Minserver simulations

A development book by the creator of the program

Abrahm Koks
abrahmwills@yahoo.com

2017-08-12

Getting started

Copy and include ‘minsrv.js’ in your web project folder, assuming you have the file or clone from the github repository using git

```
github.com/awills/minserver.git
```

You can also download the code manually from github page on a browser

```
www.github.com/awills/minserver
```

Congratulations and thank you for choosing minserver for your web development today

What is Minserver

Minserver is a program that enables the trendy features such as interactive page updates and dynamic grids in our web development

Interactive page update enables us to update our page with new content using user interactivity without a postback to the server

Dynamic grid is a visual structuring technique that can scale the contents of pages with the browser viewport

Hello world

We will learn how to use css and minserver to define, access and apply content to update our pages

Create html file 'simulation0' and enter this within style tags, we will define content for the ready page using css properties

```
...simulation0.html...

#quote:before{
      content:"hello world, am css"
}
```

Apply the content to the body element and open on a browser

```
...simulation0.html...

<body id="quote"></body>
```

You should see a hello world text on the screen, congratulations.

Create html file 'simulation1', enter this in the head section and correct 'path' in the urls

```
...simulation1.html...

<script type="text/javascript" src="path/minsrv.js"></script>
<script type="text/javascript">

      minsrv("path/hrv0.html#quote", "<p style='background-
      color:wheat;      width:&width;;      height:&height;;'>&greeting;
      &techused;</p>") ;

</script>
```

Apply the content to a page element

```
...simulation1.html...

<body>
      <div id="quote"></div>
</body>
```

Create html file 'hrv0' without a head section, we will define content for the ready page in this file using html attributes on a selected element, enter this within body tags

```
...hrv0.html...

<img techused="minserver" greeting="hello world, am" width="128px"
height="128px" />
```

Some of the attributes defined on the ‘img’ element are valid html, you can use non html attributes when defining contents

Open ‘simulation1’ on the browser, you should see a hello world text on the screen

On the two simulations we did and their similarities

You saw how we defined contents in properties or attributes, respective to the web technology we use, and their application to ready pages using html ids, we only did not know how the browser accessed our disclosed content in the css example

Constructing html

We use the program by calling the global function 'minsrv' within script tags or an external file

The required arguments are, a url, html string and optional interface function for configuring program behaviours

The url is declared in the syntax

```
path/file#frag
```

where 'file' defines our content and could be a preferred server file, there is an additional component in the url, a fragment is mandatory for specifying which page element, structure etc the content appears in

Fragments can cause serious debugging issues for pages, if this is your concern, minserver does not have any effect on your page

The html string gives access to content defined by the program and yourself

Minserver uses html syntax for special characters to access content, some spec chars in html are

```
&gt; (>)
&lt; (<)
&quot; (")
```

Each html spec char represent actual keys on screen and begin with ampersand '&' closing with semi-colon ';'

We replace the name of the html special character with the name of our defined content and avoid any naming conflictions with html

```
&techused; ("minserver")
&greeting; ("hello world, am")
&width; ("128px")
&height; ("128px")
```

Reuse your defined contents to create structures that fit in with the rest of the page by including markup, css etc in the string

```
"<p>&greeting &techused;</p>"
"<p style="background-color:pink; width:&width;;
height:&height;;">&nbsp;</p>"
```

Minserver enable us to access program generated indexations for our content using the spec char 'bullets'

Assuming you have multiple image heads declared in your content page, you should see further numeration from the example

```
    "<p>content&bullets;</p>" >> (“content0”)
```

Indexation and identification of content go hand in hand, consider the example

```
    "<p id="content&bullets;"><a href='#'
    onclick='alert(content&bullets;.textContent);'>click
    content&bullets;</a></p>"
```

Open on a browser and click on the links

Reconstructing urls

Urls are links to files, pages in our project, either used by minserver or not, sometimes a single page provide a length of content for our update needs, but to handle interactions that demand new content, we have to maintain multiple of these pages and learn how to include them on intervals of interactivity

The first argument of the 'minsrv' function enable us to set only one of those pages, we will learn how to reconstruct urls for the purpose of pointing to other pages for our updates

There are two ways in which urls can be reconstructed after being defined in the function call

By using the anchor property of the interface function or by targeting

Targeting enable us to use already defined anchor links on the page to reconstruct the url of the current minserver, as long as the link maintains a path and fragment in common with the program

Create html file 'hrv1' from 'hrv0', enter this in the body section

```
<img techused="javascript" greeting="hello world, am" width="160px"
height="160px" />
<img techused="php" greeting="hello world, am" width="150px"
height="150px" />
```

Open 'simulation1' on the editor, and begin a third argument in the function call, enter this

```
function(s){

      s.preventDefault() ;

      // window.trg0
      trg0=s.target ;
}
```

Update the html string of the function to this, and open on a browser, click on the links to update the ready page with contents, we are going to define a url on the anchor element to point to our second page for the updating content

```
"<p id='content&bullets;' style='background-color:wheat;width:&width;;
height:&height;;'>&greeting;   &techused;   <a   href='hrv1.html#quote'
onclick='trg0(event);'>click content&bullets;</a></p>"
```

The link fired a reference function, causing minserver to use its url for work, and suppress the expected default behaviour of the anchor element, so the browser did not reload as expected

```
onclick='trg0(event);'
```

A special object is passed to the interface function consisting of all states on the current process, find out which link is active in the current minserver using 'anchor' property

```
s.anchor
```

Another way to reconstruct urls does not require the use of page links and external behaviours such as clicks, but to explicitly update the url using the anchor property in the interface function

Minserver can cause pages to update using scroll detection, at the end of a page common to web standards whenever the window has an active scrollbar

Open 'simulation1' on the editor and append this in the body section, we will try to activate the scroll bar on the document

```
<a href='hrv1.html#quote' onclick='trg0(event);'>click for more</a>
<div style="height:500px"></div>
```

Create html file 'hrv2' from 'hrv0' and enter this within body tags

```
<img techused="ruby & rails" greeting="hello world, am" width="100px"
height="100px" />
<img techused="perl" greeting="hello world, am" width="250px"
height="250px" />
<img techused="node" greeting="hello world, am" width="300px"
height="300px" />
<img techused=".net" greeting="hello world, am" width="180px"
height="180px" />
```

Update the html string in the 'minsrv' function to this

```
<p style='background-color:wheat; width:&width;; height:&height;;'>
&greeting; &techused;</p>
```

Update the interface function to this, open on a browser and repeat the steps, scroll to the bottom, click on the link

Please correct 'path' in the url

```
function(s){

      // window.trg0
      trg0=s.target ;

      s.anchor.href="path/hrv2.html#quote" ;
}
```

Fragment are strategic in delaying further update from the same page provided there aren't changes in the minserver url, continuous changes to any part of this url, in query strings, pathnames etc is significant in making pages that update continuously with interactivity

Minserver is dependent on the browser behaviour to url fragment in handling duplications and inconsistencies in updates

Program behaviours

We can alter the program against its intended use, to stop interactive updates, change how contents are applied or serve content for other minservers

You can stop updates to a page by leaving the minserver url static otherwise call 'preventDefault' from the interface function

```
      s.preventDefault()
```

Alter how contents are applied

By default updates are always appended, that is, one after another into containing structures, call 'overwrite' to overwrite when applying updates

```
      s.overwrite()
```

Call other minservers

This is known as serving, and it enable us update multiple parts of a page at the same time with the current minserver

Create html file 'simulation2' and enter within script tags, we are going to setup multiple minservers for the page, please correct 'path' in the urls

```
      minx=minsrv("path/blank.html#reply", "<p style='background
      -color:wheat;     width:&width;;     height:&height;;'>     &greeting;
      &techused;</p>", function(s){

            s.preventDefault() ;
            s.anchor.href="path/hrv2.html#reply" ;
      }) ;

      minsrv("path/blank.html#quote", "<p style='background
      -color:wheat;     width:&width;;     height:&height;;'>     &greeting;
      &techused;</p>", function(s){

            s.preventDefault() ;

            // window.trg0
            trg0=s.target ;

            if(s.type=='click') s.serve('’,0, minx) ;
      }) ;
```

Enter within body tags

```
<div id="quote"></div>
<div style="background-color:pink;" id="reply"></div>
<a href='hrv1.html#quote' onclick='trg0(event);'>click for more</a>
```

Create html file 'blank' without any textual entry

Open 'simulation2' on the browser and click on the link, congratulations, you should see the blocked part of the page also has updates

minsrv function returns a unique id which identifies the current minserver

```
minid=minsrv(...) ;
```

The blank page is meant to supply no content during the load process, this is reconstructed afterwards

```
"path/blank.html#reply"
```

Find out the page behaviour for the current process using 'type'

```
s.type=='click'
```

Call 'serve' from the current minserver to load another minserver, required is an optional url for the minserver, Boolean value to overwrite its contents and its unique id

```
serve('', 0, minx) ;
```

Minserver interactive update is restricted to certain page behaviours such as load, click and programming like scroll detection, other behaviours, such as resize, are in common with the grid functionality

Grids

Grids are used to visually present content in web pages in a modular way that is easy to duplicate, fixed or fluid grids can be done using css or javascript, in the later, there is a means for controlling the appearance of the grid through variable initialisations

The 'grid' function activates the functionality for the current minserver, and can be used to change the appearance of the grid

```
s.grid('gutterwidth', 16) ;
s.grid('gutterheight', 16/2) ;
```

And get live states of the process

```
s.grid('capacity') ;
```

The functionality targets child elements in a container and absolutely positions them into visual grids that is manageable and require little use of css

The container is specified in the minserver url, we only need to change the css display property of child elements to 'absolute', that is, if we want any effect at all

css like this can save the day

```
#quote{
      margin:0 auto;
      postion:relative;
}

#quote > *{
      postion:absolute;
      width:200px !important; /*fixed width*/
}
```

The resize behaviour is common with grids and enable contents to scale dynamically to different screen sizes, we can control this using two special properties 'mincapacity' and 'maxcapacity' which can be set to the same value

```
function(s){

      // grids

      s.grid(“mincapacity”, 4) ;
      s.grid(“maxcapacity”, 4) ;

      if(s.type=='resize') return ;

      // interactive update
}
```

Summary

Other states from the grid functionality and interactive updates are dumped in the interface function and can be accessed lively using properties

As for integrating with the server, you only have to learn about url queries for separated page contents, so you don't have to manage these multiple files on your hard disk

Your copy of minserver repository contain a functional documentation you should refer to for more information, it is intended that you use this book along with the documentation or other books about minserver during your development

Goodluck.