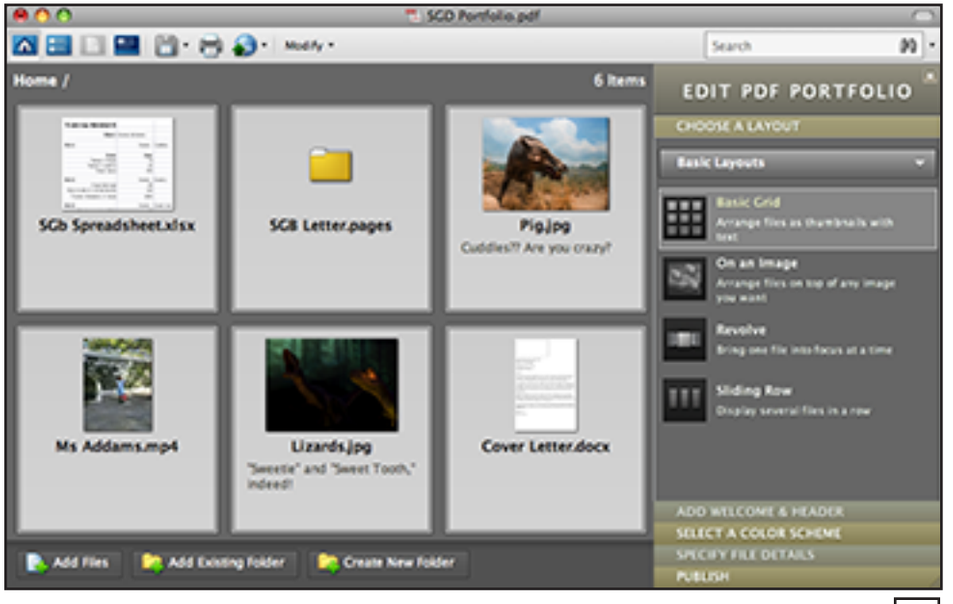
As you do so, the window will fill with previews of the component files and folders, as below right.



The Appearance Pane

The right side of the Create Portfolio window is occupied by what I call, for want of a better term, the *Appearance Pane*. This pane (beneath the "Edit PDF Portfolio" title at right) is actually a set of five "subpanes" occupying the same area of the window; clicking on the title bar of a subpane displays that subpane's contents. Together these subpanes let you specify the appearance and behavior of the portfolio.

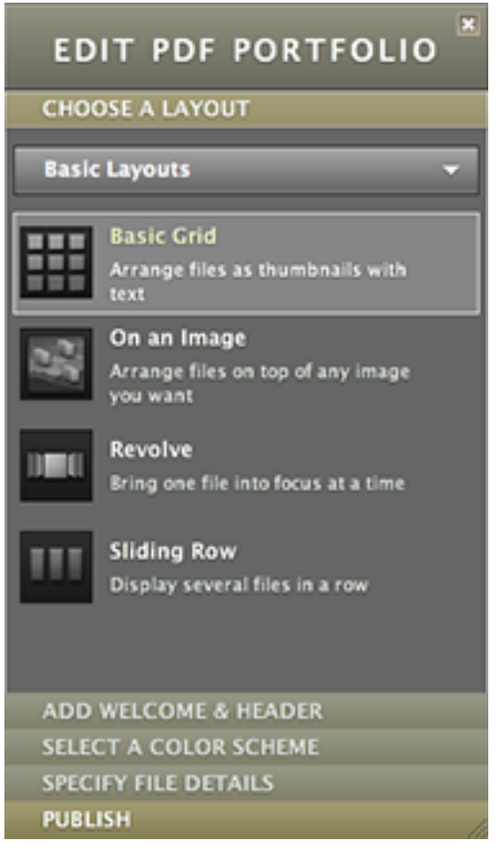
Here we shall discuss the purpose of each subpane only briefly; their use is not particularly difficult and easily discovered with a bit of experimentation.



Let's look at these subpanes in the order, top to bottom, that they appear in the Appearance pane; this is a reasonable order in which to use these subpanes, as well.

Layout Subpane

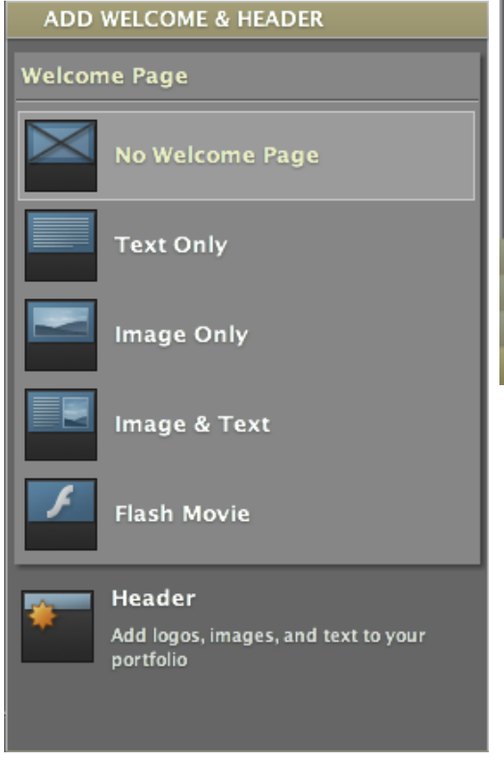
This subpane lets you specify how the component files should be presented to the user; the files may be presented as a grid, as icons on an image, as revolving or sliding previews, as we described earlier.



Add Welcome & Header

This subpane allows you to add a Welcome pane or a header to the portfolio. As we discussed earlier:

- A Welcome pane is a pane that is presented to the user upon initially opening the portfolio. It may contain a combination of text, line art, or an image.
- A Header is a small pane, containing text, line art, or an image, that appears at the top of every page in the portfolio.



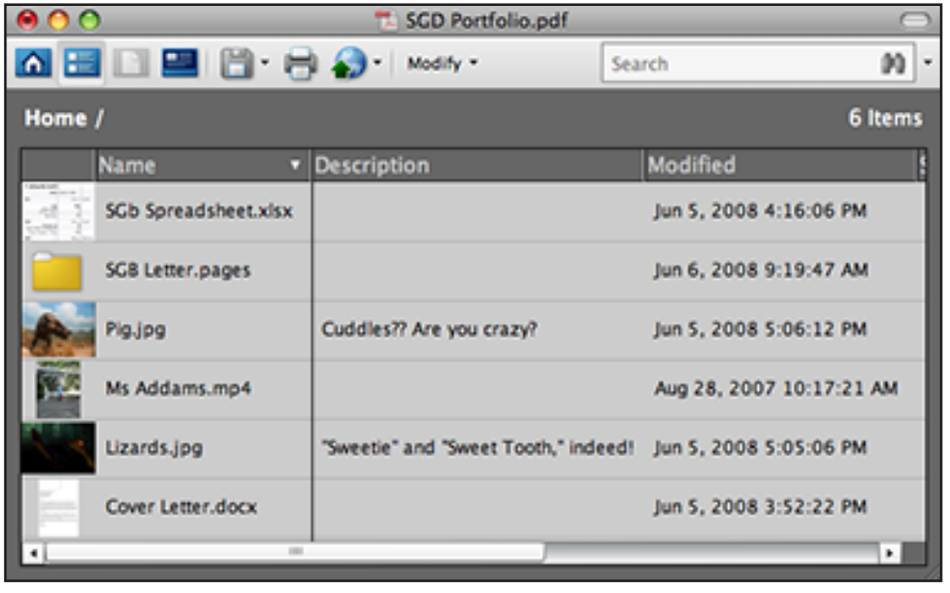
Color Scheme

The next subpane allows you to select a color scheme for your portfolio. This specifies the color for everything the User sees in the portfolio windows except the actual content of the component files.

You may select one of the predefined color schemes or devise your own custom palette. Creating a custom scheme is mildly tedious, but it is the only way you can create a portfolio appearance based on your corporate colors, your favorite team colors, or the exact teletubby hues.

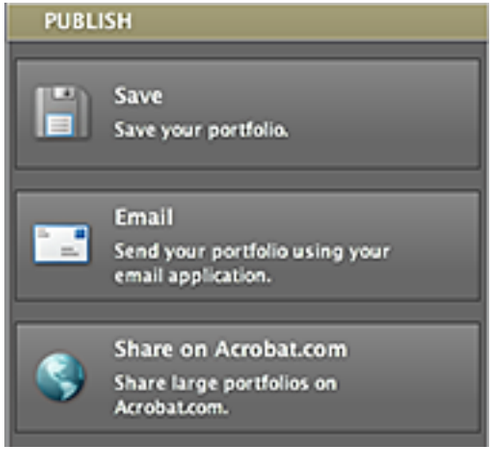
File List Contents

One of the toolbar buttons available to the user when viewing a portfolio yields a list of the files in the portfolio (as below).



The next Appearance subpane lets you specify the exact information that should appear for each file in this list.

Publish The final appearance subpane, properly speaking, has nothing to do with the portfolio's appearance. Rather, it lets you specify what to do with the portfolio now that you've created it. Your choices are:



- *Save* - Save the portfolio to disk; this is probably your most common choice.
- *Email* - Send the portfolio to someone as an email attachment.
- *Share* – Share the portfolio on Acrobat.com. (See previous issue of the *Acumen Journal* to learn more about Acrobat.com.)

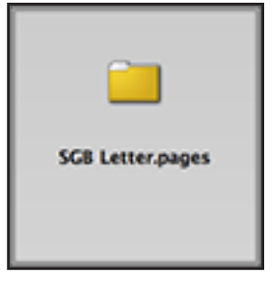
Once you have selected one of these options, you are finished creating the portfolio; you may now go live your life.

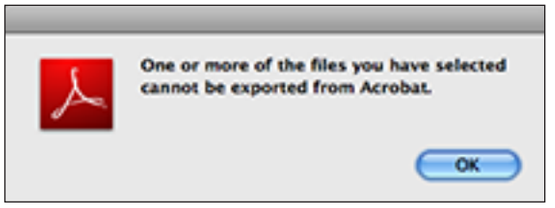
Limitations

There are only two limitations to distributing a collection of documents as a PDF portfolio, neither of them fatal:

MacOS X Packages Many files in Mac OS X—notably application executable files—are actually specially-handled folders called *packages.* Mac OS X recognizes these folders as special and presents them as single items on the disk; their true nature is hidden from casual inspection. Many applications, such as Apple's *Pages* word processor, save their documents as packages.

Unfortunately, the Acrobat portfolio mechanism knows nothing about OS X packages; these will appear on the home page as folders. This article's sample portfolio contains a document file created with Apple's *Pages,* which appears in the portfolio's home page as a folder, as at right.



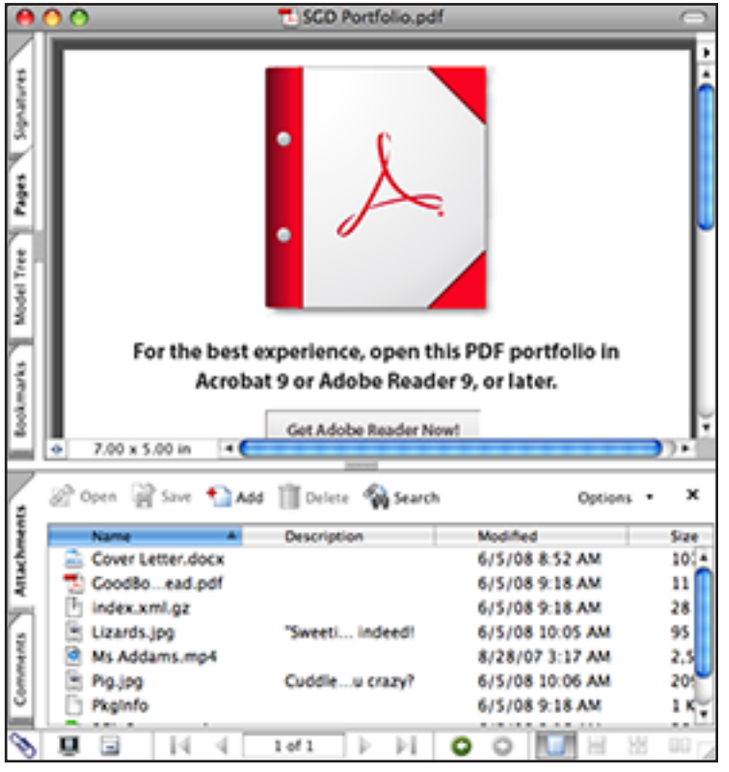


The problem is that there is no way to view this Pages document. Acrobat will not even let you save the folder to disk so that you could possibly reopen it in the Pages application; you get the alert at right when you try to do so.

I am hoping that this weakness will be fixed in a future update to Acrobat 9; for the moment, any documents you create with most Apple applications will need to be exported to PDF before embedding in a Portfolio. I should say that's probably a good idea, anyway; as is the case with nearly all PDF files, a portfolio's contents should be considered read-only and therefore generally shouldn't be application-specific files.

Compatibility

Portfolios can be properly viewed only in Acrobat 9 (or later, presumably). The portfolio can be usefully opened in earlier versions of Acrobat: it will open as a package (the ancestral version of a portfolio) in Acrobat 8; in Acrobat 7 and earlier, the portfolio contents appear as a set of attachments that the user may extract to disk.



But…why?

Why would you want to use these? The most obvious application is to proposals, information packages, and other business presentations that are made up of several documents of disparate types. A portfolio lets you bundle into a single file the images, PDF documents, spreadsheets, movies, and other documents that make up the package. You can control the appearance and presentation to the user and, therefore, the impact the document will have on the person who looks at it.

I have been experimenting with using them for such things as invoices (which include scanned receipts), print jobs to my local print shop (using the cover page for printing instructions), and an electronic brochure of my courses. Reaction has been mixed, I admit: people who have Adobe Reader 9 enjoy working with the portfolio; they find it is a very useful way of receiving a collection of files. People with earlier versions of Acrobat are initially puzzled and don't quite know what to do with the file.

I expect that portfolios will be increasingly useful as Acrobat 9 (and Adobe Reader 9) become more common in the user community.

A Call For Opinions

I'd like to hear about your experiences with Portfolios. Have you distributed any document collections as portfolios? Did people like them? What were the best and worst parts of the experience?

I'll report on your comments in a future article.

Schedule of Classes, December 2008 – February 2009

Following are the dates of Acumen Training's upcoming PostScript and pdf classes. Clicking on a class name will take you to the description of that class on the Acumen training website. These classes are taught in Orange County, California and on-site at corporate sites world-wide. See the Acumen Training web site for more information.

PDF Courses

PDF 1: File Content and Structure	Dec 15–18		Feb 9–12
PDF 2: Advanced File Content			
Support Engineers' PDF	Dec 3–4	Jan 29–30	

PostScript Courses

PostScript Foundations	Dec 8–12		Feb 2–5
Advanced PostScript		Jan 12–15	
Variable Data PostScript		Jan 19–22	
Troubleshooting PostScript		Jan 26–28	

Course Fee Classes cost $2,000 per student, except for *Troubleshooting PostScript* and *Support Engineers' PDF,* which are $1,500 per student. There is a 10% discount for signing up three or more students. 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.

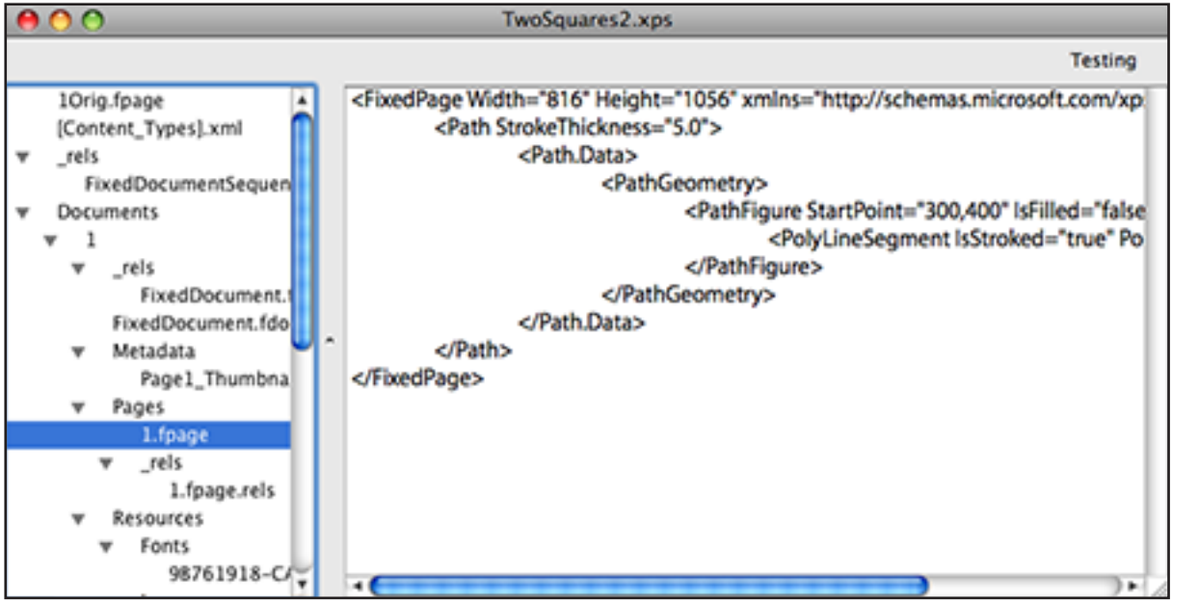
Back issues

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

What's New at Acumen Training?

XPS Editor *TextXPS* Ready for Class

The *XPS File Structure and Contents* class continues to progress. The specialized text editor that the class will use, provisionally called *TextXPS,* is functioning well enough for me to use it in developing class exercises and examples. The screenshot at right gives you an idea of what the interface looks like.



No, it's not complete, let alone ready to post on the Acumen Training website, but it does now have the minimum functionality needed for the class.

It will be available in both Mac and Windows flavors, of course, like its older brother, *Acumen Editor.*

The *XPS File Structure and Contents* class will be ready in February.

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 inexplicably make you think of that lemmings have the right idea?

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

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

Please send any comments, questions, or problems to:

john@acumentraining.com

Portfolio Home Page

SGD Portfolio.pdf

Modify

Search

Home /

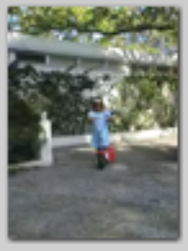
6 Items

SGb Spreadsheet.xlsx

SGB Letter.pages

Pig.jpg

Cuddles?? Are you crazy?



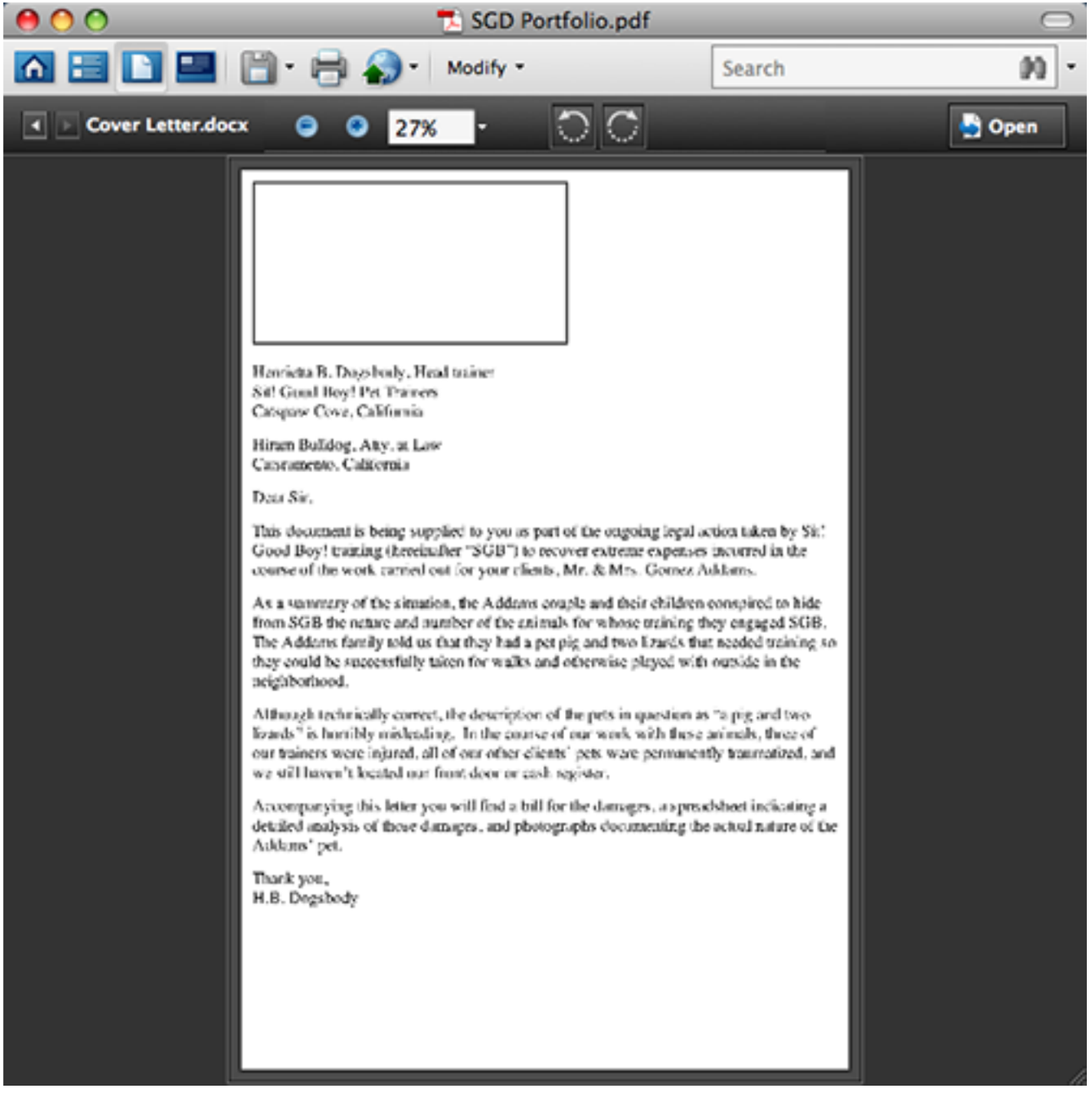
Ms Addams.mp4

Lizards.jpg

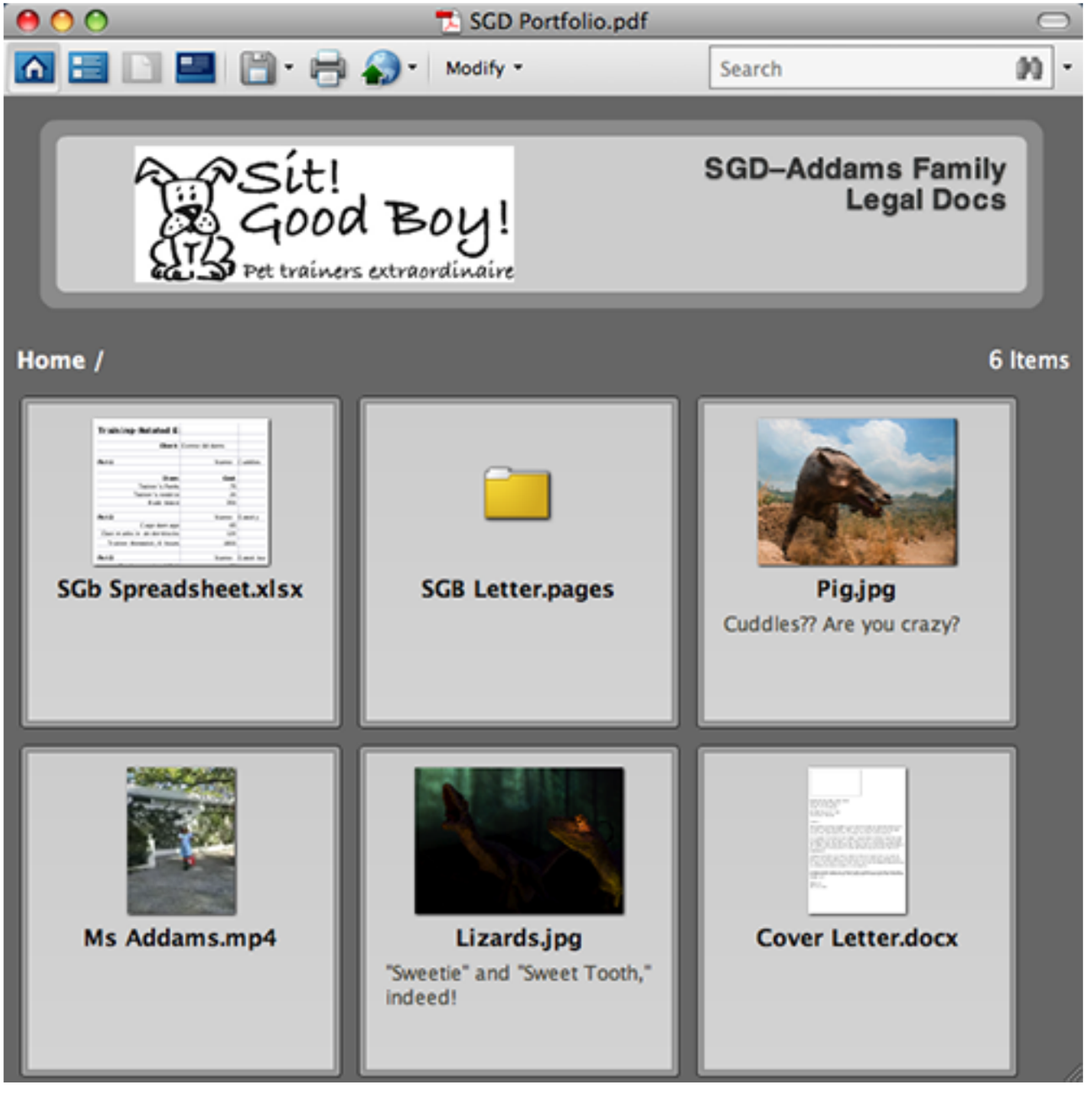
"Sweetie" and "Sweet Tooth," indeed!

Cover Letter.docx

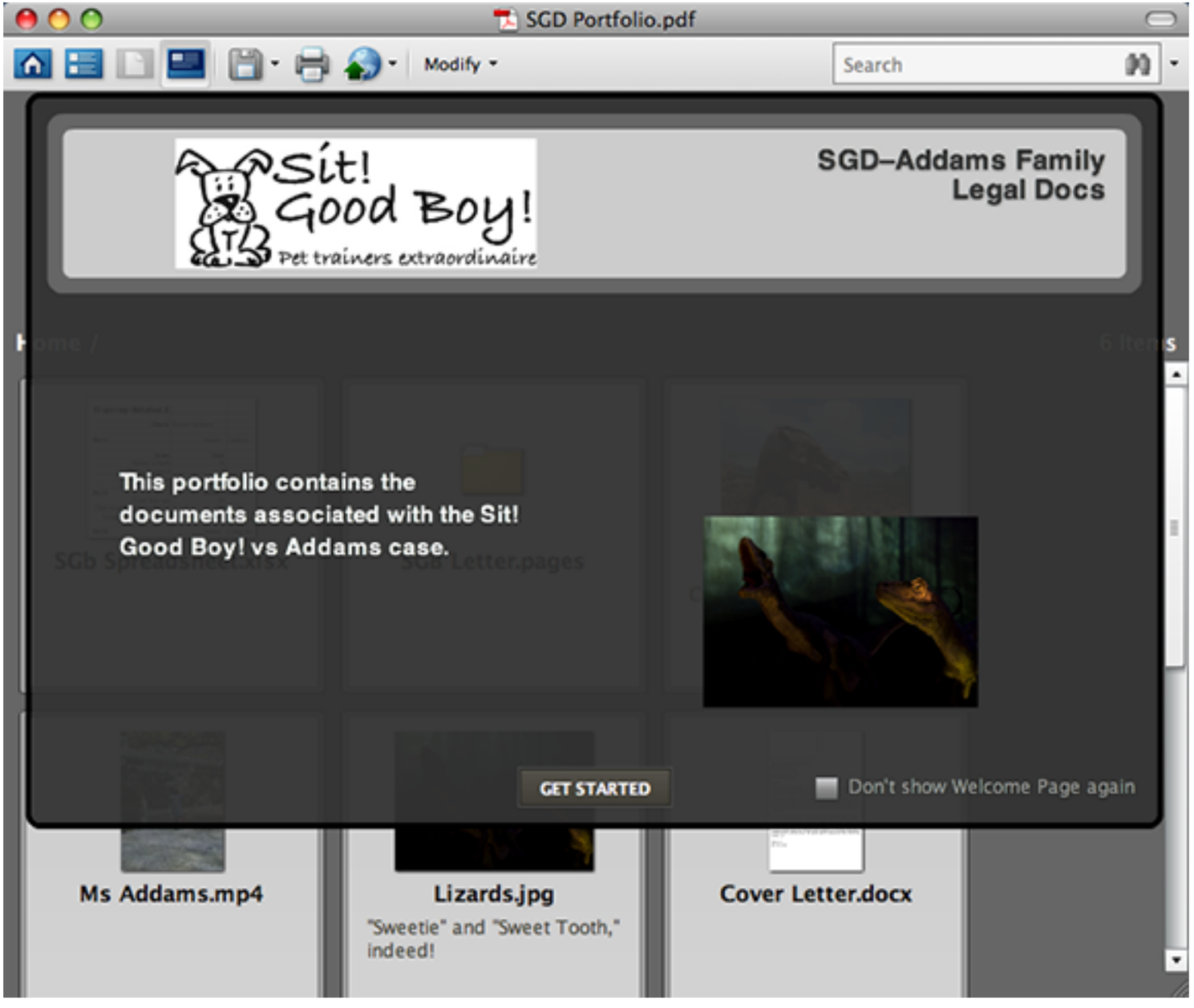
Portfolio Document Contents



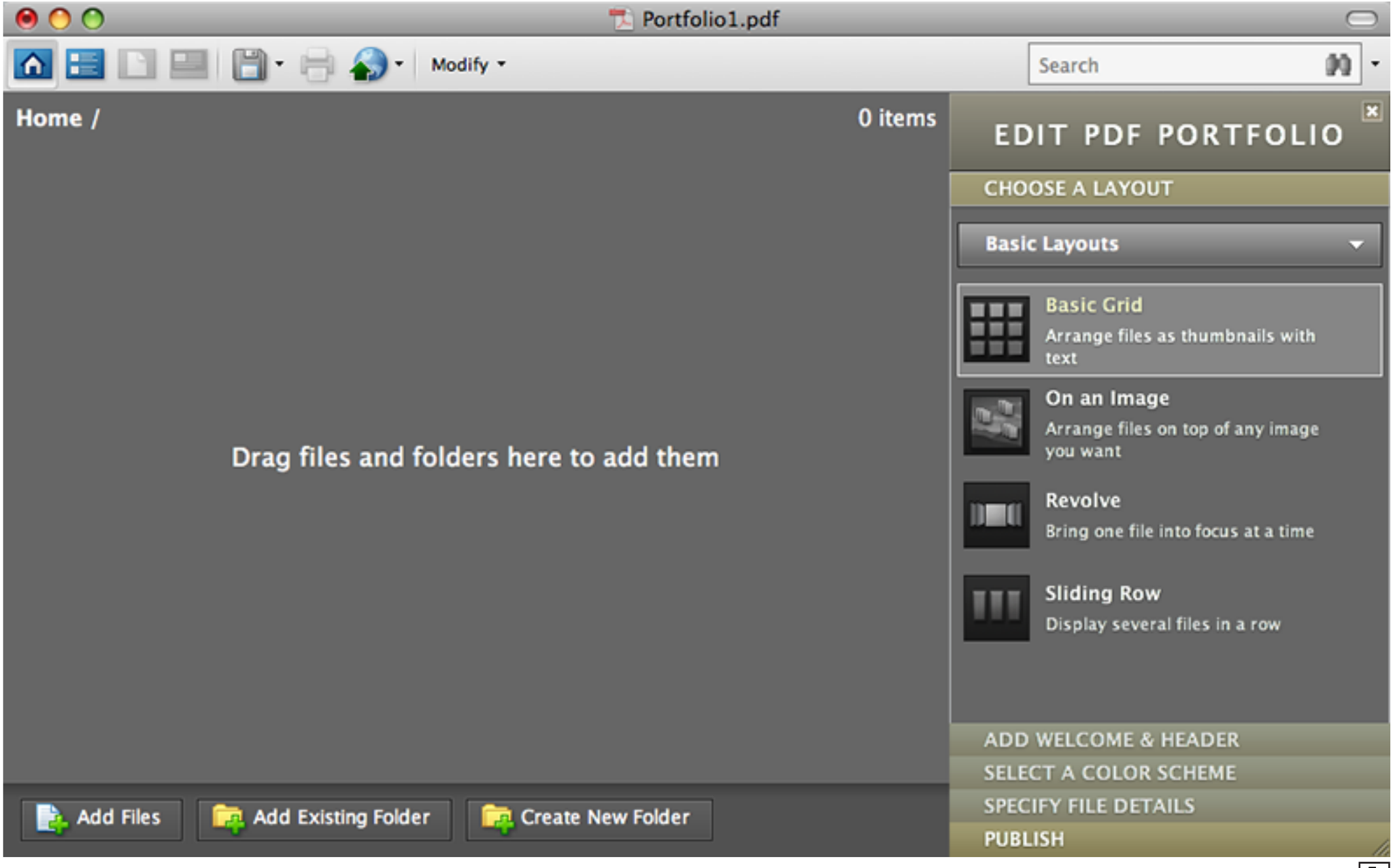
Portfolio Home Page With Header



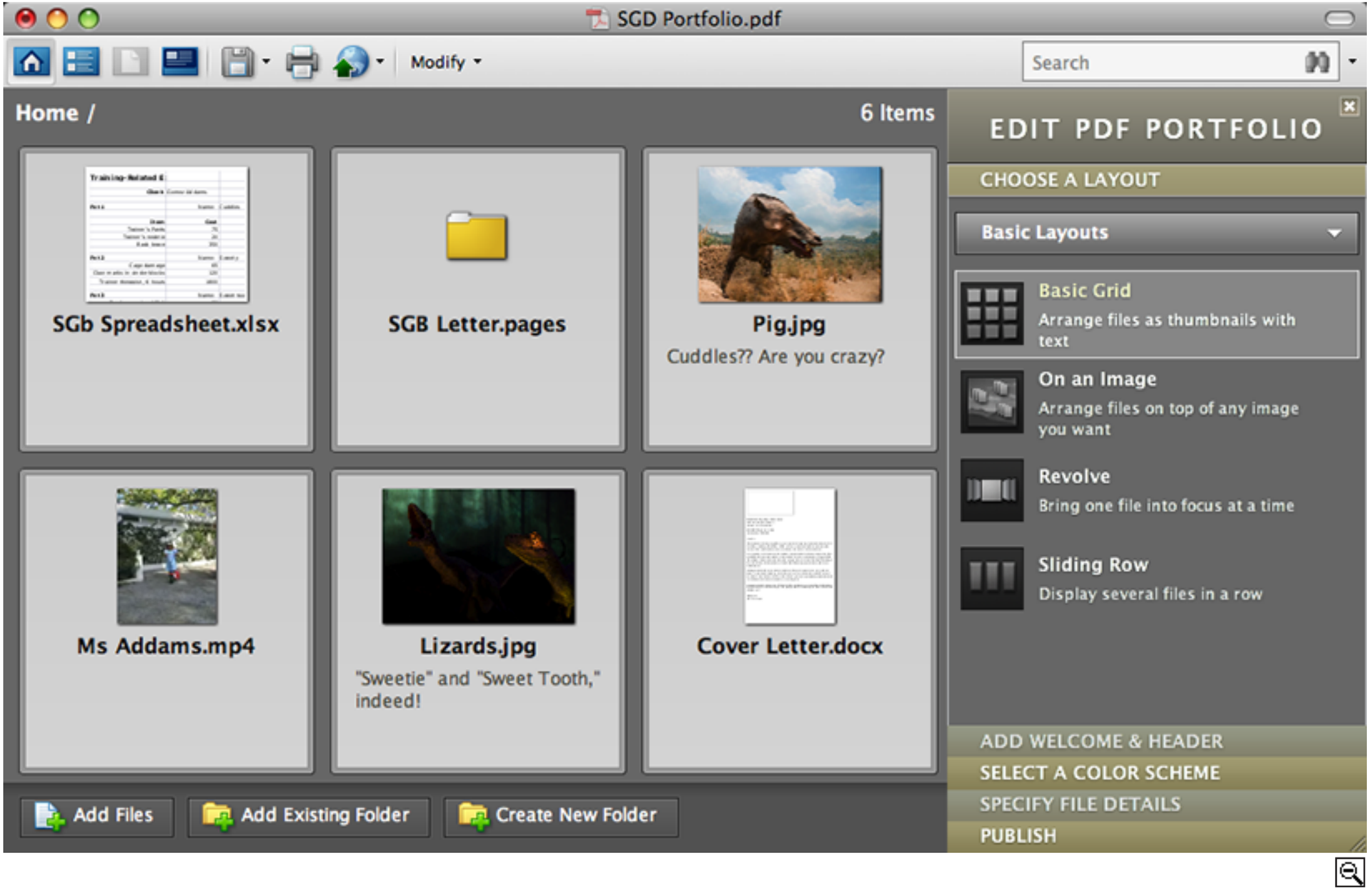
Welcome Page



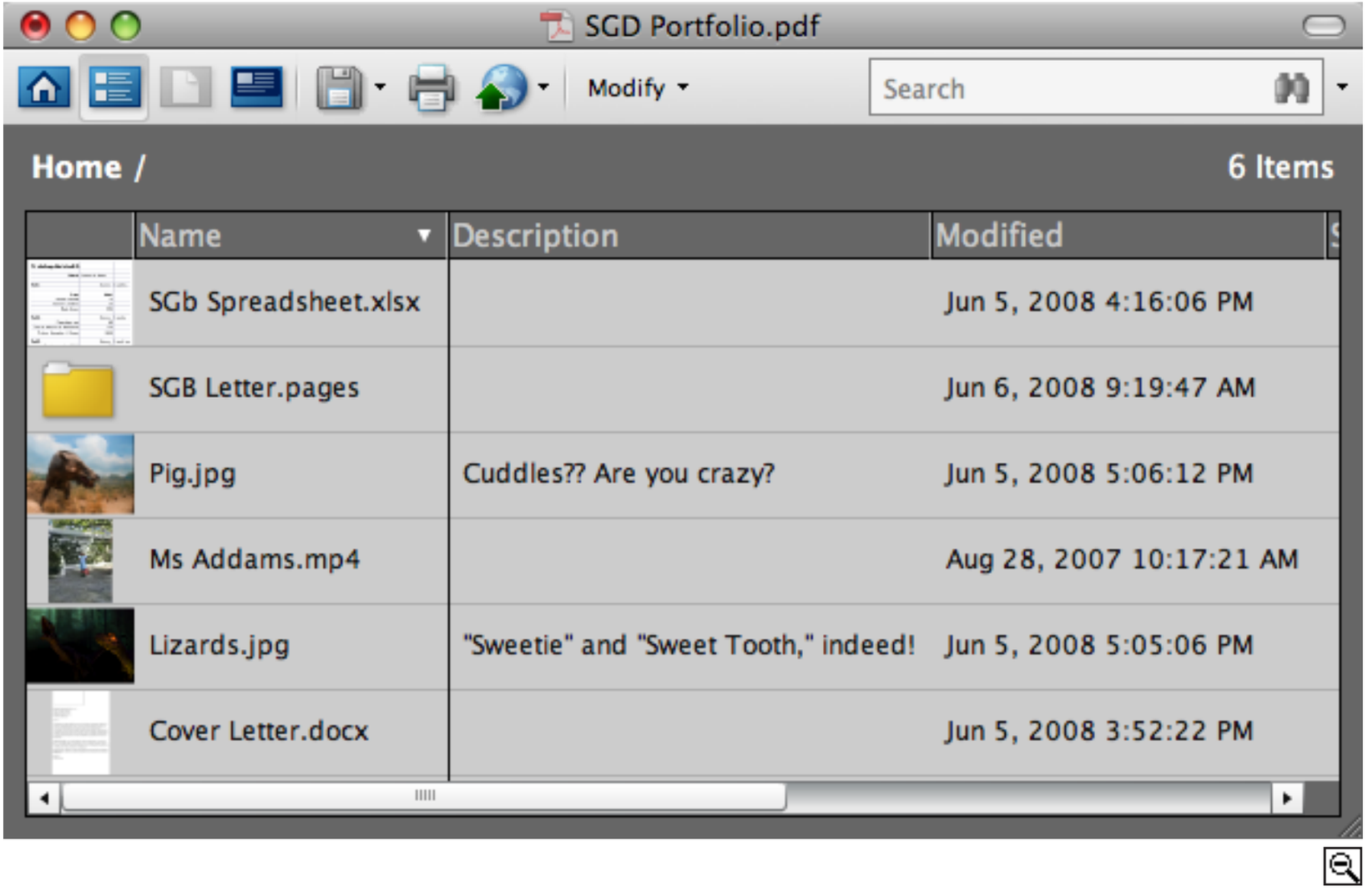
Edit Portfolio Window



Edit Portfolio Window



Portfolio List View



Portfolio in Acrobat 7