MicroPress

Visual TEX

PDF Reusable Objects

Abstract

This is the draft documentation for the VTEX Pdf form support. This is an ongoing development; this documentation is likely to be enhanced and revised. The features described in this document require VTEX 6.5 or later and Acrobat Reader 4 or later.

PDF terminology creates unfortunate confusion by using the word `form` to denote two very different objects:

- Reusable objects (also, `XObjects`)
- Fillable forms

This confusion originates from `forms` in the first sense appearing in the **Pdf** standard before **HTML** forms came into common use and Adobe had to provide a counterpart. To avoid the confusion in our documentation, we will avoid using the unqualified word "`form`" as much as possible, preferring "`reusable objects`" and "`fillable forms`".

Chapter 1

PDF Reusable Objects (Forms)

Pdf `reusable objects` allow one to group together a set of Pdf operation in one place and then refer to it from multiple places within the document. Use of such objects can substantially decrease the size of the **Pdf** file; in addition the reusable objects can be used in conjunction with `fillable forms`.

The following low-level operations are provided for working with `reusable objects`:

- the `\immediate\shipout{...}` extension produces `reusable objects`.
- the `\special{!reuse...}` backend special for referring to the `reusable objects`.
- the `\formcount` counter to keep track of the numbering of the `reusable objects`.
- the `\formwd`, `\formht`, and `\formdp` primitives to retrieve the dimensions of a stored `reusable object`.

The general technique for using the `reusable objects` is this:

- Form a `\box{..}` with some TEX material.
- Apply `\immediate\shipout` to it.
- Save the current value of the `\formcount` counter, say N.
- Use the `\special{!reuse N}` whenever you want to reinsert the material of this form.

- Adjust the positioning of the form accordingly to its dimensions.

For example,

```
\setbox0=\hbox{\fbox{Hello, World}}
\immediate\shipout\box0
\newcount\N \N=\formcount
Here we reuse:
  (((\special{!reuse \the\N}))).
```

Here we reuse: (((Hello, World)))

Notice that in most cases we should manually reserve the space on the page for a reused object. The following code will work better:

```
\setbox0=\hbox{\fbox{Hello, World}}
\newdimen\formwid \formwid=\wd0
\newdimen\formhgt \formhgt=\ht0
\immediate\shipout\box0
\newcount\N \N=\formcount
Here we reuse:
  (((\vbox to \formhgt{\hbox to \formwid{
    \special{!reuse \the\N}\hss}\vss}))).
```

Here we reuse: (((Hello, World))).

It is not necessary to record the dimensions of the form yourself; you can retrieve them with the `\formwd`, `\formht`, and `\formdp` primitives. For example,

```
\setbox0=\hbox{\fbox{Hello, World}}
\immediate\shipout\box0
\newcount\N \N=\formcount
Here we reuse:
  (((\vbox to \formht\N{\hbox to \formwd\N{
    \special{!reuse \the\N}\hss}\vss}))).
```

Here we reuse: (((Hello, World))).

The semantics of the `\formwd`, `\formht`, and `\formdp` primitives is similar to that of `\wd`, `\ht`, and `\dp`. However:

- You cannot set the dimensions; a command like `\formwd1=100pt` is always ignored.
- The return value in modes (**Dvi**, **Html**) which do not support `reusable objects` is always `0pt`.
- The return value for forms which have not been yet defined is always `0pt`.

None of these cases produces error messages.

Chapter 2

PDF fillable forms

Abstract

Acrobat forms first appeared in the Acrobat Reader 3.02 as a response to the popularity of **Html** forms on the Web. VTEX 6.5 fully supports the functionality of the **Pdf** forms.

VTEX implementation of the **Pdf** forms is provided by the `\special !field...`. Depending on its parameters (and there are *many*), this special can create buttons, check boxes, radio buttons, edit or scroll boxes.

2.1 General syntax

The `\special{!field..}` command should list a set of options defining the appearance and the behavior of the field. The options are given in the

`<key>=<value>`

format; use the comma to separate multiple options. The keys are very short to minimize the use of TEX string space; the `value`'s, depending on the `key` could be keywords, dimens, integers, references to reusable objects, or strings.

The usually required keys are

- `k=` to define the kind of a field (i.e. button, checkbox,...)
- `w=` to define the *internal* width of a field.
- `h=` to define the *internal* height of a field.
- `i=` to define the id of a field (string).

In addition to these fields, a meaningful field should contain one or more presentation appearance definitions (`p..=`) and one or more actions (`x..=`).

Due to the complexity of this `\special` we will use the *explain-by-example* approach in this chapter; a full list of all available keywords can be found below.

2.2 Simple button

Here is a rudimentary definition of a button:

```
(\special{!field k=b,w=2cm,h=1cm})
```

() This button is not particularly functional and not even very noticeable; you can, however, sense it by clicking on the screen to the right and below of the right parenthesis. Clicking would reveal the active rectangle (2cm×1cm) of the button.

There are actually four things that are wrong with the above button:

1. from the point of TEX formatting, this button takes no space.
2. this button does not have any appearance.
3. this button does not do anything.
4. this button has no `id` defined; in such cases, VTEX will supply a random `id`, but you cannot refer to this button from other fields.

We shall remedy this problems one by one. First, to ensure that TEX reserves the needed space for the button, we will form a box explicitly:

```
(\vbox to 1cm{\hbox to 2cm{%
  \special{!field k=b,w=2cm,h=1cm}\hss}\vss})
```

( )

Next, to make the button visible, we shall define a simple reusable object.

```
\newdimen\butwd\butwd=0.25in  %18 px
\newdimen\butht\butht=0.25in
\newcount\Z \Z=16
\newcount\SimpleButt
```

```
\setbox1=\hbox to \butwd{%
\special{pS:
         gsave
         2 -2 moveto
         2 setlinewidth
         2 -16 lineto
         16 -16 lineto
         16 -2 lineto
         closepath stroke
         grestore
         }\hss}
\immediate\shipout\vbox to \butht{\vss\box1}
\SimpleButt=\formcount
```

and reuse the appearance in the button:

```
(\vbox to 1cm{\hbox to 2cm{%
  \special{!field k=b,w=2cm,h=1cm,pnu=\the\SimpleButt}\hss}\vss})
```

()

Notice that because of the mismatch in the sizes of the reusable object and the field, the appearance got rescaled. Generally, it is best to keep them the same and avoid any rescaling:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
     pnu=\the\SimpleButt}\hss}\vss})
```

()

We can make things a little more interesting by defining an alternative shape to appear when the mouse is moved over the field.

```
\newcount\RedButt
\setbox1=\hbox to \butwd{%
\special{pS:
```

```
          gsave
          1 0 0 setrgbcolor
          2 -2 moveto
          2 setlinewidth
          2 -16 lineto
          16 -16 lineto
          16 -2 lineto
          closepath
          gsave
          stroke
          grestore
          1 0.5 0.5 setrgbcolor
          fill
          grestore
          }\hss}
\immediate\shipout\vbox to \butht{\vss\box1}
\RedButt=\formcount

         (\vbox to \the\butht{\hbox to \the\butwd{%
           \special{!field k=b,w=\the\butwd,h=\the\butht,
              pnu=\the\SimpleButt,pru=\the\RedButt}\hss}\vss})
```

(□)

The third possible button shape can be specified for a "depressed" button:

```
\newcount\GreenButt
\setbox1=\hbox to \butwd{%
\special{pS:
          gsave
          0 1 0 setrgbcolor
          4 -4 moveto
          4 setlinewidth
          4 -14 lineto
          14 -14 lineto
          14 -4 lineto
          closepath
```

```
          gsave
          stroke
          grestore
          0.5 1 0.5 setrgbcolor
          fill
          grestore
          }\hss}
\immediate\shipout\vbox to \butht{\vss\box1}
\GreenButt=\formcount

(\vbox to \the\butht{\hbox to \the\butwd{%
 \special{!field k=b,w=\the\butwd,h=\the\butht,
 pnu=\the\SimpleButt,pru=\the\RedButt,pdu=\the\GreenButt
   }\hss}\vss})
```

()

This shape will become visible when you click on the button.

Of course, all the buttons defined so far do exactly nothing.

2.3 Actions

To make things more interesting, we can attach actions to buttons. The following keys are defined:

- `xd=` the action to perform on field "down".
- `xu=` the action to perform on field "up".
- `xe=` the action to perform on field "enter".
- `xl=` the action to perform on field "leave".

Of the four possible actions, `xu` is the most useful.

While several possible formats are available to express the action, the most general one is JavaScript. Almost all possibly useful actions can be expressed in JavaScript.

For example, to go to a `named` destination `Label` within a document, you can use this JavaScript code:

```
this.gotoNamedDest("Label");
```

This document provides a named destination `GeneralSyntax` at the beginning of this chapter; it has been inserted with

```
\special{!aname GeneralSyntax}
```

We can now turn our button into a fully functional link:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
    pnu=\the\SimpleButt,pru=\the\RedButt,pdu=\the\GreenButt,
     xu=<JS "this.gotoNamedDest(""GeneralSyntax"");">}\hss}\vss})
```

(□)

The button below demonstrates the action on "enter":

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
    pnu=\the\SimpleButt,pru=\the\RedButt,pdu=\the\GreenButt,
     xe=<JS "this.gotoNamedDest(""GeneralSyntax"");">}\hss}\vss})
```

(□)

A couple of notes about the syntax:

- The action should be surrounded in the angular brackets and contain the type of the action (JS above) and a string that describes the actual action (in double quotes).
- If the string contains quoted substrings, write the double quote twice.

2.4 Global JavaScript

You can also include fragments of global JavaScript code into the document. These fragments contains global JavaScript variables and methods.

To include global fragments, use the

```
\special{!js name "..."}
```

special command. The name field is currently not used (but required); the string should contain the JavaScript code.

For additional details on JavaScript, refer to the NetScape and Adobe documentation.

2.5 Check Boxes

Check Boxes are fairly similar to buttons. To specify a checkbox, you should use `k=c` (`c` for "Check Box"). In most cases, you would need to define at least two appearances:

- `pnc` Normal Checked appearance.
- `pnu` Normal UnChecked appearance.

Here is another simple reusable object we will use to experiment with Check Boxes:

```
\newcount\CheckedButt
\setbox1=\hbox to \butwd{%
\special{pS:
         gsave
         2 -2 moveto
         2 setlinewidth
         2 -16 lineto
         16 -16 lineto
         16 -2 lineto
         closepath
         3 -3 moveto 15 -15 lineto
         3 -15 moveto 15 -3 lineto
         stroke
         grestore
         }\hss}
\immediate\shipout\vbox to \butht{\vss\box1}
\CheckedButt=\formcount

(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=c,w=\the\butwd,h=\the\butht,
    pnc=\the\CheckedButt,pnu=\the\SimpleButt}\hss}\vss})
```

(☐)

In total, Check Boxes allow six different appearance states:

- `pnc` Normal Checked appearance.

- `pnu` Normal UnChecked appearance.
- `prc` Resident (Mouse over) Checked appearance.
- `pru` Resident (Mouse over) UnChecked appearance.
- `pdc` Depressed (Mouse down) Checked appearance.
- `pdu` Depressed (Mouse down) UnChecked appearance.

With Check Boxes, two other parameters are important:

- `i=` the `id` field must be specified to be able to retrieve the checkbox setting by JavaScript or to submit it with the form. The `id` values should generally be unique.
- `v=` should be set to either `Off` or `On` to produce a Check Box which is unchecked or checked initially. If this key is not given, the default is "unchecked".

Just like Buttons, Check Boxes allow one to specify actions for the events of Entering, Leaving, or clicking Up and Down.

2.6 Radio Buttons

The radio buttons are specified with the kind (`k`) value of `r`.

Unlike other fields, Radio Buttons function in groups; an individual Radio Button is essentially the same as a Check Box. A group of Radio buttons shares the same id (`i`). The appearance flags (`p..`) for Radio Buttons are the same as for Check Boxes.

Here is a pair of connected Radio Buttons:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=r,w=\the\butwd,h=\the\butht,i="RB",v="1",
    pnc=\the\CheckedButt,pnu=\the\SimpleButt}\hss}\vss})
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=r,w=\the\butwd,h=\the\butht,i="RB",v="2",
    pnc=\the\CheckedButt,pnu=\the\SimpleButt}\hss}\vss})
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=r,w=\the\butwd,h=\the\butht,i="RB",v="3",
    pnc=\the\CheckedButt,pnu=\the\SimpleButt}\hss}\vss})
```

(☒) (☒) (☒)

Notice that the value field `v=` should be different for all radio buttons; the value `Off` should not be used.

Just like Buttons and Check Boxes, Radio Buttons allow one to specify actions for the events of Entering, Leaving, or clicking Up and Down.

2.7 Edit fields

Edit and Selection fields are less flexible than the already discussed fields in terms of the appearance. Rather than providing reusable objects, we specify the general appearance of these fields with these keys:

- `b=` Background Color
- `o=` Border Color
- `c=` Text Color
- `z=` Font Size (`pt`)

The first three parameters should be followed by a color specification in the form `rrggbb`.

Text fields are defined with the kind (`k`) setting of `t`. Here is a sample text field:

```
\special{!field k=t,w=3in,h=0.5in,i="tf1",v="yes",
  b=ff0000,o=00ff00,c=0000ff,z=33pt}
```

Usually, the height of the box should slightly exceed the size of the font. The default font size is 10pt; this would correspond to 12pt height:

```
\special{!field k=t,w=3in,h=12pt,i="tf2",v="Sample Text",
  b=ff0000,o=00ff00,c=0000ff}
```

The `v=` value defines the initial text to be displayed.

The `W` key (no value needed) defines a password field:

```
\special{!field k=t,w=3in,h=12pt,i="tf3",v="Password",
  b=ff0000,o=00ff00,c=0000ff,W}
```

The M key (no value needed) defines a multiline field:

```
\special{!field k=t,w=3in,h=12pt,i="tf4",
  v="This is a multiline field",
  b=ff0000,o=00ff00,c=0000ff,M}
```

The R key (no value needed) defines a read-only field:

```
\special{!field k=t,w=3in,h=12pt,i="tfa",v="Read-only",
  b=ff0000,o=00ff00,c=0000ff,R}
```

The q key defines the field alignment:

```
\special{!field k=t,w=3in,h=12pt,i="tf5",v="Left",
  b=ff0000,o=00ff00,c=0000ff,q=l}
\special{!field k=t,w=3in,h=12pt,i="tf6",v="Right",
  b=ff0000,o=00ff00,c=0000ff,q=r}
\special{!field k=t,w=3in,h=12pt,i="tf7",v="Center",
  b=ff0000,o=00ff00,c=0000ff,q=c}
```

2.8 Selection fields

Selections fields are similar to Text fields in many attributes. The new important attribute is `s=` which defines the actual selection list. The affiliated value is the list of entries to appear in the selection list and their values.

For example (!!! should look up actual codes!!!):

```
\special{!field
  k=l,w=3in,h=0.5in,i="sf1",v="20",
  b=ff00ff,o=ffff00,c=00ff00,q=c,
  s=<<"apples","1"><"oranges","2">
    <"pears","3"><"mushrooms","100">>}
```

The optional `O` key (no value) turns the list box into a pOpup.

```
\special{!field
  k=l,w=3in,h=0.5in,i="sf2",v="20",
  b=ff00ff,o=ffff00,c=00ff00,q=l,O,
  s=<<"apples","1"><"oranges","2">
    <"pears","3"><"mushrooms","100">>}
```

The optional `C` key (no value, requires `O`) turns the list box into a Combo box. A Combo box is a combination of a list and edit fields which allows both selection and editing.

```
\special{!field
  k=l,w=3in,h=0.5in,i="sf3",v="20",
  b=ff00ff,o=ffff00,c=00ff00,q=r,O,C,
  s=<<"apples","1"><"oranges","2">
    <"pears","3"><"mushrooms","100">>}
```

As the last example shows, aligning the entries to the right may not be the best idea.

2.9 Form Submission

All forms naturally fall into two categories:

- Forms that are processed locally (with JavaScript)
- Forms that are submitted to a remote server for processing.

[Naturally, many forms that are submitted are also pre-processed locally, to assure that all the needed fields are filled up.]

To submit a form, use an action with the `SF` action key. For example:

```
xu=<SF h,
 u="http://www.micropress-inc.com/pcgi-bin/test.cgi">
```

Notice here that

- The `http://` prefix is required
- The `h` key indicates usual HTML/CGI submission; without this key, the form is encoded as Adobe's FDF.

Here is full list of sub-keys that are recognized:

- `h` HTML submission
- `p` PDF (FDF) submission (default), opposite of `h`
- `e` submit empty fields
- `n` submit only non-empty fields (default), opposite of `e`
- `u=..` the URL
- `f=..` the list of fields to be submitted, in angular brackets, each name in double quotes. For example

```
f=<"lastName","firstName","middleInitial">
```

If this sub-key is not present, all fields are submitted.

Naturally, for a PDF form submission to work, you also need to provide a CGI script on the server end.

You can also submit a form from JavaScript, using the

```
this.submitForm(...);
```

method. `submitForm()` may have between one and five parameters, which are:

1. The URL (the only required parameter)
2. Boolean flag, `true` (default) means FDF, `false` means HTML form submission format.
3. The list of fields to include in submission (field IDs, `i=/v=` pairs are actually submitted). Could be a string with a single name, or an array of strings that represent field id's.
4. Boolean flag, if `true`, empty fields are submitted; if `false` (default), all fields are submitted.
5. Boolean flag, if `true`, use Get submission, rather than the default Post.

2.10 Form Reset

For resetting (clearing) a form, we supply the action key `RF`. The syntax is similar to that for the form submission (the `SF` key); but the only allowed sub-key is the field list (`f`).

In JavaScript,

```
this.resetForm();
```

can also be used; it has either no or one parameter (the field list).

2.11 Sound Actions

One other type of action supported by VTEX is *sound*. The syntax of a sound action is

```
<SN "filename">
```

or

```
<SN volume "filename">
```

where `filename` is a name of a `.wav` file that contains the sound. The volume field (which should be an integer in the range 0 through 1000) is processed correctly by VTEX but does not seem to be supported by the current versions of the Acrobat Reader. VTEX searches for the wave files using the `INCDIR` setting.

Here is an example of a simple button that supports sound:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
     pnu=\the\SimpleButt,xe=<SN "ding.wav">}\hss}\vss})
```

(□)

Try moving the mouse over the button.

2.12 Multiple Actions

You can hook more than one action to an event. Multiple actions definitions are combined with the + symbol. For example, the button below will sound a bell and then perform a hyperlink jump.

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
     pnu=\the\SimpleButt,xe=<SN "ding.wav">+
        <JS "gotoNamedDest(""GeneralSyntax"");" >
        }\hss}\vss})
```

()

Chapter 3

Dirty Tricks

3.1 Hidden Fields

One additional option allowed with all field is to hide them. The key is `H`; no value is allowed.

For example, this code will produce a hidden button:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,i="bb",
     pnu=\the\SimpleButt,H}\hss}\vss})
```

( )

Hidden fields can be un-hidden with JavaScript. Here is another button, this time visible:

(☐)

Just move the mouse over it and the first button will appear. Here is the definition of the second button that creates this effect:

```
(\vbox to \the\butht{\hbox to \the\butwd{%
  \special{!field k=b,w=\the\butwd,h=\the\butht,
     pnu=\the\SimpleButt,xe=
        <JS "var t=getField(""bb""); t.hidden=!t.hidden;">
     }\hss}\vss})
```

3.2 Multistate Buttons

Overlapping a set of fields with only one of them visible allows one to create multistate buttons. We shall prepare three new shapes:

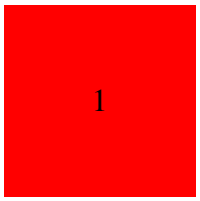
```
\setbox1=\hbox to 1in
{\special{pS: gsave 0 0 moveto 0 -72 lineto 72 -72
lineto 72 0 lineto closepath 1 0 0
setrgbcolor fill grestore }\hss1\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\newcount\one\one=\formcount
\setbox1=\hbox to 1in
{\special{pS: gsave 0 0 moveto 0 -72 lineto 72 -72
lineto 72 0 lineto closepath 0 1 0
setrgbcolor fill grestore }\hss2\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\newcount\two\two=\formcount
\setbox1=\hbox to 1in
{\special{pS: gsave 0 0 moveto 0 -72 lineto 72 -72
lineto 72 0 lineto closepath 0 0 1
setrgbcolor fill grestore }\hss3\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\newcount\thr\thr=\formcount
```

To save on the code, we will use this Global JavaScript segment:

```
{
\catcode10=\active
\special{!js junk "var z=1;
 var z1=getField(""b1"");
 var z2=getField(""b2"");
 var z3=getField(""b3"");
 function upd() { z++; if (z==4) z=1;
   if (z==1) { z1.hidden=false; z2.hidden=true;  z3.hidden=true; }
   if (z==2) { z1.hidden=true;  z2.hidden=false; z3.hidden=true; }
   if (z==3) { z1.hidden=true;  z2.hidden=true;  z3.hidden=false;}
   };"
}}
```

and produce the multistate button with

```
\vbox to 1in{\hbox to 1in{%
          \special{!field k=b,w=1in,h=1in,i="b1",
               pnu=\the\one,xu=<JS "upd(); ">}%
          \special{!field k=b,w=1in,h=1in,i="b2",
               pnu=\the\two,H,xu=<JS "upd(); ">}%
          \special{!field k=b,w=1in,h=1in,i="b3",
               pnu=\the\thr,H,xu=<JS "upd(); ">}%
          \hss}\vss}
```

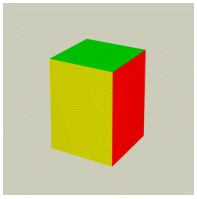
Here is the 3-state button we constructed: 

3.3 Image Buttons

You can create more attractive buttons by using images. We will redefine the shapes of the three states as

```
\setbox1=\hbox to 1in
{\hss\includegraphics[width=1in,height=1in]{cube.gif}\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\one=\formcount
\setbox1=\hbox to 1in
{\hss\includegraphics[width=1in,height=1in]{tetra.gif}\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\two=\formcount
\setbox1=\hbox to 1in
{\hss\includegraphics[width=1in,height=1in]{octa.gif}\hss}%
\immediate\shipout\vbox to 1in{\vss\box1\vss}
\thr=\formcount
```

to obtain

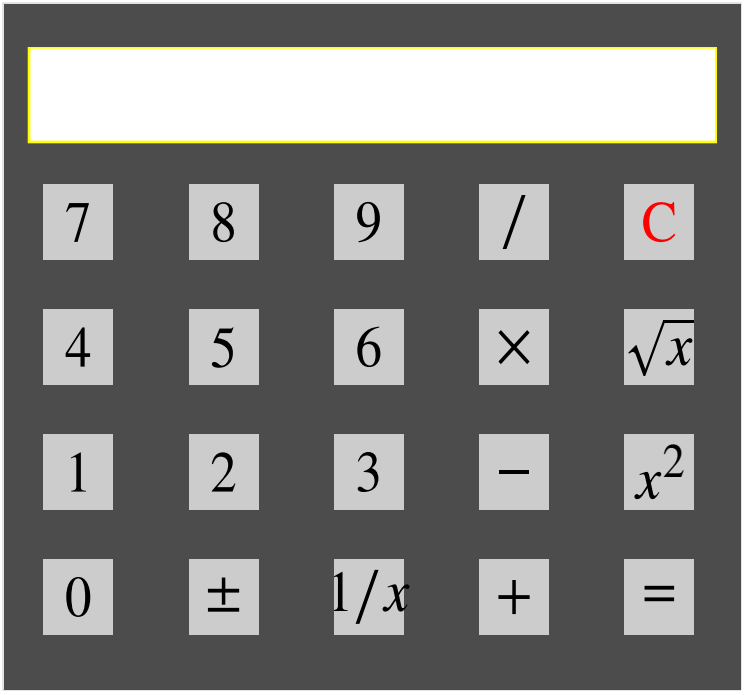


(the rest of the code is identical to that given in the previous section).

Chapter 4

The Calculator

This part of the documentation demonstrates a fairly sophisticated example of a JavaScript form: an embedded calculator.



Source:

```
\definecolor{gray}{gray}{0.8}
\definecolor{darkgray}{gray}{0.3}
{
\catcode10=\active
\special{!js calculator "var display=this.getField(""Display"");
var func;
var accum;
var entry;
var multiplier;
var divisor;
%
var bDirty = this.dirty;
all_cancel();
this.dirty = bDirty;
%
%// Function to initialise (all cancel)
function all_cancel()
{
  clear_entry();
  display.value = 0;
  accum = 0;
  func = """";
}
%
%// Function to clear current entry only
function clear_entry()
{
  entry = 0;
  multiplier = 1;
  divisor = 1;
}
%
%// Function to return number raised to the power of exponent
function pow(number,exponent)
{
```

```
  if (exponent > 1) {
    return number * pow(number, exponent - 1);
  } else {
    return 1;
  }
}
%
%// Function common to all digit buttons
function digit_button(digit)
{
  entry = entry * multiplier + digit / divisor;
  if (divisor >= 10) {
    divisor = divisor * 10;
   } else {
    multiplier = 10;
  }
  return entry;
}
%
%// Function common to all arithmetic function buttons
function func_button(req_func)
{
  if (func != """") {
    update_result();
  } else {
    if (entry != 0) {
      accum = entry;
    }
    func = req_func;
    clear_entry();
  }
}
%
%// Function to update result
function update_result()
{
  switch (func) {
    case ""PLUS"":
```

```
     accum = accum + entry;
     break;
    case ""MINUS"":
     accum = accum - entry;
     break;
    case ""MULT"":
     accum = accum * entry;
     break;
    case ""DIV"":
     if (entry != 0) {
       accum = accum / entry;
     }
     break;
    case ""INV"":
     if (accum != 0) {
       accum = 1.0 / accum;
     }
     break;
    case ""PLUSMINUS"":
     accum = -accum;
     break;
    case ""SQRT"":
     if (accum > 0) {
       accum = Math.sqrt(accum);
     }
     break;
    case ""SQR"":
       accum = accum * accum;
     break;
  }
  func = """";
  clear_entry();
}"}}

\def\cbutt#1#2%
{%
  \colorbox{#2}{\vbox to 1cm{\vss\hbox to 1cm{\hss
  \huge#1\hss}\vss}}
```

```
}%
\def\onbutt{green}
\def\offbutt{gray}

\def\shipform#1#2{\setbox1=\hbox{#1}%
\immediate\shipout\box1
\xdef#2{\the\formcount}%
}

\shipform{\cbutt{$1$}{\offbutt}}\bOne
\shipform{\cbutt{$2$}{\offbutt}}\bTwo
\shipform{\cbutt{$3$}{\offbutt}}\bThree
\shipform{\cbutt{$4$}{\offbutt}}\bFour
\shipform{\cbutt{$5$}{\offbutt}}\bFive
\shipform{\cbutt{$6$}{\offbutt}}\bSix
\shipform{\cbutt{$7$}{\offbutt}}\bSeven
\shipform{\cbutt{$8$}{\offbutt}}\bEight
\shipform{\cbutt{$9$}{\offbutt}}\bNine
\shipform{\cbutt{$0$}{\offbutt}}\bZero
\shipform{\cbutt{$+$}{\offbutt}}\bPlus
\shipform{\cbutt{$-$}{\offbutt}}\bMinus
\shipform{\cbutt{$\times$}{\offbutt}}\bTimes
\shipform{\cbutt{$/$}{\offbutt}}\bDiv
\shipform{\cbutt{$1$}{\onbutt}}\bOneF
\shipform{\cbutt{$2$}{\onbutt}}\bTwoF
\shipform{\cbutt{$3$}{\onbutt}}\bThreeF
\shipform{\cbutt{$4$}{\onbutt}}\bFourF
\shipform{\cbutt{$5$}{\onbutt}}\bFiveF
\shipform{\cbutt{$6$}{\onbutt}}\bSixF
\shipform{\cbutt{$7$}{\onbutt}}\bSevenF
\shipform{\cbutt{$8$}{\onbutt}}\bEightF
\shipform{\cbutt{$9$}{\onbutt}}\bNineF
\shipform{\cbutt{$0$}{\onbutt}}\bZeroF
\shipform{\cbutt{$+$}{\onbutt}}\bPlusF
\shipform{\cbutt{$-$}{\onbutt}}\bMinusF
\shipform{\cbutt{$\times$}{\onbutt}}\bTimesF
\shipform{\cbutt{$/$}{\onbutt}}\bDivF
\shipform{\cbutt{$=$}{\offbutt}}\bEqual
```

```
\shipform{\cbutt{$=$}{\onbutt}}\bEqualF
\shipform{\cbutt{{\color{red}C}}{\offbutt}}\bCan
\shipform{\cbutt{{\color{red}C}}{\onbutt}}\bCanF
\shipform{\cbutt{$1/x$}{\offbutt}}\bInv
\shipform{\cbutt{$1/x$}{\onbutt}}\bInvF
\shipform{\cbutt{$\pm$}{\offbutt}}\bPM
\shipform{\cbutt{$\pm$}{\onbutt}}\bPMF
\shipform{\cbutt{$\sqrt{x}$}{\offbutt}}\bSqrt
\shipform{\cbutt{$\sqrt{x}$}{\onbutt}}\bSqrtF
\shipform{\cbutt{$x^2$}{\offbutt}}\bSqr
\shipform{\cbutt{$x^2$}{\onbutt}}\bSqrF

\def\calcform#1#2#3#4#5#6{%
  \special{!field k=b,w=#1,h=#2,pnu=#3,pru=#4,%
  i="#5",xu=<JS "#6">}%
}

\def\buttontemplate#1{\vbox to 1.5cm{%
  \hbox to 1.5cm{\hskip0.25cm#1\hss}\vfil}}

\immediate\special{bc"ffffff}

\newbox\calc
\setbox\calc=\hbox{\begin{tabular}{ccccc}
 \buttontemplate{\calcform{1cm}{1cm}{\bSeven}{\bSevenF}
{seven}{display.value=digit_button(7)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bEight}{\bEightF}
{eight}{display.value=digit_button(8)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bNine}{\bNineF}
{nine}{display.value=digit_button(9)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bDiv}{\bDivF}
{div}{func_button(""DIV"")}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bCan}{\bCanF}
{clear}{all_cancel();}}
 \\
 \buttontemplate{\calcform{1cm}{1cm}{\bFour}{\bFourF}
{four}{display.value=digit_button(4)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bFive}{\bFiveF}
```

```
{five}{display.value=digit_button(5)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bSix}{\bSixF}
{six}{display.value=digit_button(6)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bTimes}{\bTimesF}
{times}{func_button(""MULT"")}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bSqrt}{\bSqrtF}
{sqrt}{func_button(""SQRT""); update_result();
 display.value=accum;}}
 \\
 \buttontemplate{\calcform{1cm}{1cm}{\bOne}{\bOneF}
{one}{display.value=digit_button(1)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bTwo}{\bTwoF}
{two}{display.value=digit_button(2)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bThree}{\bThreeF}
{three}{display.value=digit_button(3)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bMinus}{\bMinusF}
{minus}{func_button(""MINUS"")}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bSqr}{\bSqrF}
{sqr}{func_button(""SQR""); update_result();
 display.value=accum;}}
 \\
 \buttontemplate{\calcform{1cm}{1cm}{\bZero}{\bZeroF}
{zero}{display.value=digit_button(0)}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bPM}{\bPMF}
{plusminus}{func_button(""PLUSMINUS"");
 update_result(); display.value=accum;}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bInv}{\bInvF}
{inv}{func_button(""INV""); update_result();
 display.value=accum;}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bPlus}{\bPlusF}
{plus}{func_button(""PLUS"")}} &
 \buttontemplate{\calcform{1cm}{1cm}{\bEqual}{\bEqualF}
{equal}{update_result(); display.value=accum;}}\\
\end{tabular}}

\dimen0=\the\wd\calc\relax
\advance\dimen0 -0.5cm
\edef\CW{\the\dimen0}
```

```
\fcolorbox{gray}{darkgray}{%
\vbox{%
\hbox{\vrule width0pt height .2in depth.7in
  \hskip0.25cm\special{!field k=t,w=\CW,h=0.5in,
  i="Display",v="20",b=ffffff,o=ffff00,c=000000,q=r,
  xf=<JS "AFNumber_Format(2, 0, 0, 0, """", true);">,
  xk=<JS "AFNumber_Keystroke(2, 0, 0, 0, """", true);">}}%
\box\calc}%
}
```