M O U S E T R A P

Fork me on GitHub

A simple library for handling keyboard shortcuts in Javascript.
Try pressing some of the keys here:

```
// single keys
Mousetrap.bind('4', function() { highlight(2); });
Mousetrap.bind('x', function() { highlight(3); }, 'keyup');

// combinations
Mousetrap.bind('command+shift+k', function(e) {
    highlight([6, 7, 8, 9]);
    return false;
});

Mousetrap.bind(['command+k', 'ctrl+k'], function(e) {
    highlight([11, 12, 13, 14]);
    return false;
});

// gmail style sequences
Mousetrap.bind('g i', function() { highlight(17); });
Mousetrap.bind('* a', function() { highlight(18); });

// konami code!
```

```
Mousetrap.bind('up up down down left right left right b a enter', function() {
    highlight([21, 22, 23]);
});
```

INTRODUCTION

Mousetrap is a standalone library with no external dependencies. It weighs in at around 1.7kb minified and gzipped and 3kb minified.

What are you waiting for? Throw away your mouse and download it now.

If you like this and want to donate to help support development Gittip

BROWSER SUPPORT

Mousetrap has been tested and should work in

- Internet Explorer 6+
- Safari
- Firefox
- Chrome

SUPPORTED KEYS

For modifier keys you can use `shift`, `ctrl`, `alt`, `option`, `meta`, and `command`.

Other special keys are `backspace`, `tab`, `enter`, `return`, `capslock`, `esc`, `escape`, `space`, `pageup`, `pagedown`, `end`, `home`, `left`, `up`, `right`, `down`, `ins`, and `del`.

Any other key you should be able to reference by name like `a`, `/`, `$`, `*`, or `=`.

API REFERENCE

I. Mousetrap.bind

The bind method is the main call you will be making. This will bind a specified keyboard command to a callback method.

Single key

```
Mousetrap.bind('/', _focusSearch);
```

There is a third argument you can use to specify the type of event to listen for. It can be `keypress`, `keydown` or `keyup`.

It is recommended that you leave this argument out if you are unsure. Mousetrap will look at the keys you are binding and determine whether it should default to `keypress` or `keydown`.

Combination of keys

```
Mousetrap.bind('ctrl+s', function(e) {
    _saveDraft();
});
```

If you want to bind multiple key commands to the same callback you can pass in an array for the first argument:

```
Mousetrap.bind(['ctrl+s', 'command+s'], function(e) {
    _saveDraft();
});
```

Note that modifier keys are not explicitly tracked but rather are referenced using `e.shiftKey`, `e.metaKey`, `e.ctrlKey`, and `e.altKey`.

This is more reliable than tracking because if you are holding one of the modifier keys down when you focus the window the browser won't get a keydown event for that key.

Sequence of keys

```
Mousetrap.bind('* a', _selectAll, 'keydown');
```

This feature was inspired by Gmail. Any keys separated by a space will be considered a sequence. If you type each key in order the final one in the sequence will trigger the callback. If you type a key not in the sequence or wait too long the sequence will reset.

You can also make a sequence that includes key combinations within it.

```
Mousetrap.bind('g o command+enter', function() { /* do something */ });
```

Any key events that would normally fire for keys within a sequence will not fire if they are pressed within the context of that sequence.

For example if you have a keydown listener for the `o` key and you press `o` as part of the sequence above, the event for `o` on its own will not fire. As soon as the sequence is broken it will fire again.

It is important to note that Mousetrap can get very confused if you have a single key handler that uses the same key that a sequence starts with. This is because it can't tell if you are starting the sequence or if you are pressing that key on its own.

Shift key

```
Mousetrap.bind('?', function() { alert('keyboard shortcuts'); });
```

Keys that require `shift` are handled magically for you. They should just work. With `keypress` events they will try to match the character and for `keyup` and `keydown` there is a mapping to allow them to work.

Note that `keypress` is the most reliable for non US keyboards

Text fields

By default all keyboard events will not fire if you are inside of a `textarea` , `input` , or `select` to prevent undesirable things from happening.

If for whatever reason you want them to you can add the class `mousetrap` to the element.

```
<textarea name="message" class="mousetrap"></textarea>
```

You can also customize this functionality by overwriting `Mousetrap.stopCallback` . See below

Overwriting a specific event

If you bind the same key event later on in your script it should overwrite the original callback you had specified.

Checking keyboard combination from callback

As of version `1.2` the callback will be passed a second argument with a string of the keyboard combination that triggered the event.

```
Mousetrap.bind('ctrl+shift+up', function(e, combo) {
    console.log(combo); // logs 'ctrl+shift+up'
});
```

This example is a bit contrived because you obviously know the shortcut. If, however, you want to do something like use the same callback function for a bunch of combinations and then check which one triggered the event this allows you to do that

Stopping the default behavior

This is not usually a good practice, but sometimes you may want to overwrite the default behavior of a keyboard combination in the browser.

For example let’s say you want to focus a form input without typing that key into it or you have a text input that you want to save when the user presses `ctrl+s` . You have a couple ways you can achieve this.

You can explicitly prevent the default behavior:

```
Mousetrap.bind(['ctrl+s', 'meta+s'], function(e) {
    if (e.preventDefault) {
        e.preventDefault();
    } else {
        // internet explorer
        e.returnValue = false;
    }
    _saveDraft();
});
```

You can see here that the callback function gets passed the original key event that triggered it.

As a convenience you can also return false in your callback:

```
Mousetrap.bind(['ctrl+s', 'meta+s'], function(e) {
    _saveDraft();
    return false;
});
```

Returning `false` here works the same way as jQuery's `return false`. It prevents the default action and stops the event from bubbling up.

II. Mousetrap.unbind

```
Mousetrap.unbind('?');
```

This method will unbind a single keyboard event or an array of keyboard events. You should pass in the key combination exactly as it was passed in originally to `bind`.

III. Mousetrap.trigger

```
Mousetrap.trigger('esc');
```

Any keyboard event that has been bound can be triggered by passing in the string you used when you bound it originally.

Note that this is not actually triggering a key event in the browser. It is simply firing the event you bound to that key within mousetrap

This method also accepts an optional argument for what type of event to trigger

IV. Mousetrap.stopCallback

Mousetrap.stopCallback is a method that is called to see if the keyboard event should fire based on if you are inside a form input field.

It gets passed three arguments:

- e - key event that triggered this callback
- element - reference to the target element for the event
- combo - the specific keyboard combination that is triggering the event

If true is returned then it stops the callback from firing, if false it fires it.

The default implementation is:

```
stopCallback: function(e, element, combo) {
    // if the element has the class "mousetrap" then no need to stop
    if ((' ' + element.className + ' ').indexOf(' mousetrap ') > -1) {
        return false;
    }

    // stop for input, select, and textarea
    return element.tagName == 'INPUT' || element.tagName == 'SELECT' || element.
}
```

V. Mousetrap.reset

```
Mousetrap.reset();
```

The reset method will remove anything you have bound to mousetrap. This can be useful if you want to change contexts in your application without refreshing the page in your browser. You can ajax in a new page, call reset, and then bind the key events needed for that page.

Internally mousetrap keeps an associative array of all the events to listen for so reset does not actually remove or add event listeners on the document. It just sets the array to be empty.

EXTENDING MOUSETRAP

Any of the public methods can be overwritten to extend Mousetrap.

A few example extensions are:

Bind dictionary

View or Download

This extension overwrites the default bind behavior and allows you to bind multiple combinations in a single bind call.

Usage looks like:

```
Mousetrap.bind({
    'a': function() { console.log('a'); },
    'b': function() { console.log('b'); }
});
```

You can optionally pass in `keypress`, `keydown` or `keyup` as a second argument.

Other bind calls work the same way as they do by default.

Pause/unpause

View or Download

This extension allows Mousetrap to be paused and unpaused without having to reset keyboard shortcuts and rebind them.

Usage looks like:

```
// stop Mousetrap events from firing
Mousetrap.pause();
```

```
// allow Mousetrap events to fire again
```

```
Mousetrap.unpause();
```

Global bindings

View or Download

This extension allows you to specify keyboard events that will work anywhere including inside textarea/input fields.

Usage looks like:

```
Mousetrap.bindGlobal('ctrl+s', function() {
    _save();
});
```

This means that a keyboard event bound using `Mousetrap.bind` will only work outside of form input fields, but using `Moustrap.bindGlobal` will work in both places.

If you wanted to create keyboard shortcuts that only work when you are inside a specifix textarea you can do that too by creating your own extension.

Using extensions

To use any of these extensions all you have to do is include the javascript on your page after you include Mousetrap.

SUPPORT AND BUGS

If you are having trouble, have found a bug, or want to contribute don't be shy.
Open a ticket on Github.

created by Craig Campbell in 2012