

AcroTeX.Net

The cntdwn Package Handling Notification Events for the Short Countdown Timer

D. P. Story

The source file for this AcroTeX AeB Blog is attached to this PDF, and can be saved by clicking [this link](#). ~~DP~~

Table of Contents

1 Introduction	3
2 The Notification Events	3
3 Custom Event Handlers	4

1. Introduction

This article discusses the four notification events of the short countdown timer defined in the `cntdwn` package. The `cntdwn` package is available through CTAN in the `cntdwn` folder. The `cntdwn` package has its own [home page](#) on the `math.uakron.edu` server; there, you will find the (PDF versions of the) basic examples that are distributed with the `cntdwn` package as well as the documentation and the package distribution itself (`cntdwn.zip`).

2. The Notification Events

The properties of a short counter are set through the second parameter, taking key-value pairs, of the `\setShortCntDwn` command. There are several properties of interest to us in this article:

Time	Event	Default value of Event
<code>length</code>	<code>endEvent</code>	<code>sDefEndEvent(doc,cTimer)</code>
<code>notify1</code>	<code>event1</code>	<code>sDefNotify1(doc,cTimer)</code>
<code>notify2</code>	<code>event2</code>	<code>sDefNotify2(doc,cTimer)</code>
<code>notify3</code>	<code>event3</code>	<code>sDefNotify3(doc,cTimer)</code>

The sequencing of the events of a short countdown depends on whether we are counting up or down.

- **Countdown:** The count begins at the value of `length` and starts to decrease to time zero. When the count reaches time `notify1`, `event1`, a JavaScript function, executes. (The default function for `event1` is `sDefNotify1`.) The count continues down to the `notify2` time when `event2` is executed, then to `notify3` when `event3` is executed. Finally, at time zero, `endEvent` is executed. For a countdown, we have $length > notify1 > notify2 > notify3 > 0$.

Observe the behavior of the default countdown timer (set to 1 minute) by pressing the **Start** button, the second button from the left.

The changing of colors of the buttons and the blinking button and the final message are all the default actions of the countdown timer. These functions can be replaced by custom functions to create other effects, as will be explained in this article.

- **Count-up:** The count begins at time zero and starts to increase to the value of `length`. When the count reaches time `notify1`, `event1`, a JavaScript function, executes. (The default function for `event1` is `sDefNotify1`.) The count continues down to the `notify2` time when `event2` is executed, then to `notify3` when `event3` is executed. Finally, at time zero, `endEvent` is executed. For a count-down, we have $0 < notify1 < notify2 < notify3 < length$.

Observe the behavior of the default count-up timer (set to 1 minute) by pressing the **Start** button, the second button from the left.

The changing of colors of the buttons and the blinking button and the final message are all the default actions of the countdown timer. These functions can be replaced by custom functions to create other effects, as will be explained in this article.

In the next section we discuss how to change the values of `notify1`, `notify2`, `notify3`, and `endEvent`, and to write your own custom event handlers.

3. Custom Event Handlers

We demonstrate a timer that has the usual default effects (changing colors, blinking buttons), but we want to enhance these effects.

In the preamble you will see the following declaration:

```
\setShortCntDwn{sRhyme}
{%
  length=1*\minutes,
  notify1=45*\seconds,
  notify2=30*\seconds,
  notify3=15*\seconds,
  event1=myNotify1,
  event2=myNotify2,
  event3=myNotify3,
  endEvent=myEndOfTalk
}
```

This is a one minute countdown, with notifications at 45, 30, and 15 seconds. Note that `event1`, `event2`, `event3`, and `endEvent` have values different from their defaults. Their values are supposed to be JavaScript functions that do something when the various notification events occur.

Before looking at the definitions of these event handlers—and to take up the rest of the space on this page—let’s see the effects of these new functions.

Nifty! You can execute any function of your own creation to perform whatever actions you can imagine.

Continued on to the next page, please.

Also in the preamble are the definitions of these functions:

```

1  \begin{insDLJS}[myNotify1]{myevents}{CntDwn: Custom Countdown Events}
2  function myNotify1(doc,cTimer) {
3      sDefNotify1(doc,cTimer);
4      var f=doc.getField(cTimer+".cntdwn.End");
5      if (f!=null) f.value = "A custom countdown can I write,\r";
6  }
7  function myNotify2(doc,cTimer) {
8      sDefNotify2(doc,cTimer);
9      var f=doc.getField(cTimer+".cntdwn.End");
10     if (f!=null) f.value += "With rhyme that is both good and tight.\r";
11 }
12 function myNotify3(doc,cTimer) {
13     sDefNotify3(doc,cTimer);
14     var f=doc.getField(cTimer+".cntdwn.End");
15     if (f!=null) f.value += "Who knows what the next rhyme may be.\r";
16 }
17 function myEndOfTalk(doc,cTimer) {
18     var f=doc.getField(cTimer+".cntdwn.Notify3");
19     if (f!=null) f.fillColor = color.red;
20     app.beep();
21     var f=doc.getField(cTimer+".cntdwn.End");
22     if (f!=null) f.value += "Can you rhyme words as good as me?";
23 }
24 \end{insDLJS}

```

These functions take two arguments, `doc`, which is the document object, and `cTimer`, the name of the timer.¹

The definitions for `myNotify1`, `myNotify2`, and `myNotify3` are all similar, so we only discuss `myNotify1`. The first thing we do (line 3) is to call the default function `sDefNotify1` to get the usual effects, then in line 4 we get the field object of the text field that will hold the text. This text field is created by the `\cntdwnEndTarget` command. The field name created by this command is `cTimer+".cntdwn.End"` (which becomes `"_osRhyme.cntdwn.End"` for this example). After acquiring the field object, we write some text to that field.

In the other two functions, when we write to the field, we used the `+=` assignment operator, which basically takes the value of the field and adds new text (see lines 15 and 22) to the end of the string.

The definition of `myEndOfTalk` does not call the default function first, rather gets the field whose name is `cTimer+".cntdwn.Notify3"`, which is the **Stop** button, and changes its button appearance to red (line 18-19); it then beeps the computer (line 20), and writes the last line of text to the field created by `\cntdwnEndTarget`, lines 21 and 22.

That's all there is to writing custom event handlers. Now, I must get back to my retirement.

¹The `cntdwn` package internally renames the timer name. If the timer name is `sRhyme`, as this one is, the value passed through the `cTimer` parameter is `_osRhyme`.