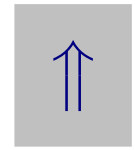
THE UNIVERSITY OF AKRON
Mathematics and Computer Science



mainmenu

Article: Links and Forms using Pdfmarks

Directory

Last Revision Date: 4/10/2001

Typeset by $\mathcal{AMS}$-TEX
& The Y&Y TEX System

Links and Forms using Pdfmarks

links

Table of Contents

⇑
toc

1. Acrobat 5.0: What's New

Acrobat 5.0 is a very *major* upgrade over version 4.0. The theme of this particular upgrade is, as I see it, "greater functionality through JavaScript". The number of JavaScript objects, properties and methods has increased tremendously. The functionality of most of the Acrobat plugins has been exposed to JavaScript.

The batch feature of Acrobat has been *significantly* expanded. There are "hard-wired" procedures that can be invoked for each of the selected files. These procedures can set security, crop and rotate pages; set document info; make format conversions (`tiff` to PDF, `html` to PDF, etc.); and print, to list just a few. More importantly, there is a general JavaScript procedure, with it, you can write your own procedures to perform whatever tasks you wish—within the capabilities of JavaScript, of course.

The documentation has grown, as well. The **Acrobat Forms JavaScript Specification, version 4.0** was a little over 60 pages, now, in Acrobat 5.0, the manual is more than 300 pages and has been renamed to the **Acrobat JavaScript Specification, version 5.0**—forms being only one part of the specification. In addition a complete

specification of Acrobat JavaScript methods, there are a large number of new and useful examples.

There are many other exciting features, including direct access to a database right from your desktop computer; and the ability to mark up a document on-line, within the browser window. The latter feature will be very useful for teams of researchers who collaborate on a grant proposal or a journal article. The new commenting scheme allows you to comment on a single centrally stored document, the comments are saved to a database and retrieved whenever the document is loaded into the browser.

However, all these new and exciting features are not the subject of this article. This short article will concentrate on those features which impact people who have a heavy reliance on the `pdfmark`. This category of people include users of TEX.

1.1. Bookmarks

In Acrobat 5.0, bookmarks now have both a *color* and a *style* attribute.

Color: The value of the color key, `/C`, determines the color of the bookmark. The value of `/C` is a three component array from **RGB**

color space; for example, `/C [ 1 0 0 ]` will give a red bookmark, in Acrobat 5.0 or later.

Style: The style of the bookmark is controlled by the flag key /F, which takes as its value an integer between 0 and 3, inclusive. There are four styles: Plain (`/F 0`), the default; *Italic* (`/F 1`); **Bold** (`/F 2`); and ***Bold & Italic*** (`/F 3`). If no **F** key is specified, the style defaults to Plain.

Example 1.1. Bookmarks with Color and Style. The following bookmarks were placed in this file. If you are using an Acrobat Viewer, 5.0 or later, you will be able to see the color and style of these bookmark entries. For Viewer's prior to 5.0, the bookmarks will appear as usual, black in color and plain in style.

```
[ /Title (Plain: /F 0)
    /C [ 0 .6 0 ]   % "webgreen"
    /F 0            % plain
/OUT pdfmark
[ /Title (Italic: /F 1)
  /C [ 1 0 0 ]      % red
  /F 1              % italic
/OUT pdfmark
[ /Title (Bold: /F 2)
  /C [ 0 1 0 ]      % blue
```

```
  /F 2                % bold
/OUT pdfmark
[ /Title (Bold & Italic: /F 3)
  /C [ 0 0 1 ]        % green
  /F 3                % bold and Italic
/OUT pdfmark
```

Example 1.1. ■

1.2. Viewer Preferences

In the Viewer Preferences of the `Catalog` dictionary, there is a new boolean key, `DisplayDocTitle`.

Example 1.2. DisplayDocTitle. The following example will cause the Acrobat Viewer, 5.0 or later, to display the document title, instead of the file name.

```
[ {Catalog}
<<
    /ViewerPreferences << /DisplayDocTitle true >>
>>
/PUT pdfmark
```

Example 1.2. ■

1.3. JavaScript and FDF

JavaScript code can now be embedded in an FDF file. These scripts can be Document-Level JavaScripts, or scripts that are executed when the FDF is imported.

To support this new functionality, there is a new `JavaScript` key that goes in the root level of an FDF. The value of the `JavaScript` key is a dictionary. This dictionary may contain one or more of the keys `Before`, `After` and `Doc`. These keys are illustrated in the examples below.

Example 1.3. The /After and /Before Keys. The value of these keys can be either a string or a stream that contain JavaScripts. The JavaScripts are executed before (the `Before` key), or after, (the `After` key), the FDF gets imported.

▶ The `After` key illustrated with a string value.

```
%FDF-1.2
1 0 obj
<<
  /FDF
  <<
    /JavaScript << /After (app.alert("Importing FDF Data now!");) >>
  >>
```

```
>>
endobj
trailer
<<
  /Root 1 0 R
>>
%%EOF
```

Example Notes: This script will be executed **after** the FDF file has been imported into the PDF file. ■

▶ The **Before** key illustrated with a stream value.

```
%FDF-1.2
1 0 obj
<<
  /FDF << /JavaScript << /Before 2 0 R >> >>
>>
endobj
2 0 obj
<<
>>
stream
  app.alert("Welcome to Acrobat 5.0!");
endstream
endobj
trailer
```

```
<<
  /Root 1 0 R
>>
%%EOF
```

Example 1.3. ■

Example 1.4. The /Doc Key. The `Doc` key takes an array (or an indirect reference to an array). There must be an even number of entries in the array. The even number entries are script names, the odd numbered ones are strings or streams containing JavaScripts.

▶ This example illustrates both methods of referencing JavaScript: String and Stream. When the FDF is imported, the document-level JavaScript are embedded, then the `After` key calls one of the new document-level JavaScripts.

```
%FDF-1.2
1 0 obj
<<
  /FDF
  <<
    /JavaScript
    <<
      /Doc 2 0 R
      /After (myHello();)
    >>
  >>
```

```
>>
endobj
2 0 obj
[ (myHello) (function myHello() {app.alert("Hello World")})
  (myHi)  3 0 R
]
endobj
3 0 obj
<<
>>
stream
function myHi()
{
  app.alert("Hi");
}
endstream
endobj
trailer
<<
  /Root 1 0 R
>>
%%EOF
```

Example 1.4. ■

The /Doc key indicates Document-level JavaScript. The above example illustrates two methods of creating document level script. The first

is the name of the function, the second is either a string or an indirect reference.

To automatically introduce the script into the document, place the following `pdfmark`:

```
[ {ThisPage} << /AA << /O << /S /JavaScript
       /JS  (%
            if (typeof myHello == "undefined")\r\t
            this.importAnFDF("dljs.fdf");
            )
    >> >> >>
/PUT pdfmark
```

If the function `myHello()` is undefined, the Doc level JavaScript `importAnFDF` imports the file `"dljs.fdf"` into the document (from the current directory).

1.4. Document-Level Actions

There are six new ways JavaScript code can be executed when certain events occur. These additional actions are inserted as values of the `/AA` key within the `Catalog` dictionary. The six keys and their descriptions are

- `/WC` "Document Will Close": JavaScript that will execute when the document is about to close.
- `/WS` "Document Will Save": JavaScript that will execute just before the document is saved.
- `/DS` "Document Did Save": JavaScript that will execute after the document is saved.
- `/WP` "Document Will Print": JavaScript that will execute just before the document is printed.
- `/DP` "Document Did Print": JavaScript that will execute just after the document is printed.

There is a UI within Acrobat to these actions; an illustration of how to define these actions with `pdfmark` follows:

```
\special{ps: [ {Catalog} <<
    /AA << /WC << /S /JavaScript /JS(console.println("Will Close")) >>
           /WS << /S /JavaScript /JS(console.println("Will Save")) >>
           /DS << /S /JavaScript /JS(console.println("Did Save")) >>
           /WP << /S /JavaScript /JS(console.println("Will Print")) >>
           /DP << /S /JavaScript /JS(console.println("Did Print")) >>
    >>
>> /PUT pdfmark}
```

1.5. Forms

Example 1.5. Arbitrary Font Definitions. Beginning with Acrobat 5.0, an arbitrary font can be used. Simply specify the PostScript name of the font in the /DA key, see the example below.

▶ Enter your name

```
\rtr\htxt{\lower4pt\Bbox[180pt,16pt]}\special{ps: %
[ /Rect \Rect
  /T (YourName)
  /FT /Tx
  /F 4 /BS << /S /I >>
  /MK << /BC [ 0 0 0 ]  >>
  /DA (/Viva-Regular 10 Tf 0 0 1 rg) % <-- specify Viva-Regular
  /DV (Enter your name)
  /V (Enter your name)
/Subtype /Widget /ANN pdfmark}\unhbox\bbox
```

Example 1.5. ■