# Contributing to RxJS #

Want to contribute to the Reactive Extensions for JavaScript (RxJS)? There are many ways of helping whether contributing code, documentation, examples, podcasts, videos and presentations.

# Get Involved!

In [the issue tracker](https://github.com/Reactive-Extensions/RxJS/issues), bugs can only be assigned to people who have commit access. Also, we aspire to make as many bugs as possible "owned" by assigning them to a core Rx contributor. Therefore, just because a bug is assigned doesn't mean it's being actively worked on. We (the core contributors) are all busy, and welcome help from the community. If you see a bug you'd like to work on that's assigned but appears to be dormant, communicate with the bug's owner with an @-reply in a comment on the issue page. If you see a bug you'd like to work on that's unassigned, it's fair game: comment to say you'd like to work on it so that we know it's getting attention.

# Pull Requests

To make a pull request, you will need a GitHub account; if you're unclear on this process, see GitHub's documentation on [forking](https://help.github.com/articles/fork-a-repo/) and [pull requests](https://help.github.com/articles/using-pull-requests). Pull requests should be targeted at RxJS's master branch. Before pushing to your Github repo and issuing the pull request, please do two things:

1. Rebase your local changes against the master branch. Resolve any conflicts that arise.

2. Run the full RxJS test suite by running `grunt` in the root of the repository.

Pull requests will be treated as "review requests", and we will give feedback we expect to see corrected on style and substance before pulling. Changes contributed via pull request should focus on a single issue at a time, like any other. We will not look kindly on pull-requests that try to "sneak" unrelated changes in. Note for bug fixes, regression tests should be included, denoted by Issue Number so that we have full traceability.

# What Are We Looking For?

For documentation, we are looking for the following:

- API Documentation that is missing or out of date

- "How Do I?" examples

- Comparison to other libraries

- Comparison to Promises

- Introduction material

- Tutorials

For coding, we have strict standards that must be adhere to when working on RxJS. In order for us to accept pull requests, they must abide by the following:

- [Coding Standard](#coding-standard)

- [Tests](#tests)

- [Documentation](#documentation)

## Coding Standard

For RxJS, we follow the [Google JavaScript Style Guide](http://google.github.io/styleguide/jsguide.html) and adhere to it strictly in areas such as documentation using JSDoc. The only exception to extending native prototypes is to polyfill behavior which may not exist in all browsers yet, for example, many of the [Array#extras](http://blogs.msdn.com/b/ie/archive/2010/12/13/ecmascript-5-part-2-array-extras.aspx) are implemented in compatibility builds. We also strictly follow [our design guidelines](https://github.com/Reactive-Extensions/RxJS/tree/master/doc/designguidelines) as well.

### Supporting Multiple Platforms

RxJS runs on a number of platforms and supports many styles of programming. RxJS supports [Universal Module Definition (UMD)](https://github.com/umdjs/umd) which allows the library to work in a number of environments such as [Asynchronous Module Definition (AMD)](https://github.com/amdjs/amdjs-api/wiki/AMD), [CommonJS](http://wiki.commonjs.org/wiki/CommonJS), [Node.js](http://nodejs.org), [RingoJS](http://ringojs.org/), [Narwhal](https://github.com/280north/narwhal), the browser and other environments such as [Windows Script Host (WSH)](http://msdn.microsoft.com/en-us/library/9bbdkx3k.aspx) and embedded devices such as [Tessel](http://tessel.io).

RxJS is committed to using the latest JavaScript standards as they start to arrive, for example, supporting generators, Maps, Sets, and Observable versions of new Array methods. We also are committed to supporting legacy code as well through compatibility builds, even supporting browsers back to IE6, Firefox 3, and older versions of Node.js. Should behavior not exist in those platforms, that behavior must be polyfilled, and made available in `\*.compat.js` files only. For example, we have `rx.lite.js` which supports modern browsers greater than or equal to IE9, and `rx.lite.compat.js` for older browsers before IE9 and modern Firefox builds. In special cases such as event handling is different, we must provide a mainstream version of the file as well as a compat file, the latter which is included in the compat file.

### Implementing Custom Operators

We welcome custom operators to RxJS if they make sense in the core RxJS, as opposed to belonging in user land. There are a number of rules that must be adhered to when implementing a custom operator including:

- Prefer composition over implementing a totally new operator from scratch

- If the operator introduces any notion of concurrency, then a scheduler must introduced. Usage of concurrency primitives such as `setTimeout`, `setInterval`, etc are forbidden. This is to ensure easy testability.

- The scheduler must be optional with the appropriate default picked

- `Rx.Scheduler.immediate` for any immediate blocking operations

- `Rx.Scheduler.currentThread` for any immediate blocking operators that require re-entrant behavior such as recursive scheduling.

- `Rx.Scheduler.timeout` for any operator that has a notion of time

To make this concrete, let's implement a custom operator such as an implementation of `\_.reject` from [Underscore.js](http://underscorejs.org/) / [Lo-Dash](http://lodash.com/).

```js

/\*\*

\* The opposite of \_.filter this method returns the elements of a collection that the callback does \*\*not\*\* return truthy for.

\* @param {Function} [callback] The function called per iteration.

\* @param {Any} [thisArg] The this binding of callback.

\* @returns {Observable} An Observable sequence which contains items that the callback does not return truthy for.

\*/

Rx.Observable.prototype.reject = function (callback, thisArg) {

callback || (callback = Rx.helpers.identity);

var source = this;

return new Rx.AnonymousObservable(function (observer) {

var i = 0;

return source.subscribe(

function (x) {

var noYield = true;

try {

noYield = callback.call(thisArg, x, i++, source);

} catch (e) {

observer.onError(e);

return;

}

if (!noYield) { observer.onNext(x); }

},

observer.onError.bind(observer),

observer.onCompleted.bind(observer)

);

});

};

```

Of course, we could have implemented this using composition as well, such as using `Rx.Observable.prototype.filter`.

```js

/\*\*

\* The opposite of \_.filter this method returns the elements of a collection that the callback does \*\*not\*\* return truthy for.

\* @param {Function} [callback] The function called per iteration.

\* @param {Any} [thisArg] The this binding of callback.

\* @returns {Observable} An Observable sequence which contains items that the callback does not return truthy for.

\*/

Rx.Observable.prototype.reject = function (callback, thisArg) {

callback || (callback = Rx.helpers.identity);

return this.filter(function (x, i, o) { return !callback.call(thisArg, x, i o); });

};

```

To show an operator that introduces a level of concurrency, let's implement a custom operator such as an implementation of `\_.pairs` from [Underscore.js](http://underscorejs.org/) / [Lo-Dash](http://lodash.com/). Note that since this requires recursion to implement properly, we'll use the `Rx.Scheduler.currentThread` scheduler.

```js

var keysFunction = Object.keys || someKeysPolyfill;

/\*\*

\* Creates an Observable with an of an object’s key-value pairs.

\* @param {Object} obj The object to inspect.

\* @returns {Observable} An Observable with an of an object’s key-value pairs.

\*/

Rx.Observable.pairs = function (obj, scheduler) {

scheduler || (scheduler = Rx.Scheduler.currentThread);

return new Rx.AnonymousObservable(function (observer) {

var keys = keysFunction(object),

i = 0,

len = keys.length;

return scheduler.scheduleRecursive(function (self) {

if (i < len) {

var key = keys[i++], value = obj[key];

observer.onNext([key, value]);

self();

} else {

observer.onCompleted();

}

});

});

};

```

Note that all operators must have the documentation and must be split out into its own file. This allows us to be able to put it in different files, or make it available in custom builds.

## Tests

When a new operator is written for RxJS, in order to accepted, must be accompanied by tests. RxJS currently uses [QUnit](http://qunitjs.com/) as a straight forward way to test our code. These tests are automatically executed by our [Grunt](http://gruntjs.com/) setup to concatenate files, minimize, create source maps, and finally run all the tests in the [tests folder](https://github.com/Reactive-Extensions/RxJS/tree/master/tests). Each file that we produce, for example, `rx.js` has an accompanying test file such as `rx.html`, which includes tests for all operators included in that file.

Each operator under test must be in its own file to cover the following cases:

- Never

- Empty

- Single/Multiple Values

- Error in the sequence

- Never ending sequences

- Early disposal in sequences

If the operator has a callback, then it must cover the following cases:

- Success with all values in the callback

- Success with the context, if any allowed in the operator signature

- If an error is thrown

To get a good feeling on what kind of rigor is required for testing, check out the following examples:

- [`concatMap`](https://github.com/Reactive-Extensions/RxJS/blob/master/tests/observable/concatmap.js)

- [`from`](https://github.com/Reactive-Extensions/RxJS/blob/master/tests/observable/from.js)

## Documentation

Documentation is also a must, as all external operators and types must be documented and put in the [API Folder](https://github.com/Reactive-Extensions/RxJS/tree/master/doc/api). Each operator on an Observable must have its own file in the [Operators Folder](https://github.com/Reactive-Extensions/RxJS/tree/master/doc/api/core/operators).

For operators, they must be linked from the [`Observable`](https://github.com/Reactive-Extensions/RxJS/blob/master/doc/api/core/observable.md) API document. In addition, each operator must be listed in which file it belongs in the [Libraries Folder](https://github.com/Reactive-Extensions/RxJS/tree/master/doc/libraries).

The standard format of operators must be such as the [`of`](https://github.com/Reactive-Extensions/RxJS/blob/master/doc/api/core/operators/of.md) operator which includes:

- File Location

- Method signature

- Method description

- List of Arguments

- Return type (if there is one)

- An example

- File Distribution(s)

- NuGet Distribution

- NPM Distribution

- Unit Tests