

AcroTeX.Net


AcroTeX PDF Blog

A Simple Countdown Widget for Presentations

D. P. Story

Use this button to reset the timer. Shift-click to change the length of the presentation.

The source files for this AcroTeX PDF Blog are attached to this PDF. Click [blog10](#) and save this file as `blog10.zip`.

You have five minutes to read this document, the countdown begins when you move to the next page. *A special surprise* awaits you at the end of the five! 

1. Introduction

At the time Acrobat 7 Pro rolled out, I was inspired to write my own presentation authoring system, for conferences, seminars, classrooms, or for the Web. (See my [AcroTeX.Net](#) web site for demo files of the [AcroTeX Presentation Bundle \(APB\)](#).) Coming from an academic background (in mathematics) as I do, there is a need for such a system that can author technical presentations (read mathematical equations).

Such authoring systems already exist in the world of \LaTeX , the typesetting system many mathematicians and scientists use. These systems were based on the approach PowerPoint uses: a slide may consist of a large number of pages with incremental content, as you page through the document the “bullet points” appear sequentially. The system works well in \LaTeX , but generates a large number of pages, which when converted to PDF translates to a large file size.

[APB](#) uses a system different from pages with incremental content, it uses layers to produce the incremental content; consequently, there is a one-to-one correspondence between slides and pages.¹ Very swave.

This is all leading up to the current topic, that of creating a countdown widget. It was my thought to incorporate a Flash countdown widget into my [APB](#). The author sets the time length of his/her talk. The counter is started when the talk begins, putting pressure is on the author to finish his talk on time! Can he do it?

2. A Description of the Countdown System

I searched the Internet for some MXML source code for a countdown widget, and settled on one written by Prayank Swaroop of Adobe. The code was modified extensively for inclusion in PDF, and to change the output of the widget. I had two sources of concern:

1. Modifying the code so that the countdown continues from page to page.
2. Creating an end-of-countdown event.

My first approach to problem (1) worked for me; this was intuitive and fairly simple. The solution to (2) was much harder for me, I tried many different approaches, all but one ended in crashing of Acrobat when the end-of-countdown event occurred. Fortunately, the solution I settled on seems pretty stable, no crashing.

The countdown you see at the top of each page is a rich media annotation (RMA) created by my `rmannot` package. Each is set to activate when the page is visible, and deactivate when the page is invisible.

¹Why did I write [APB](#) at the time Acrobat 7 rolled out? What's when the layers (optional content groups) JavaScript API fully matured.

When you are on the title page, the count has not started yet. It is only when the presenter moves to the first “slide” that the count begins. The title page is normally visible while the presenter is being introduced, this is not part of his/her allotted time for the talk.

2.1. Document and Page Close JavaScript

Before getting into the MXML source code for `cdc1lock.swf`, the underlying Flash application used here, we discuss several scripts that reside in this document.

The title page has a Page Close event, the JavaScript of which is

```
if (!bMAX_DATE) {
    MAX_DATE = (new Date()).valueOf();
    bMAX_DATE=true;
}
```

When the title page is closed, and if the global variable `bMAX_DATE` is `false`, we populate the global variable `MAX_DATE` with the current date, we then set `bMAX_DATE` to `true`. The variable `bMAX_DATE` prevents the `MAX_DATE` reset each time the author, or you, my dear readers, return to the title page and travel through the document again. The count will not be reset by returning to the title page.

The document JavaScript for this document is

```
var aSecond=1000, aMinute=60*aSecond, aHour=60*aMinute;
var total_time=5*aMinute;
var MAX_DATE;
var bMAX_DATE = false;
var bFinalEvent = false;
function getTimeObject() {
    return { MAX_DATE: MAX_DATE, total_time:total_time };
}
function myTimeOutEventInPDF() {
    if (!bFinalEvent) {
        this.pageNum=this.numPages-1;
        toggleSetThisLayer("FinalEvent", true);
        bFinalEvent = true;
    }
}
```

Time is measured in milliseconds, so we begin defining some convenience variables, `aSecond`, `aMinute` and `aHour`. We set the variable `total_time` to the length of the presentation, in milliseconds. The function `getTimeObject()` is called from the Flash widget `cdc1lock.swf` each time a page becomes visible to get the time the presentation starts `MAX_DATE` and the length of the presentation `total_time`. The `getTimeObject()` function is key to continuing

the countdown from page to page. The other function `myTimeOutEventInPDF()` is the end-of-countdown event. You can see that the document is turned to the last page, and a hidden layer is made visible. The variable `bFinalEvent` is used to prevent the end-of-countdown event from executing more than once.

2.2. The Countdown Widget `cdclock.swf`

A verbatim listing of the MXML source of `cdclock.swf` appears below. Some of the lines have been removed, only the lines pertaining to the function of the countdown are listed and commented upon.

```
1 <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
2     applicationComplete="initApp();" xmlns:utils="utils.*" >
3 <mx:Script>
4 <![CDATA[
5     import flash.external.ExternalInterface;
6     import mx.events.FlexEvent;
7     public function initApp():void {
8         stage.scaleMode = StageScaleMode.EXACT_FIT;
9         timeObject= ExternalInterface.call("getTimeObject");
10        MAX_DATE = new Date(timeObject.MAX_DATE);
11        totalTime = int(timeObject.total_time);
12        var addHours:int = Math.floor( totalTime / aHour );
13        var pHours:Number= totalTime % aHour
14        var addMinutes:int = Math.floor( pHours / aMinute);
15        var pMinutes:Number = pHours % aMinute;
16        var addSeconds:int = Math.round ( pMinutes / aSecond );
17        MAX_DATE.setHours(MAX_DATE.getHours()+addHours);
18        MAX_DATE.setMinutes(MAX_DATE.getMinutes()+addMinutes);
19        MAX_DATE.setSeconds(MAX_DATE.getSeconds()+addSeconds);
20        countDownClock.countDownDate = MAX_DATE;
21    }
22    public function outOFTimeEvent(evt:FlexEvent):void {
23        if (evt.currentTarget.text == "00:00:00" ) try {
24            ExternalInterface.call('eval',
25                'app.setTimeout("myTimeOutEventInPDF()", 250)');
26        } catch(e:Error){};
27    }
28 ]]>
29 </mx:Script>
30 <utils:CountDownClock height="100%" width="100%"
31     valueCommit="outOFTimeEvent(event)"
32     id="countDownClock" fontWeight="bold" fontFamily="Arial Bold" />
33 </mx:Application>
```

When the application is activated, `initApp()` is executed. In line 9, we call the document-

level function `getTimeObject()`, as discussed earlier. We get the start time `MAX_DATE`, and parse the `totalTime`, breaking it down into hours, minutes and seconds (`addHours`, `addMinutes`, and `addSeconds`). We then add these to `MAX_DATE` (lines 17–19), and set `countDownClock.countDownData` to this `MAX_DATE`. This is the date the presentation ends.

The `utils:CountDownClock` tag is a custom component, whose code is contained in `CountDownClock.mxml` in the `utils` folder. This file was modified to obtain a time output format of the form `xx:xx:xx`, as you see from the countdown in the top center of the page. This code is not presented here.

One point concerning this tag, there is a `valueCommit` property (line 31); essentially, each time the value of this field is changed, the function `outOfTimeEvent` is executed. The function `outOfTimeEvent`, defined in lines 22–27, tests whether the value of the field is `00:00:00`, if yes, the function `myTimeOutEventInPDF()`, back in the PDF, is executed by using the `ExternalInterface.call` method. Notice that we delay the execution of `myTimeOutEventInPDF()` by 250 milliseconds, this seems to avoid crashing problems. :–{)

I should note that when you view this document with a two-column format, the countdown widget does not always display on both page. By paging back and forth, you can get them both to appear.

2.3. Resetting the Counter

The code for the Reset button that appears on the first page follows:

```

1  if (event.shift) {
2      var resp = app.response({cQuestion: "Enter length of talk in minutes",
3          cTitle:"AcroTeX PDF Blog \#\thisBlog"});
4      if ( resp != null ) {
5          resp = parseFloat(resp);
6          if (!isNaN(resp)) {
7              total_time=resp*aMinute;
8              bFinalEvent = false;
9              toggleSetThisLayer("FinalEvent", false);
10             bMAX_DATE = false;
11         }
12     }
13 } else {
14     bFinalEvent = false; toggleSetThisLayer("FinalEvent", false);
15     bMAX_DATE = false;
16 }
```

When the user, that's you, clicks the Reset button, lines 14–15 are executed. We return to a "start-up" state, `bFinalEvent=false` (so the end-of-countdown event is allowed to occur), we toggle the layer off using `toggleSetThisLayer()` (a JS function that comes with one of

my \LaTeX packages), and finally, put `bMAX_DATA=false`. With this variable set to `false`, when the title page is closed, a new `MAX_DATE` value is computed based on the time the page was closed. Any widgets firing up thereafter will read this new value.

3. And in Conclusion

The advantage of using a Flash widget, as opposed to using a form field, to display the remaining time, is that a widget can be designed in any number of clever and beautiful ways. I personally have no creativity to do such a thing, it can be done, but it is beyond my pay scale.

That's pretty much it for now.