Web Species blog

As you might know from our website we work with a lot of different technologies, tools and projects.
It only makes sense for us to share our experiences to help others build things faster and better.

HTML5 History API - dynamic websites like never before

by Juozas

I have talked about this before, but JavaScript should not dictate content or website structure. It should only improve the UI, but even with JavaScript disabled website should work. Using the new HTML5 History API allows to do that one step further - making dynamic websites behave like *normal* ones.

What is History API?

History API is quite a simple concept - a JavaScript API you can use to control history state. If user clicks on an image and you show a lightbox with enlarged version, clicking `Back` sends user back to previous page, rather than closing lightbox popup. It does this, because there is no state information browser can use to know how to close that popup window.

With using History API you can add an entry to history stack once some dynamic content is loaded. When user clicks `Back` browser would go back by one element in history stack and fire off an event which then you can handle by closing the popup window. Same thing happens when user clicks `Forward` - event happens and it's up to your script to handle this gracefully.

All this behavior is very well explained in *"Dive into HTML5"* book, but in short when you load some content using Ajax or you move user to a place in a page which you want to have linkable - use History API. Of course this involves handling the links in server side too, but this is quite trivial - build the website first then make use of History API to improve the interface. In my tests this improved user experienced a lot and allowed to achieve very rich user interfaces without destroying website structure.

Of course like with most of the new HTML5 functionality, this is not supported in all browsers. Most importantly this is not supported at all in any IE versions, not even IE9 which was released only couple months ago. But you can handle this by having improved UI for some users and falling back to normal non-js behavior for IE, for example this is what Github does (click on folders).

How to use it?

There is only one main method:

```
history.pushState(state, title, link);
```

and one event:

```
window.addEventListener("popstate", function(e) {
    alert("location: " + document.location + ", state: " + JSON.stringify(event.state));
}
```

State example from Mozzila documentation:

```
history.pushState({page: 1}, "title 1", "?page=1");
history.pushState({page: 2}, "title 2", "?page=2");
history.replaceState({page: 3}, "title 3", "?page=3");
history.back(); // alerts "location: http://example.com/example.html?page=1, state: {"page":1}"
history.back(); // alerts "location: http://example.com/example.html, state: null
history.go(2);  // alerts "location: http://example.com/example.html?page=3, state: {"page":3}
```

However I would recommend using some abstraction for this, mainly because you need to do quite some manual work, which I hate doing. A great tool exists called History.js which does exactly that - abstracts History API and **also** has fall-back support for lame browsers like IE. I've used it extensively and it works great and it even has adapters for jQuery et al so you can use the same interface.

Most impressive part of it is optional fall-back for older browsers. What it does is uses a similar to hashbang url, like `http://webspecies.co.uk/#/about` which it handles inside and all methods for getting current url still return `/about`. And if you go to this url from a modern browser it detects the support for proper History API and redirects to correct url. All combined, makes everything work nicely and future-proof, here is an example of full script for Ajax page.

How I used it

When building our Web Species website designer had an idea to have all content in one scrollable page. If you go to the homepage and start scrolling you'd notice that immediately - menu stays in place but content slides scroll as usual. Now even though this is very nice from user perspective, it creates a problem from content side. **Very big problem.**

Main problem is that there is no way to link people to specific slide and that it's hard to get good rankings on Google. Now even though the first problem can be easily fixed by using anchor urls (like `http://webspecies.co.uk/#about`), but fixing SEO is, I believe, impossible. Impossible because one page contains a lot of content and you won't get good rankings for any keywords.

If you look at view source of `http://webspecies.co.uk/about` you'd see correct title and only "about" content, but as

soon as you load this in browser AJAX loads all slides and replaces existing slide with them. So from user perspective there still exists one-page effect, but actual pages can be called directly (in server side I have all separate pages like about, training, clients etc. and one with all slides).

To fix linking I employed History API and it worked out beautifully. If you click a menu item on left side of the website, AJAX event fires off resulting in two actions - `pushState()` and scroll to correct slide. Same thing happens on `Back` or `forward` history actions - content is scrolled to where it belongs. If you go to news section clicking on news items result in same behaviour too.

Conclusion

As with all HTML5 functionally History API is still quite rarely used, but if you feel like building something awesome - I'd say give it a go. It works beautifully in modern browsers and there are tools which can make it happen for older browsers too. From my point of view, it allows to achieve crazy UI ideas and still have semantically correct websites, which is always my goal.

Author: Juozas *"Joe"* Kaziukėnas, CEO of Web Species Ltd.
Can be reached using twitter as @juokaz or in his website http://juokaz.com.
Published: May 26, 2011 @ 13:47
Filed Under: JavaScript, HTML5
Tags: html, html5, history api, jquery, javascript, webspecies

Hashbangs do solve SEO, but they are broken by design and I can write whole post just discussing why :)

Calvin Froedge

I had wondered what GitHub was using to let me use my back button when looking through code, even though my URL wasn't changing. Now I know. Thanks for the article =)

' Subscribe by email + RSS