# "Auto-entry" and "Auto-tracking" Acrobat PDF web Links

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

It is a fairly simple matter to add a web url link to an Acrobat PDF document. Simply get into full Acrobat, select your link area, and enter your url. Then resave.

The problems come about when you have hundreds or thousands of links in your document, when you want to link from a master data base list, when you want the editable links to look and behave like HTML, or when you want the links to automatically "follow" or track any edit repositioning or font size changes.

I've long had a fast, simple, and convenient PostScript fully automatic smart-font entry and autotracking PDF linking code as part of my Gonzo Utilities. What I thought I'd do here is show you exactly what these routines do and how they work. Chances are the same or similar concepts can be applied to your own raw PostScript formatting, or to any content editor or layout program in which you have at least some control over plug-ins, macros, or sourcecode.

Let's start with an example of how your text gets marked to include a url...

```
...part of my |/surl |3Gonzo Utilities|1|/gonzo01 . What...
```

In this case, |/surl or "start url" and its trailing space tells Gonzo that "a url link is about to begin. The |3 calls for a font change to the bold font needed for the url. The |1 returns us to normal text. And the |/gonzo01 and its trailing space calls for a specific url linking.

Your url linkings can come from a data base, from a list, or be generated on the fly. Each url linking proc must have a unique name. The simplest and most obvious way is to just do it...

```
/gonzo1 {(http://www.tinaja.com/post01.asp#gonzo) eurl } def
```

Which passes the url name string to a linking proc called eurl or "end url" that we will look at shortly. Known or common url's can be precombined into an external or internal file. Should you have lots of urls, using a "dictionary" or some "batch process" method can be cleaner and uses fewer characters...

```
<<
   /acrob01 (http://www.tinaja.com/acrob01.asp)
   /barg01 (http://www.tinaja.com/barg01.asp)
   /gonzo01 (http://www.tinaja.com/post01.asp#gonzo)
    . . . . . .
   /weblib01 (http://www.tinaja.com/weblib01.asp)

>>    {mark exch /eurl cvx ] cvx def} forall
```

OK. We now have a way to mark text where we want a link to appear. And have a list of places for each individual url to go. We need three routines to complete the process, surl to start the entry and eurl to complete the url generation. Followed by a makeurl that does the actual Acrobat stuff. Here's surl...

```
/surl {mark
  /blue cvx 0.33 /setgray cvx        % change text to blue
  /currentpoint cvx                   % remember box start
  /urly /exch cvx /store cvx
  /urlx /exch cvx /store cvx
      ] cvx                          % complete deferred command

  printlist exch 3 index exch put    % stuff into gonzo printlist
  exch 1 add exch                    % increment gonzo list count
     } def
```

All surl does is change the link color and font. It then remembers the currentpoint at which the linked text message starts. But this gets subtle and tricky in a big hurry. Why? Because you need to known the text start currentpoint at imaging time. The reason is that Gonzo or most other typical layout programs will analyze its lines on the first pass and then only put them down after text justification and repositioning calculations are complete. Thus, the currentpoint calculation must be deferred till actual imaging takes place.

In the case of Gonzo, we create an executable color change and "find the currentpoint" proc and stuff it into the printlist where it will execute at the proper time. Exact details will vary with your choice of layout code.

We have thus saved the position of the start of the url linking message with surl. Specifically, we have two new start coordinates named urlx and urly that mark the exact url text start positions. On to eurl...

```
/eurl {mark                          % start deferred proc
    exch                             % position url string
    /aqua cvx /black cvx             % reset to main text color
    /makeurl cvx                     % defer call of url builder
       ] cvx                         % complete deferred proc
```

```
    printlist exch 3 index exch          % stuff into gonzo printlist
    put exch 1 add exch                  % increment gonzo list count
      } def
```

This is somewhat similar to our previous code. All it does is change the text color back and pass a url string on to makerurl. As before, execution get deferred until imaging time by joining the Gonzo printlist.

At this point, we need an ordinary PostScript proc named makerurl to do the actual url magic. Such a proc relies heavily on the PDFmark operator described in depth in ATN 5150 from Adobe. Any layout or editing program capable of sending PostScript code to Distiller should be able to use (or least adapt) what follows.

makerurl works with seven pieces of information: The urlx and urly position info for the start of the url caption; the currentpoint info that holds the position for the end of the url caption; the font height info extractable from currentfont; a urlover constant that somewhat magnifies the hot box; and the url name string found on the stack top. Here's some code…

```
/urlover 0.2 def                         % fraction of hot area over bounds

/makeurl {
  /cururlname exch store                 % save the url string
  mark                                   % start pdfmark

  currentfont /ScaleMatrix get 3 get
  /fsize exch store                      % guess font height

  /Rect [ urlx fsize urlover mul sub      % set box left x
        urly fsize urlover mul sub        % set box left y
        currentpoint
        exch fsize urlover mul add exch
        fsize urlover mul add
        ]
  /Border [ 0 0 0]                       % [0 0 0 ] = none; [0 0 2] = debug
  /Color [ .7 0 0 ]
  /Action <</Subtype /URI /URI cururlname>>
  /Subtype /Link
  /ANN                                   % annotation type
  pdfmark                                % call pdf operators
} def
```

The cururlname is first grabbed. A guess is made at the font height by using the fourth value in the font's ScaleMartix. This value is then expanded by urlover to make a hot box somewhat larger than the actual text itself, extending slightly on all four sides. An invisible hot box is chosen, giving the illusion of an HTML link.

There are one or two minor glitches: If a multi-word url extends over more than one line, you can either recode the second line or else activate each word as a separate url. With Gonzo, you also may have to add some padding end-of-line spaces to make sure /surl ends up on the right line.

Note that the hot boxes automatically move around on the page as text is lengthened or shortened, and that they automatically scale as font size increases or decreases. As well as automatically repositioning on fill- or other justification changes. No post-Distilling treatment is usually needed.

Sourcecode for this document is Found Here.

Additional consulting services available per www.tinaja.com/info01.asp.