

**THE UNIVERSITY OF AKRON**  
**Theoretical and Applied Mathematics**

**The insDLJS Package: Demo the  
execJS and defineJS Environments**

**D. P. Story**

## 1. Introduction

This document demos two environments, `defineJS` and `execJS`, defined in the `insDLJS` Package, a stand alone package distributed with the `AcroTeX eDucation Bundle`. The example presented here is an `animation` (of sorts) created completely from a `LATEX` source file that utilizes these two environments.

## 2. The `execJS` Environment

The `execJS` environment can be used to create “executable and discardable” JavaScript. The applications of this environment are very exciting (to me), however, the *full Acrobat product is required* to make it work; more exactly, Acrobat 5.0 or greater is required. It should be emphasized, however, that the document author can create the PDF document using the Acrobat distiller, `pdftex` or `dvipdfm`, the Acrobat Viewer—not the Acrobat Reader—is needed to import and execute the JavaScript.

The `execJS` environment is used to write JavaScript. When the document is `LATEX`ed, the script is written verbatim to an FDF (Forms

Data Format) file. The environment also adds an open action, so that when the newly created PDF document is opened for the first time in Acrobat, the FDF file is imported and executed. After the JavaScript has executed, the next thing to do is to save the document. The FDF that is imported is not saved with the document, and will not be imported again into the document, thereafter. The document is then ready for distribution.

**Important:** This environment, and the technique on which it is based, described next, is used for “post-creation” document addendum; that is, after the document is created, the JavaScript that appears within the `execJS` environment, is executed. This script can perform a variety of tasks such as importing sound or PDF icons.

## 2.1. Security Restricted Methods

If the JavaScript you want to execute has no security restrictions, then no special preparations are needed to use this environment. Jump to the next section.

Many of the more interesting JavaScript methods have security

restrictions, they can only be executed during a menu event, a console event, or a batch event. Using the method described below, you can executed some or all of your JavaScript defined in the `execJS` environment through a menu. To do this, you need a custom menu.

Copy the following lines and paste them into a text file. Save the file with a `.js` extension, and place it in the `JavaScripts` folder of the Acrobat installation directory tree.

```
_MenuProc = function() {;}  
app.addMenuItem( { cName: "MenuProc", cUser: "Menu Procedure",  
    cParent: "Tools", cExec: "_MenuProc()", nPos: 0 } );
```

This code will create a custom menu item under the `Tools` menu. It is through this menu that we shall execute our JavaScript.

**Note:** For security reasons, rename the variable `_MenuProc` and the menu name `"MenuProc"` to something “secret”. Arbitrary JavaScript can be executed through this menu, so no one should be able to use this technique on your computer, but you!

## 2.2. The Environment

See `insDLJS.dtx` for more details of the `execJS` environment. An example of usage follows:

```
\begin{execJS}{execjs}
function importMyIcons ()                                (1)
{
  for ( var i=0; i < \numPics; i++)
    this.importIcon("rotate"+i,"animation.pdf",i);
}
this.addIcon("nullIcon", this.createIcon("", 0, 0));      (2)
_MenuProc = importMyIcons;                               (3)
app.execMenuItem("MenuProc");                            (4)
_MenuProc = function() {;}                               (5)
\end{execJS}
```

This is taken from the preamble of this document, and is used in the [animation](#) example that follows. In line (1), we define a JavaScript function `importMyIcons`; the body of which imports and names a series of PDF images from the `animation.pdf` document. (The document `animation.pdf` was produced using `PSTricks`.) We also create a null icon, (2), using the undocumented method, `this.addIcon`. This

icon is used to clear the button face of the `animation`.

The Acrobat JavaScript method `importIcon` has security restrictions, it can only be executed through a menu, the console, or in a batch sequence.

The trick to this technique is to then assign, in line (3), the variable `_MenuProc`, defined in the `.js` file described above, a value of `importMyIcons`, then execute the function `importMyIcons` through the menu item with the `app.execMenuItem("MenuProc")`. We finish off by resetting the value of `_MenuProc` to its default in line (5).

### 3. The `defineJS` Environment

The `defineJS` environment can be used to write JavaScript for buttons and other field fields.

Use `defineJS` to define a text macro whose expansion is the lines of code laid out within the environment. The `defineJS` environment takes one required argument and one optional one. The required parameter is the name of the text macro to be defined. The JavaScript code lines entered inside the `defineJS` environment are read verbatim

and are added to the definition of a macro. The optional parameter allows you to modify the verbatim read.

From the preamble, we have

```
\begin{defineJS}[\makecomment\%\makeesc\@]{\animateAction}
aMyIcons = new Array();
for ( var i=0; i < @numPics; i++)
    aMyIcons[i] = this.getIcon( "rotate" + i);
var count = 0;
function ShowIt()
{
    f.delay = true;
    f.buttonSetIcon( aMyIcons[run.count], 0 );
    run.count++;
    run.count @%= @numPics;
    f.delay=false;
}
var f = this.getField("myAnimation");
var run = app.setInterval("ShowIt()",150);
run.count = 0;
var timeout = app.setTimeout( "app.clearInterval(run);", 3*36*150);
\end{defineJS}
```

This environment defines a command `\animateAction`, whose expan-

sion is the JavaScript code you see above. The optional argument was used to change catcodes of `%` and `@`, to comment and escape, respectively. The command `\animateAction` to control the animation, see the next section.

Now we define a new command

```
\newcommand{\animateCtrl}
{
  %
  /A << /S /JavaScript /JS(\animateAction) >>
}
```

an optional definition. Finally, we create the button with the following code:

```
\eqGenButton[\CA{Push}\BC{1 .973 .863}
  \rawPDF{\animateCtrl}]{aniCtrl}{36bp}{10bp}%
```

The creation of the `\animateCtrl` is just a convenience. We could have defined the button this way as well:

```
\eqGenButton
[
  \CA{Push}\BC{1 .973 .863}
  \rawPDF{/A << /S /JavaScript /JS(\animateAction) >>}
]{aniCtrl}{36bp}{10bp}%
```

## 4. An Animation

Below is a simple animation— or picture show—which uses the techniques described above. The animation is produced by defining a sequence of PDF images as the appearance of the button below. The images are shuffled in and out at regular intervals.