Rich media annotations and AcroFLeX

D. P. Story

Abstract

The *Adobe Supplement to the ISO 32000* document introduces a new annotation type, the rich media annotation. This paper describes rmannot, a new LaTeX package that implements this new annotation. Additionally, the acroflex package, a major application of rmannot, is also discussed with some technical details.

Work on these two packages followed the time the author spent working for Adobe Systems in 2008 as part of the Acrobat 9 development team.

1 Introduction

A new and exciting feature of Acrobat 9 and Adobe Reader 9 is that an Adobe Flash player is embedded in the application's executable code; consequently, version 9 offers native support for Flash video (FLV), Flash applications (SWF), and the digital audio encoding format MP3. This enables a reliable cross-platform playback experience for the user. The user accesses the Flash player, which plays the media, through the new *rich media annotation*.

The Acrobat 9 Pro user interface allows for the creation of a rich media annotation, the specification of the media to be either embedded or streamed from the Internet, the selection of a skin from a collection of predefined skins to control the media, and so on. The rmannot package implements the rich media annotation, as specified in Chapter 9.6 in the extensions document [1], and all the user interface features of the rich media annotation.

One of the motivations for writing the rmannot package was the desire to write an interactive graphing system for PDF. The acroflex package creates the AcroFLeX Graphing System, and represents a major application of the rmannot package. To this end, the AcroFLeX graphing widget, an SWF file, was created using Adobe FLEX 3. The AcroFLeX graphing widget is embedded in the document using the rmannot package, and it, as its name implies, performs and displays the graphing.

Another important part of the Acrobat/Flash connection is to establish a communication link between the two. The scripting language of Acrobat is JavaScript, while the Flash player uses ActionScript. These two scripting languages can communicate using the *scripting bridge*, created for Acrobat for this purpose. Details of the scripting bridge, and how it is used, are presented in section 3 on the acroflex package.

2 The rmannot package

The *PDF Specification, Sixth Edition, version 1.7*, see [3], has been made into an international standard, called ISO 32000 (ISO 32000-1), see [4]. The PDF specification, as documented in [4], is no longer under control of Adobe, consequently, Adobe now publishes an extensions document [1] that specifies new PDF language features that are not in ISO 32000. The resource document for rmannot and acroflex is the *Adobe Supplement to the ISO 32000* [1].

2.1 Multimedia for Versions 6 and 9 compared

The rich media annotation, as specified in [1] turned out to be very straightforward to implement. The new annotation is much easier than the complex multimedia approach of version 6. The multimedia of version 6 uses what the *PDF Specification* [3] calls a *screen annotation*, and has been implemented for LaTeX in the movie15 package [5] by Alexander Grahn.

The multimedia approach of version 6, which supports a long list of media types, requires the underlying operating system to locate and launch an appropriate multimedia player residing on the user's system to play the media clip. In rich media annotation of version 9, only SWF, FLV, and MP3 files are supported (other media types can be converted to one of these using Acrobat Pro Extended, or a third party conversion utility), but the embedded Flash player is used to play the media, so no external multimedia player is used or needed.

2.2 Implementation notes for rmannot

As it is now written, rmannot requires the document author to use Acrobat 9 Pro to create a PostScript file, perhaps using dvips, and to distill it using Acrobat Distiller 9. Once the document is built into a PDF, it can be viewed with Adobe Reader 9 (or later). Currently, this package is not available to users of pdftex.

The rmannot package is part of the AeB Pro family of packages (see [7]), this family is a collection of packages that require the use of Acrobat Distiller to create PDF. The package, documentation, and demo files can be obtained from the home page of rmannot.[1]

2.2.1 Embedding media

To embed media (SWF, FLV, and MP3 files) in the document, rmannot uses another package in the AeB

[1] The rmannot home page is at `http://www.math.uakron.edu/~dpstory/rmannot.html`.

Pro family, called graphicxsp (see [7]). The rmannot package defines a command `\saveNamedPath` that is to be executed in the preamble of the document; `\saveNamedPath` uses graphicxsp commands. The syntax is as follows:

`\saveNamedPath{`⟨*name*⟩`}{`⟨*path*⟩`}`

The ⟨*path*⟩ is the absolute path to the media file (with extension `.swf`, `.flv`, or `.mp3` included). The absolute path is required because Distiller does not work with relative paths and does not have a notion of current directory. The ⟨*name*⟩ is a symbolic name that is used to reference this media throughout the document.

2.3 Creating a rich media annotation

Once the media file has been embedded, in the body of the document, rich media annotations can be created using the `\rmAnnot` command:

`\rmAnnot[`⟨*options*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*name*⟩`}`

The ⟨*options*⟩ are key-value pairs that allow the document author to specify any of the rich media options available through the Acrobat user interface. The ⟨*width*⟩ and ⟨*height*⟩ are the width and height, specified in any of the scales of measurement LaTeX supports. The annotation may be resized, using, for example, the `\resizebox` command of the graphicx package, to any size while maintaining the aspect ratio of the media clip. Finally, the ⟨*name*⟩ is the name given the media by a `\saveNamedPath` command. This example,

```
\rmAnnot[poster=myPoster,
    skin=skin3]{640bp}{480bp}{myFLV}
```

shows some of the many optional key-value pairs available; here we specify a poster for the media (the appearance of the media when the annotation is not activated), and the skin to use to control this Flash video.

The rmannot package embeds a media file only once, but that media can be displayed and played on multiple pages without significantly increasing file size. Acrobat, by contrast, through its user interface, will embed the same media multiple times, once for each rich media annotation that uses that media file. In this regard, the approach taken to embedding by rmannot is superior to that of Acrobat.

There are other economies that should be mentioned as well. The embedded Flash player handles SWF files natively, but requires an SWF application to play video and sound files. FLV and MP3 files are actually played by `VideoPlayer.swf` and `AudioPlayer.swf`, respectively. These two SWFs are shipped with Acrobat Pro (and Extended). In addition to these, there are seven skin files (also SWF files) that are shipped with Acrobat that provide control over Flash video. The rmannot package takes care to embed each of these, as needed, only once; while Acrobat embeds the players and skins multiple times. This saving of file size becomes very important with the acroflex package, where the document author may want many graphing screens throughout the document.

It should be noted that the SWF players and skins that come with Acrobat are not distributed with the rmannot package, which would violate the licensing with Adobe; rather, rmannot requires the document author to have Acrobat Pro, so these files are already on his system. The players and skins are, by the way, one of the problems with porting rmannot to pdftex. The SWFs cannot be redistributed, so a package developer must write his own players and skin SWFs, and provide them in the package distribution.

3 The acroflex package

The acroflex package is a major application of the rmannot package. The keys to this package are the AcroFLeX graphing widget, which was written using Adobe FLEX 3, and the scripting bridge that allows communication between Acrobat (or Adobe Reader) and the graphing widget.

FLEX 3 — available from Adobe without charge to educators and students — is an XML-like (MXML) markup language that is compiled into an SWF file.[2] FLEX has charting (graphing) capabilities that are exploited by the acroflex package.

The acroflex package is part of the AeB Pro family of packages (see [7]). The package, documentation and demo files can be obtained from the home page of acroflex.[3]

3.1 Features of AcroFLeX

The AcroFLeX graphing screen (the visual appearance of the AcroFLeX graphing widget) can be interactive or non-interactive. The document author can create as many graphing screens as needed; each screen can appear in the rich media annotation at a fixed position in the document, or in a floating window.

For an interactive graphing screen, the user can enter an expression (into an Acrobat form text field) representing a function of a single variable x, a polar function of t, or a set of parametric equations that are functions of t. Various controls are provided to

[2] Adobe FLEX 3 can be found at `http://www.adobe.com/`.

[3] The acroflex home page is at `http://www.math.uakron.edu/~dpstory/acroflex.html`.

change the viewing window, for shifting horizontally and vertically, and for zooming in or out.

The author can also pre-populate the fully interactive screen by creating one or more links using the \sgraphLink command. When activated by clicking, the link passes graphing data to the graphing screen to be viewed by the user. The user may then interact with the graph.

For a non-interactive graphing screen, no controls are provided to manipulate the graph; the user can only view the graph. The screen is populated when the user clicks a link created by the command \sgraphLink, as described above. Information passed by the executing JavaScript of the link to the graphing routines of AcroFLeX includes the functional expression (or list of points to plot), domain, and range.

In the current version of AcroFLeX , up to four functions can be graphed, four functions with the shaded regions between the graph and horizontal axis can be graphed, and four sets of plotted points can be displayed, all on one graphing screen.

3.2 Implementation notes

The two major challenges of this package were to write the graphing widget using FLEX 3, and to write document-level JavaScript to calculate plot data to be passed to the graphing widget. The plot data is passed using the scripting bridge.

Development of the AcroFLeX graphing widget evolved over time as my understanding of Adobe FLEX grew. The rmannot package is then used to embed the widget in the PDF document, and to display the widget as the graphing screen through a rich media annotation.

On the Acrobat side, the JavaScript functions Graph_xy and Graph_xyt were written to prepare plot data for functions of a single variable, and for polar functions and parametric equations, respectively. These functions get graphing data supplied by the user (in the case of an interactive graphing screen), or receive graphing data as part of their parameters (in the case of using \sgraphLink to pre-populate a graphing screen). The exerquiz package (part of AeB, see [6]) is used to parse the target function, and the graphing data then create the plot data.

The plot data created by the graphing functions is built as an XMList. For example, suppose the function is x^2, to be plotted over the interval $[-2, 2]$, with $n = 5$ data points, the plot data has the following form:

```
cPlotData=<points>
    <point><x>-2</x><y>4</y></point>
    <point><x>-1</x><y>1</y></point>
    <point><x>0</x><y>0</y></point>
    <point><x>1</x><y>1</y></point>
    <point><x>2</x><y>4</y></point>
</points>
```

Once this XMList has been constructed, it is converted to a string,

```
cPlotData=cPlotData.toXMLString();
```

We then get the rich media annotation object,

```
var annot = this.getAnnotRichMedia(pNum,
    "afRM"+baseName);
```

and send the data to the AcroFLeX graphing widget by way of the scripting bridge,

```
annot.callAS("getPlotData", graph_props, oDR,
    cPlotData);
```

The bridge from Acrobat-to-Flash is the JavaScript callAS method of the rich media annotation. (See [2] for information on the callAS method). The function getPlotData is an ActionScript function defined in the AcroFLeX graphing widget. We pass the function name, "getPlotData", as a string; the properties of the plot graph_props (a JavaScript object); the range and domain specification oDR (a JavaScript object); and finally, the plot data itself, cPlotData.

In order for getPlotData ActionScript function to be recognized by the widget, it must be exposed by the ExternalInterface.addCallback method of ActionScript. Within the source code (MXML) of the AcroFLeX graphing widget, we have

```
private function initApp():void {
      ExternalInterface.addCallback(
        "getPlotData",getPlotData);
};
```

The initApp function is executed when the graphing widget is activated.

The function getPlotData, on the AcroFLeX widget side, interprets the graph_props object to determine what type of plot is to be created (plot points or draw a curve, draw continuous curve or segmented curve, shade graph or not). AcroFLeX maintains an array of length 12 to manage all the curves and plotted points. The function passes the viewing domain and range, the oDR object, to the chart for labeling, and assigns a unique color for the curve or plot. Finally, getPlotData converts the parameter cPlotData to XML,

```
var xmlPlotData:XML = new XML(cPlotData);
```

and populates the chart with the plot data.

3.3 Compatibility with exerquiz

The exerquiz package plays an important role in the acroflex package by providing the parsing routines

for algebraic expressions; however, exerquiz was created to provide quizzing environments for educators. The AcroFLeX graphing screens can be integrated into exerquiz quizzes using support commands provided by the acroflex package. The demo file `afgraph.pdf` contains several examples.

The interested reader is encouraged to view the demo file `afgraph.pdf`, located at the home page of the `acroflex` package.[4]

4 The AcroTEX PDF blog

Following the development of rmannot and acroflex, the *AcroTEX PDF Blog*[5] was created for the PDF and LaTeX communities; the blog covers various topics in PDF, including extensive information on the rich media annotation (RMA) and on the scripting bridge. The reader interested in these topics may read PDF blogs #1–11; these blog articles were written using the rmannot package.

5 Concluding remarks on LaTeX

The rmannot and acroflex packages forge no new ground in LaTeX code; they build on the AeB and AeB Pro bundles. Developing a package has become easier through the years because there are so many basic packages that are available for developers: hyperref, xkeyval, and xcolor, to name a few. In any case, LaTeX is shown to be a fine authoring system for new applications of emerging technologies.

[4] `http://www.math.uakron.edu/~dpstory/acroflex.html`

[5] `http://www.math.uakron.edu/~dpstory/pdfblog.html`

References

[1] Adobe Systems, Inc. *Adobe Supplement to the ISO 32000, BaseLevel 1.7, ExtensionLevel 3*. `http://www.adobe.com/go/pdf_developer`, 2008.

[2] Adobe Systems, Inc. *JavaScript for Acrobat API Reference*. `http://livedocs.adobe.com/acrobat_sdk/9/Acrobat9_HTMLHelp/index.html`, 2008.

[3] Adobe Systems, Inc. *PDF Reference, Sixth Edition, version 1.7, Adobe Portable Document Format*. `http://www.adobe.com/go/pdf_developer`, November 2006.

[4] International Organization for Standardization. *ISO 32000-1:2008, Document management — Portable document format — Part 1: PDF 1.7*. `http://www.iso.com`, 2008.

[5] Alexander Grahn. The movie15 package. Available from CTAN, `macros/latex/contrib/movie15`, 2008.

[6] D. P. Story. AcroTEX eDucation Bundle (AeB). Available from CTAN, `latex/contrib/acrotex`, 2008.

[7] D. P. Story. AeB Pro Family of Software. Available from CTAN, `latex/contrib/aeb_pro`, 2008.

◇ D. P. Story
Department of Mathematics
Northwest Florida State College
Niceville, FL 32578
U.S.A.
`dpstory (at) acrotex dot net`
`http://www.math.uakron.edu/~dpstory/`