More dynamic websites thanks to AJAX

AJAX POWER

AJAX technology adds dynamic elements to enhance sluggish websites. All it takes is a server-side Perl program and some client-side JavaScript code. **BY MICHAEL SCHILLI**

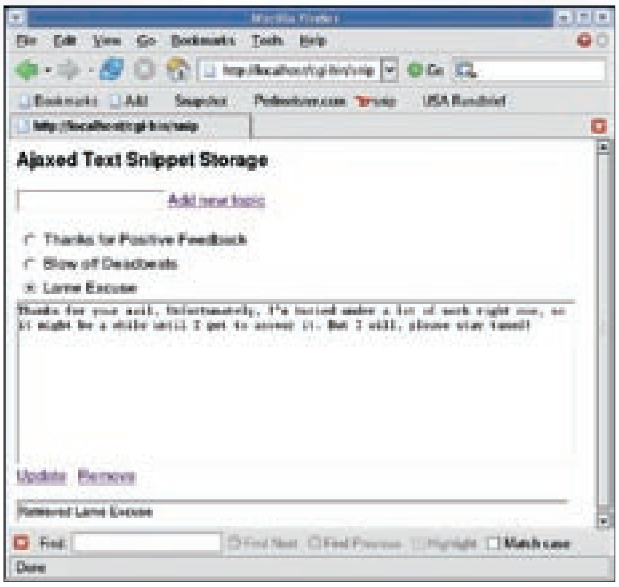
Web developers were rudely awakened when Google introduced its Maps service. All of a sudden users could move maps dynamically, as though the application were running as a local GUI rather than in a browser. All of a sudden, time-consuming client-server round trips were hardly noticeable, since the current page didn't need to be reloaded in order to reflect state changes in the application. Today, Ajax applications are sprouting all over the web. The beta release of Yahoo! Webmail, for example, looks very much like a desktop application; you have to take a very close look to see that your web browser is running the show.

AJAX (Asynchronous JavaScript and XML) is based on dynamic HTML and client-side JavaScript. The *XMLHttpRequest* object, originally added by Microsoft and flying under the radar until Google helped it to fame, allows a JavaScript script downloaded from a website to exchange data asynchronously with the web server. It then dynamically smuggles this data into the HTML page, meaning that only minor changes need to take place on the page.

Figure 1 shows a sample application that manages text snippets commonly used in email and serves them up in a text field for cutting and pasting. You can select *Add new topic* to add a new text snippet to be stored on the server. *Update* sends the corresponding text to the server and *Remove* deletes the selected entry.

The page is loaded just once by the browser. The underlined links may look like hyperlinks, and they are clickable, but they won't cause the browser to jump to a new URL. Instead, they simply execute the JavaScript code specified by the *OnClick* handler, talking to the server behind the scenes.

The *CGI::Ajax* Perl module by Brent Pedersen makes this mechanism quite simple to implement. The module defines a client-server protocol (in JavaScript and Perl), which the client-side JavaScript code can use to call server-side Perl functions by referencing their names and parameter lists. A *XMLHttpRequest* (*ActiveXObject("Microsoft.XMLHTTP") on IE*) JavaScript object enables the browser to sends a GET request to the server-side CGI script. The request triggers a previously specified Perl function.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Bookmarks Add Snapshot Perlmeister.com snip USA Rundbrief
http://localhost/cgi-bin/snip

Ajaxed Text Snippet Storage

Add new topic

( ) Thanks for Positive Feedback
( ) Blow off Deadbeats
(•) Lame Excuse

```
Thanks for your mail. Unfortunately, I'm buried under a lot of work right now, so
it might be a while until I get to answer it. But I will, please stay tuned!
```

Update Remove

Removed Lame Excuse

Find: Find Next Find Previous Highlight Match case
Done

Figure 1: The text snippet manager in a browser.

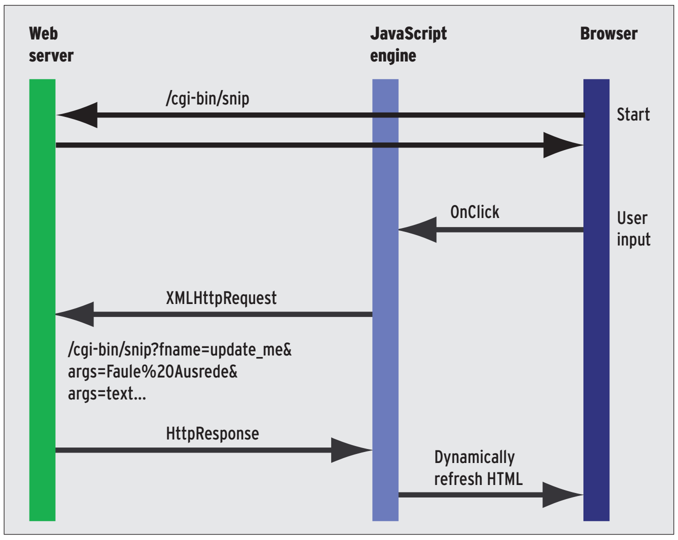


Figure 2: Communication between the browser, the JavaScript engine, and the web server.

which returns one or more values. The JavaScript code picks them up and refreshes predefined fields in the browser GUI.

The *CGI::Ajax* object created by *CGI::Ajax->new('display' => \&display, ...);* in the CGI script ensures that the HTML sent back will contain a JavaScript section. The web application uses a JavaScript function named *display* to fill the browser's text area with the currently selected text snippet. Correspondingly, a Perl function named *display()* function is defined on the server; the JavaScript handler later calls it using an HTTP request. A radio button defined in HTML with the OnClick handler, *OnClick="display(['Lame Excuse'], ['tarea', 'statusdiv'])"*, calls the JavaScript function *display()* and passes the two specified arrays to the function. The first array contains the *id* attribute of the selected radio button; its text value (*Lame Excuse*) gets passed to the server function. The second array contains the *id* attributes of the HTML tags, which the handler refreshes with the return value from the server function after completing the request. This way, both the text area and the status text field get updated with fresh server data.

In our case, the radio button with the *id* "Lame Excuse" also has the property *VALUE="Lame Excuse"*, meaning that the server-side Perl *display()* function (line 12 in Listing *snip*) is passed this text string as its first parameter. *display()* does nothing apart from retrieving the text block to match "Lame Excuse" from the server-side cache and sending it back to the browser along with a status message. This is where the JavaScript event handler cuts in again, refreshing the large text field (*id='tarea'*) and the status field down at the bottom (*id='statusdiv'*) with the strings re-

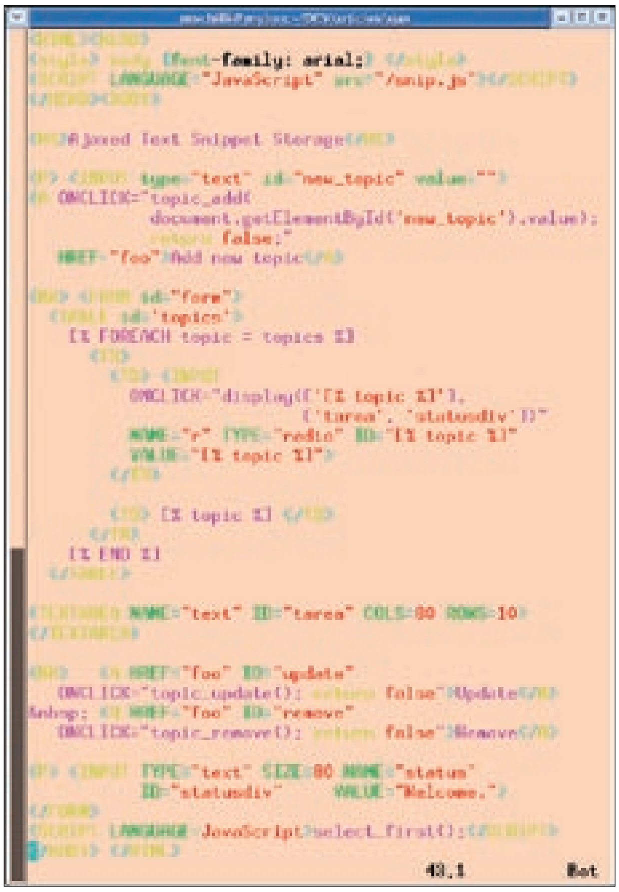
Listing 1: snip

```
01 #!/usr/bin/perl -w
02 use strict;
03 use CGI;
04 use CGI::Ajax;
05 use Cache::FileCache;
06 use Template;
07
08 my $cache =
09   Cache::FileCache->new();
10
11 ################################
12 sub display {
13 ################################
14  my ($topic) = @_;
15
16  return $cache->get($topic),
17    "Retrieved $topic";
18 }
19
20 ################################
21 sub remove_me {
22 ################################
23  my ($topic) = @_;
24
25  $cache->remove($topic);
26  return "Deleted $topic";
27 }
28
29 ################################
30 sub update_me {
31 ################################
32  my ($topic, $text) = @_;
33
34  $cache->set($topic, $text);
35
36  my $disptext = $text;
37  $disptext =
38    substr($text, 0, 60)
39    . "..."
40    if length $text > 60;
41  return
42    "Topic '$topic' updated "
43    . "with '$disptext'";
44 }
45
46 ################################
47 sub show_html {
48 ################################
49  my $template =
50    Template->new();
51
52  my @keys =
53    sort $cache->get_keys();
54
55  $template->process(
56   "snip.tmpl",
57   { topics => \@keys },
58   \my $result)
59    or die $template->error();
60
61  return $result;
62 }
63
64 ################################
65 # main
66 ################################
67 my $cgi = CGI->new();
68 $cgi->charset("utf-8");
69
70 my $pjx = CGI::Ajax->new(
71  'display'   => \&display,
72  'update_me' => \&update_me,
73  'remove_me' => \&remove_me
74 );
75 print $pjx->build_html($cgi,
76  \&show_html);
```

Listing 2: snip.js

```
// ############################
function topic_add(topic) {
// ############################
  var itemTable = document.getElementById("topics");
  var newRow    = document.createElement("TR");
  var newCol1   = document.createElement("TD");
  var newCol2   = document.createElement("TD");
  var input     = document.createElement("INPUT");

  if(topic.length == 0) {
      alert("No topic name specified.");
      return false;
  }

  input.name    = "r";
  input.type    = "radio";
  input.id      = topic;
  input.value   = topic;
  input.onclick = function() {
      display([topic], ['tarea', 'statusdiv']);
  };
  input.checked = 1;
  newCol1.appendChild(input);

  var textnode = document.createTextNode(topic);
  newCol2.appendChild(textnode);

  itemTable.appendChild(newRow);
  newRow.appendChild(newCol1);
  newRow.appendChild(newCol2);

  document.getElementById('tarea').value = "";
  document.getElementById('new_topic').value = "";

  return false;
}

// ############################
function topic_update() {
// ############################
  if(!id_selected()) {
    alert("Create a new topic first");
    return;
  }
  update_me( [ id_selected(), 'tarea' ],
             'statusdiv');
}

// ############################
function topic_remove() {
// ############################
  var sel = id_selected();

  if(!sel) { alert("No topic available");
    return;
  }

  remove_me([sel], 'statusdiv');

  var node = document.getElementById(sel);
  var row  = node.parentNode.parentNode;
  row.parentNode.removeChild(row);
  select_first();
}

// ############################
function select_first() {
// ############################
  var form = document.getElementById("form");
  if(! form.r) { return; }
  if(! form.r.length) {
    form.r.checked = 1;
    if(! document.getElementById(id_selected()) ) {
       document.getElementById('tarea').value = "";
      return;
    }
    display([id_selected()], ['tarea', 'statusdiv']);
  }

  for(var i = 0; i < form.r.length; i++) {
    form.r[i].checked = 1;
    break;
  }
  display([id_selected()], ['tarea', 'statusdiv']);
}

// ############################
function id_selected() {
// ############################
  sel = id_selected_first_pass();

  if(! document.getElementById(sel) ) {
    document.getElementById('tarea').value = "";
    return;
  }
  return sel;
}

// ############################
function id_selected_first_pass() {
// ############################
  var form = document.getElementById("form");
  if(! form.r) { return 0; }
  if(! form.r.length) { return form.r.id; }

  for(var i = 0; i < form.r.length; i++) {
    if(form.r[i].checked) {
      return form.r[i].id;
    }
  }
  alert("Selected ID is unknown");
  return 0;
}
```

please continue on p76



```
<HTML><HEAD>
<style> body {font-family: arial;} </style>
<SCRIPT LANGUAGE="JavaScript" src="/snip.js"></SCRIPT>
</HEAD><BODY>

<H2>Ajaxed Text Snippet Storage</H2>

<P> <INPUT type="text" id="new_topic" value="">
<A ONCLICK="topic_add(
            document.getElementById('new_topic').value);
            return false;"
  HREF="foo">Add new topic</A>

<BR> <FORM id="form">
 <TABLE id='topics'>
  [% FOREACH topic = topics %]
   <TR>
    <TD> <INPUT
      ONCLICK="display(['[% topic %]'],
                       ['tarea', 'statusdiv'])"
      NAME="r" TYPE="radio" ID="[% topic %]"
      VALUE="[% topic %]">
    </TD>

    <TD> [% topic %] </TD>
   </TR>
  [% END %]
 </TABLE>

<TEXTAREA NAME="text" ID="tarea" COLS=80 ROWS=10>
</TEXTAREA>

<BR>    <A HREF="foo" ID="update"
  ONCLICK="topic_update(); return false">Update</A>
&nbsp; <A HREF="foo" ID="remove"
  ONCLICK="topic_remove(); return false">Remove</A>

<P> <INPUT TYPE="text" SIZE=80 NAME="status"
           ID="statusdiv"     VALUE="Welcome.">
</FORM>
<SCRIPT LANGUAGE=JavaScript>select_first();</SCRIPT>
</BODY> </HTML>
                                          43,1          Bot
```

Figure 3: The HTML template that snip processes using the template toolkit.

turned by *display()*. All of this is neatly abstracted by *CGI::Ajax*, which sends the required JavaScript code to the browser, and sets up the server-side handler for accessing the Perl functions.

Client-side, *snip* does more than just refreshing text fields. If the user deletes one of the headings by clicking on *Remove*, this not only deletes the radio button with the heading, but also selects the first heading in the remaining list and loads the matching text block from the server. *CGI::Ajax* can't handle client-side trickery like this yet, but additionally defined JavaScript functions will help.

On a positive note, *CGI::Ajax* is extremely easy to use. As Listing 1 shows, you only need to define functions for the various client actions (remove/update/display) and provide a *show_html* function, which returns the client application HTML on initial loading.

HTML and Perl Separated

The *snip* Perl script takes the HTML to be sent from the *snip.tmpl* template, which is shown in Figure 3. The template toolkit loads the template and provides a number of constructs that let you embed simple *for* loops or conditions.

It uses *[% FOREACH topic = topics %]* to iterate over the @topics array previously provided by *snip* with the text block headings and outputs a number of radio buttons, each in a separate table row. *[% topic %]* returns the value of the *topic* template variable each time. The *id* property of each radio button is set to the text string in the heading, and the *OnClick* handler calls the *display()* function described previously, which exists both in the client-side JavaScript and in the Perl universe on the server.

The *select_first()* function selects the first entry in the list of radio buttons and requests the text block for the appropriate heading. It first calls the method *document.getElementById* to search for the HTML tag with the ID of *form* (the HTML form in *snip.tmpl*). All radio buttons are named *r*, so *form.r* should be an array, holding entries for each radio button found. If the list of headings is empty, *select_first()* obviously selects nothing; nor does it talk to the server.

The tag type *<A>* hyperlinks used in *snip* should have *return false* as the last action in their *OnClick* handlers. This ensures that the browser executes the JavaScript code assigned to the link, rather than following the bogus *HREF* attribute.

The template loads the JavaScript library *snip.js* first (Listing 2); the library provides a number of functions to allow the GUI to run properly. The JavaScript *topic_add()* function in *snip.js* expects the string from a heading and appends a new entry with this name to the end of the radio button table.

topic_remove() deletes a heading from the list of radio buttons and sends a request to the server, which in turn deletes the text snippet from the cache.

id_selected() outputs the *id* property of the selected radio button, that is, the heading of the entry the user wants to see. If the list is empty, Firefox can become confused and still return a value. To work around this bug, *id_selected()* rechecks the result of *id_selected_first_pass()* and returns *undefined* if it catches Firefox cheating.

If the list of radio buttons has two or more entries, *form.r.length* returns the list size. If the list only has a single entry, *form.r.length* returns an undefined value. If the list is empty, *form.r* is undefined. The *checked* flag can then either be verified via *form.r.checked*, or via element "i" in the array: *form.r[i].checked*. After this, *select_first()* then calls the *display()* function to get the text for the selected heading from the server.

Rough Edges

This script is only intended as an example. To keep it simple, code for handling error conditions has been left out of the script. You might experience compatibility problems with browsers other than Firefox.

Installation

The script requires the *Class::Accessor*, *CGI::Ajax*, and *Template* modules from CPAN. Then add the *snip* script (executable!) and the *snip.tmpl* template (in the *cgi-bin* on the web server) and the *snip.js* JavaScript script directly below the document root (typically *htdocs*).

If the terminal used for cutting and pasting does not speak UTF-8, you might like to comment out line 68 in *snip* to tell the web server to send *charset=iso-8859-1* in the CGI header instead. And if you prefer not to have the document cache reside below */tmp*, you will need to specify your preferred directory by setting *my $cache = Cache::FileCache->new({cache_root => "/path"})* in line 9. Finally, point your browser to *http://server/cgi-bin/snip*, and – assuming that JavaScript is enabled – the text snippet function should run, talk to the server, and keep the snippet repository on the server up to date. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/62/Perl*