Cut and Copy Commands

By Matt Gaunt

Matt is a contributor to Web**Fundamentals**

IE10 and above added support for the 'cut' and 'copy' commands through the Document.execCommand() method. As of Chrome version 43, these commands are also supported in Chrome.

Any text selected in the browser when one of these commands is executed will be cut or copied to the user's clipboard. This lets you offer the user a simple way to select a portion of text and copy it to the clipboard.

This becomes extremely useful when you combine it with the Selection API to programmatically select text to determine what is copied to the clipboard, which we'll be looking at in more detail later on in this article.

Simple Example

For example's sake, let's add a button which copies an email address to the user's clipboard.

We add the email address in our HTML with a button to initiate the copying when it's clicked:

```
<p>Email me at <a class="js-emaillink" href="mailto:matt@example.co.uk">mat

<p><button class="js-emailcopybtn"><img src="./images/copy-icon.png" /></button><
```

Then in our JavaScript, we want to add a click event handler to our button in which we select the email address text from the **`js-emaillink`** anchor, execute a copy command so that the email address is in the user's clipboard and then we deselect the email address so the user doesn't see the selection occur.

```
var copyEmailBtn = document.querySelector('.js-emailcopybtn');
copyEmailBtn.addEventListener('click', function(event) {
  // Select the email link anchor text
  var emailLink = document.querySelector('.js-emaillink');
  var range = document.createRange();
  range.selectNode(emailLink);
  window.getSelection().addRange(range);
  try {
    // Now that we've selected the anchor text, execute the copy command
    var successful = document.execCommand('copy');
    var msg = successful ? 'successful' : 'unsuccessful';
    console.log('Copy email command was ' + msg);
  } catch(err) {
    console.log('Oops, unable to copy');
  }

  // Remove the selections - NOTE: Should use
  // removeRange(range) when it is supported
  window.getSelection().removeAllRanges();
});
```

What we are doing here is using a method of the Selection API, window.getSelection() to programmatically set the 'selection' of text to the anchor, which is the text we want to copy to the user's clipboard. After calling document.execCommand() we can remove the selection by calling window.getSelection().removeAllRanges().
If you wanted to confirm everything worked as expected you can examine the response of document.execCommand(); it returns false if the command is not supported or enabled. We wrap execCommand() in a try and catch since the 'cut' and 'copy' commands can throw an error in a few scenarios.

The 'cut' command can be used for text fields where you want to remove the text content and make it accessible via the clipboard.

Using a textarea and a button in our HTML:

```
<p><textarea class="js-cuttextarea">Hello I'm some text</textarea></p>

<p><button class="js-textareacutbtn" disable>Cut Textarea</button></p>
```

We can do the following to cut the content:

```
var cutTextareaBtn = document.querySelector('.js-textareacutbtn');

cutTextareaBtn.addEventListener('click', function(event) {
  var cutTextarea = document.querySelector('.js-cuttextarea');
  cutTextarea.select();

  try {
    var successful = document.execCommand('cut');
    var msg = successful ? 'successful' : 'unsuccessful';
    console.log('Cutting text command was ' + msg);
  } catch(err) {
    console.log('Oops, unable to cut');
  }
});
```

queryCommandSupported and queryCommandEnabled

Ahead of calling document.execCommand(), you should ensure that this API is supported using the document.queryCommandSupported() method. In our example above we could set the button disabled state based on support like so:

```
copyEmailBtn.disabled = !document.queryCommandSupported('copy');
```

The difference between document.queryCommandSupported() and document.queryCommandEnabled() is that cut and copy could be supported by a browser, but if no text is currently selected, they won't be enabled. This is useful in scenarios where you aren't setting the selection of text programmatically and want to ensure the command will do as expected, otherwise present a message to the user.

Browser Support

IE 10+, Chrome 43+, Firefox 41+, and Opera 29+ support these commands.

Safari does not support these commands.

Known Bugs

- ~~Calling queryCommandSupported() for cut or copy always returns false until after a user interaction. This prevents you from disabling your UI for browsers which don't actually support it.~~ Fixed on Chrome 48.
- Calling queryCommandSupported() from devtools will always return false.
- At the moment cut only works when you programmatically select text.

Last updated July 2, 2018.