jQuery — New Wave JavaScript

Contribution Guides

In the spirit of open source software development, jQuery always encourages community code contribution. To help you get started and before you jump into writing code, be sure to read these important contribution guidelines thoroughly:

1. Getting Involved
2. Core Style Guide
3. Writing Code for jQuery Foundation Projects

References to issues/PRs

GitHub issues/PRs are usually referenced via `gh-NUMBER`, where `NUMBER` is the numerical ID of the issue/PR. You can find such an issue/PR under `https://github.com/jquery/jquery/issues/NUMBER`.

jQuery has used a different bug tracker - based on Trac - in the past, available under bugs.jquery.com. It is being kept in read only mode so that referring to past discussions is possible. When jQuery source references one of those issues, it uses the pattern `trac-NUMBER`, where `NUMBER` is the numerical ID of the issue. You can find such an issue under `https://bugs.jquery.com/ticket/NUMBER`.

Environments in which to use jQuery

- Browser support
- jQuery also supports Node, browser extensions, and other non-browser environments.

What you need to build your own jQuery

To build jQuery, you need to have the latest Node.js/npm and git 1.7 or later. Earlier versions might work, but are not supported.

For Windows, you have to download and install git and Node.js.

macOS users should install Homebrew. Once Homebrew is installed, run `brew install git` to install git, and `brew install node` to install Node.js.

Linux/BSD users should use their appropriate package managers to install git and Node.js, or build from source if you swing that way. Easy-peasy.

How to build your own jQuery

First, clone the jQuery git repo.

Then, enter the jquery directory and run the build script:

```
cd jquery && npm run build
```

The built version of jQuery will be put in the `dist/` subdirectory, along with the minified copy and associated map file.

If you want to create custom build or help with jQuery development, it would be better to install grunt command line interface as a global package:

```
npm install -g grunt-cli
```

Make sure you have `grunt` installed by testing:

```
grunt -V
```

Now by running the `grunt` command, in the jquery directory, you can build a full version of jQuery, just like with an `npm run build` command:

```
grunt
```

There are many other tasks available for jQuery Core:

```
grunt -help
```

Modules

Special builds can be created that exclude subsets of jQuery functionality. This allows for smaller custom builds when the builder is certain that those parts of jQuery are not being used. For example, an app that only used JSONP for `$.ajax()` and did not need to calculate offsets or positions of elements could exclude the offset and ajax/xhr modules.

Any module may be excluded except for `core`, and `selector`. To exclude a module, pass its path relative to the `src` folder (without the `.js` extension).

Some example modules that can be excluded are:

- **ajax**: All AJAX functionality: `$.ajax()`, `$.get()`, `$.post()`, `$.ajaxSetup()`, `.load()`, transports, and ajax event shorthands such as `.ajaxStart()`.
- **ajax/xhr**: The XMLHTTPRequest AJAX transport only.
- **ajax/script**: The `<script>` AJAX transport only; used to retrieve scripts.
- **ajax/jsonp**: The JSONP AJAX transport only; depends on the ajax/script transport.
- **css**: The `.css()` method. Also removes **all** modules depending on css (including **effects**, **dimensions**, and **offset**).
- **css/showHide**: Non-animated `.show()`, `.hide()` and `.toggle()`; can be excluded if you use classes or explicit `.css()` calls to set the `display` property. Also removes the **effects** module.
- **deprecated**: Methods documented as deprecated but not yet removed.
- **dimensions**: The `.width()` and `.height()` methods, including `inner-` and `outer-` variations.
- **effects**: The `.animate()` method and its shorthands such as `.slideUp()` or `.hide("slow")`.
- **event**: The `.on()` and `.off()` methods and all event functionality.
- **event/trigger**: The `.trigger()` and `.triggerHandler()` methods.

- **offset**: The `.offset()`, `.position()`, `.offsetParent()`, `.scrollLeft()`, and `.scrollTop()` methods.
- **wrap**: The `.wrap()`, `.wrapAll()`, `.wrapInner()`, and `.unwrap()` methods.
- **core/ready**: Exclude the ready module if you place your scripts at the end of the body. Any ready callbacks bound with `jQuery()` will simply be called immediately. However, `jQuery(document).ready()` will not be a function and `.on("ready", ...)` or similar will not be triggered.
- **deferred**: Exclude jQuery.Deferred. This also removes jQuery.Callbacks. *Note* that modules that depend on jQuery.Deferred(AJAX, effects, core/ready) will not be removed and will still expect jQuery.Deferred to be there. Include your own jQuery.Deferred implementation or exclude those modules as well (`grunt custom:-deferred,-ajax,-effects,-core/ready`).
- **exports/global**: Exclude the attachment of global jQuery variables ($ and jQuery) to the window.
- **exports/amd**: Exclude the AMD definition.

The build process shows a message for each dependent module it excludes or includes.

AMD name

As an option, you can set the module name for jQuery's AMD definition. By default, it is set to "jquery", which plays nicely with plugins and third-party libraries, but there may be cases where you'd like to change this. Simply set the `"amd"` option:

```
grunt custom --amd="custom-name"
```

Or, to define anonymously, set the name to an empty string.

```
grunt custom --amd=""
```

Custom Build Examples

To create a custom build, first check out the version:

```
git pull; git checkout VERSION
```

Where VERSION is the version you want to customize. Then, make sure all Node dependencies are installed:

```
npm install
```

Create the custom build using the `grunt custom` option, listing the modules to be excluded.

Exclude all **ajax** functionality:

```
grunt custom:-ajax
```

Excluding **css** removes modules depending on CSS: **effects**, **offset**, **dimensions**.

```
grunt custom:-css
```

Exclude a bunch of modules:

```
grunt custom:-ajax/jsonp,-css,-deprecated,-dimensions,-effects,-offset,-wrap
```

There is also a special alias to generate a build with the same configuration as the official jQuery Slim build is generated:

```
grunt custom:slim
```

For questions or requests regarding custom builds, please start a thread on the Developing jQuery Core section of the forum. Due to the combinatorics and custom nature of these builds, they are not regularly tested in jQuery's unit test process.

Running the Unit Tests

Make sure you have the necessary dependencies:

```
npm install
```

Start `grunt watch` or `npm start` to auto-build jQuery as you work:

```
grunt watch
```

Run the unit tests with a local server that supports PHP. Ensure that you run the site from the root directory, not the "test" directory. No database is required. Pre-configured php local servers are available for Windows and Mac. Here are some options:

- Windows: WAMP download
- Mac: MAMP download
- Linux: Setting up LAMP
- Mongoose (most platforms)

Building to a different directory

To copy the built jQuery files from `/dist` to another directory:

```
grunt && grunt dist:/path/to/special/location/
```

With this example, the output files would be:

```
/path/to/special/location/jquery.js
/path/to/special/location/jquery.min.js
```

To add a permanent copy destination, create a file in `dist/` called ".destination.json". Inside the file, paste and customize the following:

```
{
  "/Absolute/path/to/other/destination": true
}
```

Additionally, both methods can be combined.

Essential Git

As the source code is handled by the Git version control system, it's useful to know some features used.

Cleaning

If you want to purge your working directory back to the status of upstream, the following commands can be used (remember everything you've worked on is gone after these):

```
git reset --hard upstream/main
git clean -fdx
```

Rebasing

For feature/topic branches, you should always use the `--rebase` flag to `git pull`, or if you are usually handling many temporary "to be in a github pull request" branches, run the following to automate this:

```
git config branch.autosetuprebase local
```

(see `man git-config` for more information)

Handling merge conflicts

If you're getting merge conflicts when merging, instead of editing the conflicted files manually, you can use the feature `git mergetool`. Even though the default tool `xxdiff` looks awful/old, it's rather useful.

The following are some commands that can be used there:

- `Ctrl + Alt + M` - automerge as much as possible
- `b` - jump to next merge conflict
- `s` - change the order of the conflicted lines
- `u` - undo a merge
- `left mouse button` - mark a block to be the winner
- `middle mouse button` - mark a line to be the winner
- `Ctrl + S` - save
- `Ctrl + Q` - quit

QUnit Reference

Test methods

```
expect( numAssertions );
stop();
start();
```

Note: QUnit's eventual addition of an argument to stop/start is ignored in this test suite so that start and stop can be passed as callbacks without worrying about their parameters.

Test assertions

```
ok( value, [message] );
equal( actual, expected, [message] );
notEqual( actual, expected, [message] );
deepEqual( actual, expected, [message] );
notDeepEqual( actual, expected, [message] );
strictEqual( actual, expected, [message] );
notStrictEqual( actual, expected, [message] );
throws( block, [expected], [message] );
```

Test Suite Convenience Methods Reference (See test/data/testinit.js)

Returns an array of elements with the given IDs

```
q( ... );
```

Example:

```
q("main", "foo", "bar");

=> [ div#main, span#foo, input#bar ]
```

Asserts that a selection matches the given IDs

```
t( testName, selector, [ "array", "of", "ids" ] );
```

Example:

```
t("Check for something", "//[a]", ["foo", "bar"]);
```

Fires a native DOM event without going through jQuery

```
fireNative( node, eventType )
```

Example:

```
fireNative( jQuery("#elem")[0], "click" );
```

Add random number to url to stop caching

```
url( "some/url" );
```

Example:

```
url("index.html");

=> "data/index.html?10538358428943"


url("mock.php?foo=bar");

=> "data/mock.php?foo=bar&10538358345554"
```

Run tests in an iframe

Some tests may require a document other than the standard test fixture, and these can be run in a separate iframe. The actual test code and assertions remain in jQuery's main test files; only the minimal test fixture markup and setup code should be placed in the iframe file.

```
testIframe( testName, fileName,
  function testCallback(
      assert, jQuery, window, document,
      [ additional args ] ) {
    ...
  } );
```

This loads a page, constructing a url with fileName `"./data/" + fileName`. The iframed page determines when the callback occurs in the test by including the "/test/data/iframeTest.js" script and calling `startIframeTest( [ additional args ] )` when appropriate. Often this will be after either document ready or `window.onload` fires.

The `testCallback` receives the QUnit `assert` object created by `testIframe` for this test, followed by the global `jQuery`, `window`, and `document` from the iframe. If the iframe code passes any arguments to `startIframeTest`, they follow the `document` argument.

Questions?

If you have any questions, please feel free to ask on the Developing jQuery Core forum or in #jquery on libera.