Abstract: Currently, Servo has no AJAX support exposed to JavaScript. AJAX is an integral part of the web these days, so the lack of this feature makes Servo much less useful as a browser. The goal of this project is to have a partial implementation of the XMLHttpRequestobject -- one which is self-consistent and covers most use cases that are found today.

Personal Details

- Name: Manish Goregaokar
- Email: something
- IRC nick on irc.mozilla.org: Manishearth
- Telephone: something
- Other contact methods: Google Talk (IRC preferred for short communications)
- Country of residence: India
- Timezone: IST (GMT + 0530)
- Primary language: English

Project Proposal

The final goal is to partially implement the XMLHttpRequest and associated helper objects. The following portions of the protocol will be implemented (full spec here):

- `open()` method: Both sync and async arguments, GET/POST requests. No support for HTTP auth.
- `send()` method: Initially focus on simple GET sending, then move on to `application/x-www-form-urlencoded` and `text/plain` (if possible, `multipart/form-data`). If there is time (not necessarily during the summer), expand to Blobs and other types.
  - Some work for this could be done by expanding rust-http (specifically RequestWriter or a wrapper of the same), which currently has no distinction between header and postdata. There's a pending issue on this.
  - resource_task.rs contains code that handles forming tasks for HTTP requests. It seems to be easy to make async (in a sense it already is). Header support will need to be added, as well as postdata support after the rust-http changes are made.
  - For suporting `application/x-www-form-urlencoded` , we can use form_urlencoded.rs from `rust-url`.
  - For `multipart/form-data`, formdata.rs can be extended so that a FormData object can be loaded and converted to text. The goal is to have something like this work (for strings), since some AJAX on the web is done this way.
  - I will try to implement as much as possible of this separately from the XHR code, this will be useful for forms later. I plan to look into making text based forms (sans UI) work before the summer starts.
- Essentials like `readyState`, `statusText`, `responseText` must be implemented.
- Allow for setting timeouts, custom headers, and fetching of headers from a request.
- Events: `onreadystatechange, onload, onerror, ontimeout, onabort`. Investigate necessity of `onprogress` and implement if necessary.

Additional components of the proposal are:

- A testsuite that covers most of the implemented work, which is easy to extend. Use wptserve or similar for generating responses. This will basically spawn instances of wptserve from the usual content test engine, run tests, and collect results.
- Try to make jQuery's `$.get,$.post,$.ajax` work with the implemented XHR object. Nowadays most of the AJAX on the Web is done by proxy of jQuery. In parallel with the XHR work (or before the summer), I can attempt to create a dummy XHR object in JS, identify, and try to fix any issues with jQuery on Servo.

Schedule of Deliverables

My summer vacation is from May 1 to (approximately) July 21 (as opposed to GSoC's May 19-Aug 18) . This period is slightly shorter (though it starts early), however I can also work during the autumn semester until the official end of GSoC. I will be unable to devote a full 5-7 hours per day for this last part, it should realistically be around 1-2 hours on weekdays and 5-9 hours on weekends.

I also plan to get used to the codebase during this semester, so I doubt that the shift in timings should be an issue.

I've tried to structure the timeline so that there are fewer weeks which are purely for coding or purely for learning. The last 1-2 days of a week will usually be reserved for code review and pull requesting (if necessary), though I intend to keep in touch with the mentor throughout the week and ensure that I'm going in the right direction. I'll try to pull request consistent portions of the project bit by bit, instead of submitting it all at once -- given the frequency of rust upgrades, and the instability of other modules, it's best to keep up with the rest of the code. Since the testing framework won't be in a commitable state until later, I intend to keep a makeshift testcase in the pull requests themselves.

Timeline:

- (Before May 1):
    - Write some fake XHR objects that return dummy data. These will contain only the features that the proposal plans to implement. I can use this with jQuery on Chrome and see if it's enough to make jQuery AJAX work; the results of these tests will let me know if there are additional features that my proposal will need. Then, I can test it with Servo and note down what JS features need to be implemented, and file issues for the same. jQuery is a rather complex library -- the goal here is not to get the entire library to work, but to fix enough of the library so that the basic AJAX methods work.
    - Try to get some experience in the relevant portions of the codebase by partially implementing forms. This is not part of the proposal, but if implemented it would be significantly easier to implement POST requests.
- May 1 - May 4: Discuss project implementation in detail with mentor and knowledgeable community members. Focussed studying of spec (as well as Firefox's implementation), and comparison with implemented code for other objects. Ideally this will be done beforehand, but I'm leaving this time period in here just in case. Additionally, I can learn more about how tasks from `script_task` and `resource_task` work.
- May 4 - May 11: Start coding. Work on prerequisite framework:
    - Tweak `resource_task` by adding an async method.
    - If the rust-http issue has not yet pushed through, work on a simpler fix that adds a simple `send()` to `RequestWriter`. This might take longer than a week, since tests will need to be written and code will need to be reviewed. Since this is not entirely necessary for the first part, it can be done in parallel.
    - Pull request for the above two points (individually)
    - Read wptserve docs and formulate a plan for testing. We'll need to figure out what kinds of tests we want, and how we will structure the tests. (I have already thought up a bit about the structure of the framework, more details below)
- May 11 - May 18: XHR basics
    - Add the necessary webidls and codegen changes for the XMLHTTPRequest-related objects. Write an impl for the object that contains dummy methods (similar to most of the unimplemented html*element.rs files).
    - Implement a basic `open()` that just preps the object without doing anything.
    - Pull request for the above two points
    - Start work on the testing framework. This involves writing modifying the content test runner so that it spawns a dummy wptserve instance which is able to return various responses from the test directory.
- May 18 - May 25: Implement `send()` for GET. Unless `resource_task` makes this problematic, try to

get both sync and async working. Test against localhost (SimpleHTTPServer or Flask as makeshift testers), the testing framework probably won't be ready till now. Support redirects and other similar cases, as well as some basic HTTP errors. `responseText` should work as well. Pull request for the above, with details on the makeshift testing method

- May 25 - June 8 (2 weeks):
  - Handle `onreadystatechange`. This might have to be tested via `addEventListener()` depending on the implementation status of EventHandler (I plan to look into EventHandler during the current semester).
  - Add header / content support to resource_task.rs. Hook up the header support to the XHR code. This might change depending on the status of the pending `RequestWriter` changes (though a makeshift wrapper for RequestWriter can be built for the scope of this project, and swapped out later on)
  - Pull request the above two points (separately)
  - Investigate the full range of possible errors and bad data input, this will be necessary later for testing.
  - Continue work on the testing framework. At this stage, one should be able to write simple tests where the details of server response is stored in something akin to Mochitest's ^headers^ file. While not all types of response details might be implemented in this portion of the summer, the basic framework should be up, and it should be easy to add support for more response details later. There will be certain tests that will require server-side checking of sent data. For these we can keep the test result in the response itself, and check it in the client.
- June 8 - June 15: Add support for POST (`application/x-www-form-urlencoded` and `text/plain`) to `send()`. This will require using the changes to resource_task.rs implemented in the previous week. A new struct, URLSearchParams (similar to FormData), will have to be implemented that uses form_urlencoded.rs for encoding.
- June 15 - June 22:
  - Implement the other event listeners for the XHR object, with error handling. Also implement timeouts.
  - Start writing tests. Add options to the framework when needed.
  - Pull request the above two separately. I'm uncertain if bors will be able to handle the new framework without changes, so for now the tests may be kept independent of `make check` and will have to be run locally by others.
  - Prepare for midterm evaluation. (June 22)
- June 22 - June 29:
  - Continue writing tests. Which tests are necessary will need to be discussed with the community (and gleaned from existing Firefox code), so this could take a while.
  - Make bors work with the framework, if necessary. Hook the tests up with `make check` if we are reasonably sure of their reliability. It's best if we avoid breaking tests for everyone.
  - Implement any remaining necessary attributes from the spec (eg `responseType` if necessary, etc)
  - Pull request above three points separately.
- June 29 - July 6: By now, a usable XHR object should be implemented. In this week, I intend to test it out on existing non-jQuery websites. I will already have done some of the work for this with the dummy XHR object. Fix any issues that occur.
- July 6 - July 13: Work on getting it to work with jQuery. Write some basic jQuery testcases and see what errors are thrown. Hopefully the issues that were filed during the bonding period will have been fixed (by me or someone else), otherwise work on them now. One specific thing that will need to be implemented is the ability to spoof the content-type header. While jQeury appears to send `application/x-www-form-urlencoded` data, internally it uses a urlencoded string passed to the object.
- July 13 - July 20: Wrap up the bulk of the project, pull request any remaining code. Document it. (I intend to try to keep the wiki updated as the project goes on, but it might be better to document it at the end).
- July 20 - Aug 18: Semester will start. This time is intended to be a buffer, in case some of the above spills over. If there is extra time, I can work on the following (if not, I can always look into these after GSoC gets over):
  - `ProgressEvent` support

    - `EventHandler` support if it doesn't already exist. This includes hooking it up to the `on*` attributes of XHR.
    - Other postdata types like `text/html`, `multipart/form-data` (this one might take time since it depends on formdata.rs), and `Blob` (ditto for blob.rs)

Open Source Development Experience

- I've been contributing to firefox for a couple of months (bugzilla profile), mainly in Javascript. I also have recently started mentoring bugs, and in general enjoy helping new users get started.
- I have a couple of commits to servo itself
- I am one of the developers for the Wikipedia Account Request system (repo, commits), though recently I haven't been contributing there much due to time constraints.
- I've helped plan and develop a couple of Stack Exchange-related apps in this organization, in collaboration with two others.
- Quite a bit of my own projects are open source (on github). Some recent ones:
    - AnnoTabe, a tab annotator for Chrome
    - Kapi, a Windows 8 app for fluid mathematical note-taking. This is in collaboration with two other students.
    - A noticeboard for Raspberry Pi with a centralized upload system. This is in collaboration with one other student.
    - Some StackExchange userscripts and a python wrapper for Stack Exchange chat

Work/Internship Experience

As mentioned above, I have some experience with Firefox and servo. I'm been an active Javascript coder for many years now, and while I avoid the "design" portion of "web design", I understand most of how javascript works.

I've not done any programming/CS interns (mostly physics), but I have been contracted for a couple of scripts or website work in the past (eg, recently I wrote one for Mathoverflow). Most of my programming has been done as a volunteer in open source projects.

Academic Experience

I'm currently studying physics. I've always had a passion for both programming and physics. However, while it is possible for me to improve my programming on the side while studying physics (most of my programming was self-taught anyway), I don't think it's possible for me to keep in touch with physics when my courses will be programming courses. Physics is something that is significantly harder to teach oneself in my opinion. Of course, I'll never reach the programming/CS level of a student who has that as their primary topic, but I guess some compromise has to be made if I want to enjoy both worlds.

Why Mozilla

I've already been participating in the Mozilla community for a while, mainly with small Firefox features. Compared to the other online communities I've participated in, Mozilla is the most welcoming, and I enjoy being a part of it. Till now I haven't had a chance to do anything major here, and I feel that GSoC is a great way for me to do a major project. Additionally, Servo in particular is very much a work in progress, and I feel that I can have a better impact by implementing core features in a budding browser over enhancing an existing application.