AcroTEX.Net

Attaching and Linking PDFs

D. P. Story

Jump to an attachment: target.pdf

Jump to another attachment: target1.pdf

View LaTeX source file

Save unicodes.sty

Last Revision Date: September 19, 2006

Version 1.0

Table of Contents

1. Introducion

The purpose of this file is to layout some techniques using the AcroTEX System Tools for

- attaching a PDF to a parent file
- creating links between parent and child file (the attached file) and between children.

The resources for the development of these techniques are

- *PDF Reference*, Version 1.6
- *Acrobat JavaScript Scripting Reference*, Version 7.0
- *The Unicode Standard*, Version 4.0, Addison-Wesley, Reading, MA, 2003.
  On the web, see http://www.unicode.org.
- The AcroTEX System Tools, available for free download at www.acrotex.net. This is a LaTeX-based system.

It should be said up front that these techniques require Acrobat 7.0 Professional (or later), and, since you have Acrobat Pro, my friend, you might as well use the Acrobat distiller!

2. Attaching a File

We use a standard technique from the eforms package, part of AeB, to embed an attachment to a parent file. In my @EASE exam assembly system (www.acrotex.net), I use this technique to attach the LaTeX source file to the PDF.

The key JavaScript method for attaching a file to a PDF is the `importDataObject` method of the Doc object. This method has security restrictions on it so that it cannot be

used directly by JavaScript embedded in a PDF. To call this method from embedded JS, a trusted function must be used, with the script residing in a JavaScript folder.

First, we insert the following script in the preamble of the parent document:

```
\begin{execJS}{execjs}
var myPath = /.*\//i.exec(this.path);
aebTrustedFunctions(this, aebImportDataObject,
    {cName: "My Cool Attachment",cDIPath: myPath +"/target.pdf"})
\end{execJS}
```

We use the `execJS` environment to write script that will execute only once when the document is opened for the first time after distilling. The script enclosed in the `execJS` environment calls the JS function, `aebTrustedFunctions`, defined as folder JavaScript, The Doc object is passed as a parameter, along with `aebImportDataObject` (also defined as folder JS), and an object of arguments.

The code for the two functions `aebTrustedFunctions` and `aebImportDataObject` follows. Copy this paste this code into a file, and save it with a `.js` extension to the user JavaScript folder of Acrobat. The user JavaScript folder can be found on your system by executing the code `app.getPath("user","javascript")` from the console window.

```
aebTrustedFunctions = app.trustedFunction( function ( doc, oFunction, oArgs )
{
    app.beginPriv();
        var retn = oFunction( oArgs, doc )
    app.endPriv();
    return retn;
});
```

```
aebImportDataObject = app.trustPropagatorFunction( function ( oArgs, doc )
{
    app.beginPriv();
        return retn = doc.importDataObject(oArgs);
    app.endPriv();
});
```

3. Linking to the attachment with GoToE

The parameter cName in the above execJS code is of particular importance. The value of cName is used in the names tree for embedded files. It is used to reference the attachment in the link code. After the file is imported, the value of cName is converted by Acrobat to Unicode.

In section 8.5.3 of the *PDF Reference*, an example illustrates how to link from a parent file to a child:

```
1 0 obj         % Link to a child
    <<
        /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /T << /R /C /N (Embedded document) >>
    >>
endobj
```

The string value of the /N key must match the value of cName. As the value of cName is converted to Unicode, I have found, by experimentation, that the string value of /N must

also be written in Unicode. Consequently, it was necessary to write a small package, called the unicodes package, to convert a string to Unicode.

3.1. The unicodes Package

A brief description of this package is presented in this section

Define a helper macro to create the mapping from ASCII to Unicode hex encoding

```
\def\convertChriiUnicode#1#2{\expandafter\def\csname uni@#1\endcsname{#2}}
\convertChriiUnicode\space{0020}
\convertChriiUnicode{.}{002E}
\convertChriiUnicode{/}{002F}
\convertChriiUnicode{0}{0030}
...
\convertChriiUnicode{9}{0039}
\convertChriiUnicode{?}{003F}
\convertChriiUnicode{@}{0040}
\convertChriiUnicode{A}{0041}
...
\convertChriiUnicode{Z}{005A}
\convertChriiUnicode{a}{0061}
...
\convertChriiUnicode{z}{007A}
```

Create a helper macro to display the Unicode of the parameter #1:

```
\def\displayUnicode#1{\csname uni@#1\endcsname}
```

The Unicode hex bytes begin with FEFF followed by the encoding of the string, to quote from section 3.8.1 of the PDF Reference,

> For text strings encoded in Unicode, the first two bytes must be 254 followed by 255. These two bytes represent the Unicode byte order marker, U+FEFF, indicating that the string is encoded in the UTF-16BE (big-endian) encoding scheme specified in the Unicode standard.

The command `\stringiiUnicode` inserts `FEFF` then calls `\@stringiiUnicode` after expanding its argument. The argument of `\@stringiiUnicode` should be a text macro defined by the `\defineDescriptionStr`, defined below. `\@stringiiUnicode` reads one token at a time, and converts it to its Unicode encoding, until the next token is a '>'.

```
\def\stringiiUnicode{FEFF\expandafter\@stringiiUnicode}
\def\@stringiiUnicode#1{%
    \ifx#1>\expandafter>
    \else
        \displayUnicode{#1}\expandafter\@stringiiUnicode
    \fi
}
```

The command `\defineDescriptionStr` used to define a text macro that expands to the description of the attachment, the value of `cName` and the value of the `/N` key. This string is converted to Unicode using `\stringiiUnicode`. This macro reads its argument with LaTeX in `\obeyspaces` mode.

```
{\obeyspaces\gdef\aebarg#1{\gdef\thearg{#1}\@defineDescriptionStr}}
\def\defineDescriptionStr#1{\bgroup\def\ddargi{#1}\obeyspaces\aebarg}
\def\@defineDescriptionStr{%
    \global\expandafter\let\csname\ddargi\endcsname\thearg\egroup}
```

3.2. Setting up and creating the link

In the preamble, we declare the description of our attachment that is used to reference the attachment in the link code:

```
\defineDescriptionStr{myDescriptiveStr}{My Cool Attachment}
```

This creates command `\myDescriptiveStr` that is used to reference the attachment

As explained earlier, we include the `execJS` environment in the preamble:

```
\begin{execJS}{execjs}
console.println("Importing data object");
var myPath = /.*\//i.exec(this.path);
aebTrustedFunctions(this, aebImportDataObject,
    {cName: "\myDescriptiveStr",cDIPath: myPath +"/target.pdf"})
\end{execJS}
```

Note the use of the command `\myDescriptiveStr`, this text macro expands to "`My Cool Attachment`", as set by the `\defineDescriptionStr` example above.

To create a link to the attachment, I use the `\setLinkText` command from the eforms package:

```
Jump to an \setLinkText[\Border{0 0 0}
    \A{/D[0/Fit]/S/GoToE/T<</N<\stringiiUnicode\myDescriptiveStr>/R/C>>}]
    {\textcolor{webbrown}{attached file}} (first page, fit view)
```

Let's focus on the link action:

```
/D[0/Fit]/S/GoToE/T<</N<\stringiiUnicode\myDescriptiveStr>/R/C>>
```

We jump to the first page, with a "fit" view `/D[0/Fit]`, we jump to an attached document `/S/GoToE`, as indicated by the `GoToE` key. The target dictionary, `/T`, references the name

of the attachment with the /N key. The /R/C indicates the relation the target has to the parent, this code indicates the target is a child file.[1]

The value of the /N key is <\stringiiUnicode\myDescriptiveStr>.[2] The enclosing angle brackets (as opposed to parentheses) indicates the content is hex encoded. Between the angle brackets is \stringiiUnicode\myDescriptiveStr. The command \stringiiUnicode takes its argument (\myDescriptiveStr) and converts it to Unicode.

For example, if we define \myDescriptiveStr in this way,

```
\defineDescriptionStr{myDescriptiveStr}{My Cool Attachment}
```

then <stringiiUnicode\myDescriptiveStr> expands to

```
<FEFF004D007900200043006F006F006C00200041007400740061006300680061D0065006E0074>
```

Cool!

Important: The right angle bracket, '>', is used as a terminator for the recursive code in the definition of \stringiiUnicode. It is important to have no space between the end of the text macro \myDescriptiveStr and the '>'. Write <\stringiiUnicode\myDescriptiveStr> not <\stringiiUnicode\myDescriptiveStr >. Remember, spaces are active.

[1] See the *PDF Reference*, section 8.5.3, Tables 8.47 and 8.48, for a detailed description of the keys to the embedded go-to action (GoToE).

[2] The conversion to Unicode is not needed for the value of cName, as discussed at the beginning of section 3, when the files are attached using the importDataObject, Acrobat makes the conversion for us.

4. Let's Jump!

This file was created so that there are two attached files, we can jump to either one.[3] Once an attached file is open, we can jump back to the parent file, or jump to the other attached file. The file names of the two files are `target.pdf` and `target1.pdf`.

Jump to an attachment: target.pdf (first page, fit view)

Jump to another attachment: target1.pdf (jump to a destination)

The verbatim listing is found on the next page.

```
Jump to an attachment: \texttt{\setLinkText[\Border{0 0 0}
    \A{/D[0/Fit]/S/GoToE/T<</N<\stringiiUnicode\myDescriptiveStr>/R/C>>}]
    {\textcolor{webbrown}{target.pdf}}} (first page, fit view)

Jump to another attachment: \texttt{\setLinkText[\Border{0 0 0}
    \A{/D(Doc-Start)/S/GoToE/T<</N<\stringiiUnicode\myDescriptiveStri>/R/C>>}]
    {\textcolor{webbrown}{target1.pdf}}} (jump to a destination)
```

See the source files to `source.tex`, `target.tex` and `target1.tex` to see the code. See the `PDF Reference`, section 8.5.3, Table 8.48, for additional details on the keys to the target dictionary, `/T`

[3] That's not quite true, I've also included the source files to the PDF documents as well.