Pushing and Popping with the History API

Tuesday, November 15th, 2011 by Mike Robinson.

Until recently, we developers couldn't to do much with the state and history of the browser. We could check the number of items in the history and push users forwards and backwards, but this provides little benefit to the user. With the rise of more dynamic web pages, we need more control. Thankfully, HTML5 gives us that control by extending the JavaScript History API.

What's the point?

It goes without saying that URLs are important. They're *the* method of accessing the vast collections of information and resources on the web, and more recently, they've begun representing the intended state of a web application. You can copy these URLs and share them with your friends or use them to create links from any HTML document. They're the veins of the web, and they need to be looked after.

Previously, the JavaScript History API offered some very simple functionality:

```
// Check the length of the history stack
console.log(history.length);

// Send the user agent forward
console.log(history.forward());

// Send the user agent back
console.log(history.back());

// Send the user agent back (negative) or forward (positive)
// by a given number of items
console.log(history.go(-3));
```

With dynamic Ajax web applications, where the browser updates the page in parts instead of changing location entirely, it's difficult to give the user a URL to bookmark or share the current application state. Fragment identifiers, like those used on this article's headings via the `id` attribute, provide some state information, but they're entirely dependent on client-side scripts.

The changes to the History API are intended to give developers ways to push history items to the browser so the native back and forward actions can cycle through those items. These history items can also hold data that you can later extract to restore the page state.

> Pages can add state objects between their entry in the session history and the next ("forward") entry. These are then returned to the script when the user (or script) goes back in the history, thus enabling authors to use the "navigation" metaphor even in one-page applications.
>
> – *WHATWG*

If the user copies or bookmarks a stateful URL and visits it later, your back-end can be configured to interpret such a URL and jump the user right to the correct page and/or state.

In this article, I'll cover the client-side use of the History API, so make sure you set up your server to work with the new URLs. If you've already built an accessible website that provide these entry points, you're laughing!

Those *f#!king* hashbangs…

You may have already seen articles fussing over the adoption of the "hashbang" (#!) pattern on sites like Twitter. This technique updates the address bar with a fragment identifier that can then be used by JavaScript to determine which page and state should be displayed.

This works as a method of creating a bookmarkable, shareable URL for a page's state in the absense of a standard API. While the Twitter implementation accepts both `http://twitter.com/#!/akamike` and `http://twitter.com/akamike`, it has some disadvantages:

- The fragment identifier is only accessible on the client side. This means that only JavaScript can utilise it, so browsers without JavaScript enabled are out of luck.
- As the server does not receive the path following the hashbang, removing that JavaScript drops support for all those URLs. That's a lot of broken links, so you're stuck with that JavaScript *forever*.
- It's ugly. It's a hack and it looks like one.

The hashbang was never intended to be a long-term solution, so don't rely on it. If you do use hashbangs, be prepared to deal with the consequences (and possible backlash from web purists).

Making History

These examples will build on top of each other. We'll start with a basic HTML document with some inline styles and scripts for your convenience.

For a simple HTTP server, open the command line, `cd` to the directory you would like to serve, run `python -m SimpleHTTPServer 8080`, then open `localhost:8080` in your browser. Alternatively, try a bundled setup like XAMPP or MAMP.

Save this file and open it in your favourite editor. It must be accessed via HTTP, so that means you need either a local server (e.g. `http://localhost/`) or an online web server you can upload to. **Viewing the file directly using your browser's Open File function will not work**, since it uses the `file://` protocol and not HTTP. Also be sure to update the `href` attributes on each of the navigation links to ensure the correct directory structure is used. Personally, I'm viewing the demo locally at `http://localhost/history`.

We'll be working exclusively within the `<script>` element at the end of the `<body>`. The code includes some simple styles and dynamically changes the content as you click the links. In reality, this could be loaded from your server via `XMLHttpRequest`, but for the purposes of this demonstration I've bundled it up into a self-contained file. The important part is that we have a quick-and-dirty dynamic page to work with, so let the fun begin!

At the moment there, is no bookmarkable URL for the different states of this page. If you click around the navigation items, then click Back in your browser, you won't be taken back to the previous state and may even be taken away from the page to whatever you viewed before (depending on your browser). It would be nice if you could share "Socks" with your friends, right? We can do that via `history.pushState()`.

The `history.pushState()` method takes three parameters:

`data`
: Some structured data, such as settings or content, assigned to the history item.

`title`
: The name of the item in the history drop-down shown by the browser's back and forward buttons. (Note: this is not currently supported by any major browsers.)

`url` *(optional)*
: The URL to this state that should be displayed in the address bar.

With these parameters, you can define the state of the page, give that state a name, and even provide a bookmarkable address, as if the page had reloaded entirely. Let's dive right in and add this to the `clickHandler` function, right above the `return` statement:

```
function clickHandler(e) {
  /* Snip... */

  // Add an item to the history log
  history.pushState(data, event.target.textContent, event.target.href);

  return event.preventDefault();
}
```

The single line of code we added informs the `history` object that:

- we want to add an item to the log,
- it should remember the data that we've already loaded,
- it should assign a name to this state based on the text of the link we clicked (even though this isn't used — it's good to get into the habit of recording a name for the state), and
- it should update the address bar with the `href` attribute of that link.

Reload the page in your browser and click a few of the links, keeping an eye on the address bar. Notice how it changes on each click, despite the fact that you aren't actually navigating away from this page. If you also have a look at your history log, you'll see a long list of page titles (in this case "Kittens!" over and over). Provided your server is set up to serve the correct page upon access, the user could copy that URL and paste it into a new browser window to jump straight to that kitten.

At the moment, clicking the back button will pop you through the history items, but the page won't react to these changes. That's because so far, we've only created the history records. How can we allow active users to return to a previous state? We listen to the `popstate` event.

Historical Events in Navigation

The user agent fires a `popstate` event when the user navigates through their history, whether backwards or forwards, provided it isn't taking the user away from the current page. That is, all those `pushStates` we called

will keep the user on the current page, so the `popstate` event will fire for each history item they pop through.

Before the closing `</script>` tag, add a new listener for the `popstate` event:

```
// Revert to a previously saved state
window.addEventListener('popstate', function(event) {
  console.log('popstate fired!');

  updateContent(event.state);
});
```

We attach the event listener to the `window`, which is responsible for firing the event, and pass this event into our handler. We log a message (so we can see when this event is firing), and then we update the content using the state we saved previously. The state is attached to the `event` object via the `state` property.

Open up the page fresh in your browser, click around like before, and then click back. As before, the URL in the address bar changes as you cycle through states, but now the content is also restored back to what it should be. Click forward, and the content is likewise correctly restored.

If you look at the developer console in Chrome when you load the page for the first time, you'll see the `popstate` event fired immediately, before you've even clicked a link. This is because Chrome considers the initial page load to be a change in state, and so it fires the event. In this instance, the `state` property is `null`, but thankfully the `updateContent` function deals with this. Keep this in mind when developing as it could catch you out, especially if other browsers assume this behavior.

Rewriting history

Unfortunately, as fantastic as HTML5 is, it doesn't allow us actual time travel. If it did, I would be going back to my childhood and telling a younger me, "Yes, you should have a slice of cake". Take that as you will.

The History API does, however, allow us to make amends to our history log items. For example, we could update the current state in response to fresh user input in a form. We can do this with `history.replaceState`.

`replaceState` works just as `pushState` does, with the exact same parameters, except that it updates the current entry instead of adding a new one. I can think of one situation in our demo where this could be used: the initial page load. If you click back for long enough, you'll find that going back to the original URL doesn't provide you the original content. Let's fix that by adding the following to the bottom of our script:

```
// Store the initial content so we can revisit it later
history.replaceState({
  content: contentEl.textContent,
  photo: photoEl.src
}, document.title, document.location.href);
```

As this runs when the page loads, it saves the initial page state. We can later load this state when the user browses back to this point via the event listener we set up previously. You can try it out by loading up the page, clicking a few links, and then hitting back until you return to the original URL. The initial content has returned!

Demo

I've set up a demo of our completed code. I've also added a little back-end magic to make our `history.pushState` URLs work like a real site. Remember that the URLs you push should be live URLs that the user can bookmark and share as real entry points to your site.

View the History API demo

Browser support

Up-to-date copies of Chrome (5+), Safari (5.0+), Firefox (4.0+), and Opera (11.50+) have support for the new History API. Even some mobile browsers work just fine, like Mobile Safari on iOS 4+. Unfortunately, IE 9 and below lack support, but it when it arrives.

Safari 5.0 sometimes exhibits one oddity: navigating between states causes the loading spinner to appear and stay even when the state has been loaded. This stops when you navigate away using a link or action that does not involve a state saved by the History API.

Polyfill

A polyfill does exist for the History API. The aptly named History.js uses HTML4's `hashchange` event with document fragment identifiers to mimic the history API in older browsers. If one of the hash URLs is used by a modern browser, it uses `replaceState` to quietly correct the URL.

It sounds like magic, but make sure you're aware of the consequences of using fragment identifiers, as mentioned previously in this article. As such, the author of History.js has put together a guide titled Intelligent State Handling.

Closing thoughts

URLs go beyond just the browsing session of a user. They're historically important markers for resources that could very well remain in use for many years to come. Before you embark on developing your site's JavaScript, you should give thought to the design of your URLs. Make them meaningful and organised. Make sure you can directly access them without JavaScript. Only then should you add your JavaScript to enhance the browsing experience.

- **Category**
  - JavaScript APIs
- **Tags**
  - api
  - history
  - HTML 5
  - html5

Mike Robinson

This article was written by Mike Robinson. A developer at Lift in Reading, England, you can catch him on Twitter or occasionally blogging on his own site, akamike. Beyond the web, Mike is usually gaming or listening to progressive rock.

26 Responses on the article “Pushing and Popping with the History API”

-  *Brian LePore* says:
November 15, 2011 at 3:33 pm

I literally *just* discovered this and worked on this for a client two weeks ago. I even contemplated writing a blog about it because of how cool I found this. This really can be a game changer.

I’m surprised you did not cover the history.state feature? I believe this is the current spec, though it only works in Firefox. The code used here, while it does work, I thought was part of the original version of the spec. Also, worth mentioning is that you are *supposed* to pass in JavaScript object like in your example. Microsoft’s IE blog has examples where a string is passed in. I’ve not tested support on this across browsers, but it’s worth mentioning.

I literally cannot tell you how many hours the whole “Store the initial content so we can revisit it later” issue turned out to be an issue for me. It seems unintuitive to me that the default state that you come to is not in the history stack.

Reply

-  *Xavi* says:
December 9, 2011 at 1:01 pm

I’m trying the demo page on Internet Explorer 9 and I can’t tell any difference to Chrome so it seems to work.

I’ll be using this article for a future project, for sure. Thank you!

Reply

-  *Raphael Sebbe* says:
December 16, 2011 at 6:10 pm

Hi Mike,

thank you for sharing your experience, very informative.

Could you please elaborate on what you did on the server to handle the fake URLs? Are they manual redirects you inserted, or automatic rewrites? How are they converted?

Thank you so much!

Reply

 *Eric* says:
January 6, 2012 at 12:09 am

Hi Mike,
I also have the same question as Raphael. What did you do on the back end in order to handle the URLs?

Thanks!

Reply

 *Raphael Sebbe* says:
January 6, 2012 at 8:31 am

In the mean time, I advanced a bit on that same subject. If your web server is Apache, you can configure it to use mod_rewrite, which allows to convert user (or crawler) typed URLs into other server side URLs:

http://www.mywebsite.com/myproduct/test.html -> http://www.mywebsite.com/index.php?product=myproduct&page=test

This is done by adding a number of rules in the .htaccess at the root of your website.

HTH,

Raphael

Reply

 *Abid Din* says:
February 8, 2012 at 9:28 am

Literally, just released my new site which makes use of this technique. It works beautifully!

It's nice to (maybe) be able to do away with hashes and hashbangs on future ajax-based sites.

I'm really looking forward to seeing how this gets used as it's adopted more widely.

Reply

 *Interaction Designer Craig Dennis* says:
February 8, 2012 at 10:01 am

Great article.

The only issues arise (as is often the case) with cross-browser implementations of this. The history API may be supported by the browsers you mentioned but with varying degrees of success.

Check out History.js (note the capital ‘H’) by Ben Lupton here:https://github.com/balupton/History.js/ which offers a means to use HTML5 history states while accounting for craoos browser inconsistencies.

Also, have some fun with this gist too: https://gist.github.com/854622

Again, great article. Really gets down to the underlying principles.

Reply

 *Alex* says:
February 8, 2012 at 11:59 am

This is great, but if it doesn’t work on IE9 or below it’s pretty useless for production sites currently (without lots of messy hacks and fallbacks!)

Reply

 *Mike Robinson* says:
February 20, 2012 at 4:47 pm

@Raphael + Eric:

My apologises for the late reply! For the purposes of the demo, I used a very basic mod_rewrite and PHP setup.

```
RewriteRule ^(fluffy|socks|whiskers|bob)$ index.php?p=$1 [NC,L]
```

The above rewrite rule masks the demo URLs and passes it into a single PHP file with a query string of what cat the user is attempting to view. The PHP file then checks `$_GET['p']` is valid (always sanitise and check your input as appropriate), then using the pre-defined variables I have set for that cat.

If this were a real application, the requested path could be used to construct database queries (sanitised of course) to pull in more dynamic content. CMS like WordPress use something like this for their “clean URL” implementation.

Hope this helps.

Reply

 *Raphael Sebbe* says:
February 21, 2012 at 10:21 am

Hi Mike,

thanks for sharing. I had gone through a similar route using mod_rewrite in the mean time within .htaccess.

Thx!

Reply


Justin says:
March 30, 2012 at 6:31 pm

Hello Mike,

Would it be possible to see that htaccess file closer. I understand everything going on here, within the history api part, except the mod_rewrite. Do you actually have more pages living somewhere for each of these different cats? Refreshing the page after you click a few of the cats returns a page not found error. This has to do with the htaccess part that I am missing, right? Thanks.

Reply


Alex says:
May 25, 2012 at 4:27 pm

Despite my comment above about poor browser support I decided to switch from hashed URLs (/#/slug) to HTML5 history for http://goproheroes.com and decided to just fall back to normal page loading for IE9 and below using:

```
function supports_history_api() {
return !!(window.history && history.pushState);
}
```

Reply


Marc M says:
September 6, 2012 at 3:41 am

Thanks Mike, your demo helped me out. I had my attempt a bit more complicated than it needed to be.

However, I'm trying to track and detect when the user steps forward and back through the history so I can add appropriate transitions to my page. Unfortunately, the history API doesn't let you know if a forward or back button was clicked on a popstate event. That would be really helpful.

I'm thinking that a cookie might be the only way to attempt to track the order. Saving a previous variable in the state object complicated things and caused browser issues.

If anyone knows of a simple method for detecting if someone has moved forward or back on a popstate, I would love to hear about it.

Thanks again for the great demo.

Reply


Kristian Andersen says:
September 23, 2012 at 3:10 pm

htaccess for html5 history pushstate url routing.

I just found this link that others might find useful.

http://www.josscrowcroft.com/2012/code/htaccess-for-html5-history-pushstate-url-routing/

Reply

•

 *thinsoldier* says:
November 9, 2012 at 10:15 pm

popstate fires whenever you press the back or forward button.

But what even fires when you initially trigger the pushState command via clicking a link for example?

Reply

•

 *adam pery* says:
November 15, 2012 at 7:04 am

For example: http://html5doctor.com/demos/history/

In directory: demos/history/
create .htaccess with next code:

RewriteEngine On
RewriteRule ^(fluffy|socks|whiskers|bob)$ index.php?p=$1 [NC,L]

Reply

•

 *Harrison* says:
December 14, 2012 at 1:06 am

Hi Mike,

Thanks for this tutorial,
I'm following the article to make a basic practice, but stuck in popstate. I can't figure how to use it make the back button work.

here is link and code example

Demo
$('.subject').click(function(){
$.ajax({
url: 'index.php?id=' + $(this).attr('rel'),
success: function(html){
$('.content').append(html);
}
});
var pageurl;

pageurl = ‘index.php?id=’ + $(this).attr(‘rel’);
if(pageurl != window.location){
window.history.pushState({path: pageurl}, “”, pageurl);
}
return false;
});

index.php
<?php
if($_GET['id']){
…
print"”;
}
else{
…
print””
}
?>

grateful for any help,

Harrison

Reply

 *Harrison* says:
December 14, 2012 at 1:07 am

http://stackoverflow.com/questions/13869174/html5-history-popstate-tutorial

Reply

 *George* says:
February 25, 2013 at 7:16 am

Thanks Mike explaining in a simple way. Just wondering, when I check http://html5doctor.com/demos/history/page on IE9 or IE8, history seems working as it works on chrome. However, as you said history API lacks support on IE9 and less, then how is that working. I also, don’t see that you’ve used history.js

Reply

 *Visual* says:
February 28, 2013 at 3:47 pm

Hi
Wondering what is the php code behind to figure out what page you are on?

Is is possible to post the full source

Thanks!

Reply

•

 *Shashank Duhan* says:
March 19, 2013 at 4:29 am

Thanks you made my day. :)

Reply

•

 *Kevin Ashcraft* says:
May 1, 2013 at 7:59 pm

Adding to the list,
thanks Mike!

Those adorable kitten's helped me setup some pushing and popping functions on the soon-to-be released new version of my site!

Thank you!!!

Reply

•

 *Blog Content Writing Services* says:
May 6, 2013 at 4:29 am

When I initially commented I clicked the "Notify me when new comments are added"
checkbox and now each time a comment is added I get several emails
with the same comment. Is there any way you can remove me from that service?
Bless you!

Reply

•

 *Ramón Ramos* says:
September 20, 2013 at 8:39 am

Thanks for the tut!

In your clickhandler function you have something wrong in your post code (it's ok on kittieland though):

1. The event object is declared on the arguments as "e", not "event".

2. There is no updateContent(); anywhere! I had to check your working source to see if I was doing something wrong.

But wait, there's more! …some updates too:

3. Chrome 19+ starts to mingle with the state property of the event and stuff.

As a tip: I used a lame setTimeout() to bind the event listener in order to skip Chrome's eagerness to pop its state.

Thanks again :)

Reply

-  *Stan* says:
October 9, 2013 at 9:14 pm

hashbang is for SEO purpose not like the hack you said
https://developers.google.com/webmasters/ajax-crawling/docs/getting-started

Also using hash url is pretty common in AJAX-driven SPA to maintain the state of the app

Reply

-  *cjramki* says:
November 13, 2013 at 3:05 pm

There is any option to back button tracking.
If the user click on the browser back button, show some confirm box. Is it possible with this?

Reply

Join the discussion.

Name (required)

Mail (will not be published) (required)

Website

Some HTML is ok

You can use these tags:

```
<a href="" title="">
<abbr title="">
<b>
<blockquote cite="">
<cite>
<del datetime="">
<em>
<i>
<q cite="">
<strong>
```

You can also use `<code>`, and **remember to use `&lt;` and `&gt;` for brackets**.

Enter comment

Submit Comment

Notify me of followup comments via e-mail

Element Index

Sponsors

HTML5 Doctor

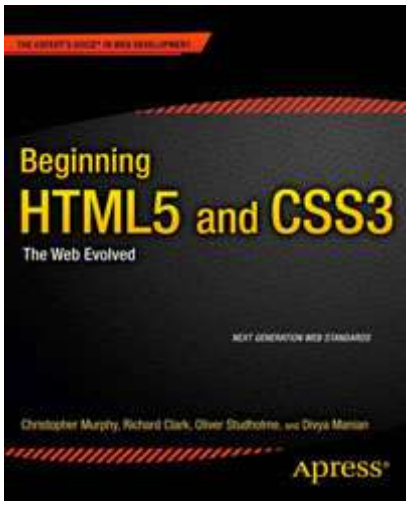
- Articles
- Categories
- Authors

- Computer says NO to HTML5 document outline
- The woes of date input
- HTML Developers: Please Consider
- On HTML belts and ARIA braces (The Default Implicit ARIA semantics they didn't want you to know about)
- HTML5 – Check it Before you Wreck it with Mike[tm] Smith

View article archive

Submit your site to the Gallery

Beginning HTML5 & CSS3



The Web Evolved: A new book on HTML5 & CSS3 by Richard Clark, Oli Studholme, Christopher Murphy and Divya Manian.

Buy it now

Related Posts:

Recent Posts

- Computer says NO to HTML5 document outline
- The woes of date input
- HTML Developers: Please Consider

- On HTML belts and ARIA braces (The Default Implicit ARIA semantics they didn't want you to know about)
- HTML5 – Check it Before you Wreck it with Mike[tm] Smith

Recommended

Buy this book