THE DEFINITIVE GUIDE TO COPYING AND PASTING IN JAVASCRIPT

At Lucid Software, we strive to build desktop quality software that runs in the browser. Until recently, however, we had settled for using our own internal clipboard implementation rather than the system clipboard — limiting copying and pasting with external applications to just plain text. With customers clamoring for the ability to paste images captured from screenshots or copied from other browser windows, we decided to use the system clipboard exclusively, thus opening the doors to rich content addition.

Along the way, we encountered three major challenges: limited clipboard access, inconsistent events across browsers, and ~~Internet Explorer~~ limited support of multiple data types. This blog post will discuss what we learned and how you can put our insights to use. Keep reading to see code snippets and in-depth answers to these problems. If you have questions, just let us know. We'll be happy to add more details.

Step #1 — Access Clipboard Events on Any Browser

There are several security issues with letting a web page access the system clipboard. Because of this, browsers limit access to the clipboard. In general, you can only access the clipboard during a *system* cut, copy, or paste event. These are fired when a user presses the keyboard shortcuts or uses the browser's menu. This is limiting for two reasons:

1. Browsers (with the exception of Chrome) only fire clipboard events when there is a valid selection and focus on HTML elements. To see this for yourself, use Chrome to play with the following code.

   ```
   ['cut', 'copy', 'paste'].forEach(function(event) {
       document.addEventListener(event, function(e) {
           console.log(event);
       });
   });
   ```

   Now try it in another browser. Nothing. The events don't fire. Now try adding a text input field. If your text cursor is in the input field and you have some text on the clipboard, it pastes just fine. To get copy or cut to fire, you'll need to have some text selected in the input area.

   In Lucidchart, we don't use HTML elements to handle or render our text or shapes. To access the clipboard, we need a clipboard event. To get a clipboard event, we need the focused HTML elements the browser expects. But we don't use those HTML elements.

2. We would like to support copy and paste from our context menu when a user right-clicks, but since we use our own context menu and not the browser's, the system clipboard event isn't fired. And that means we don't get access to the system clipboard.

Consistently Getting Clipboard Events

Our solution looks something like the following code:

```
var hiddenInput = $('#hidden-input');
```

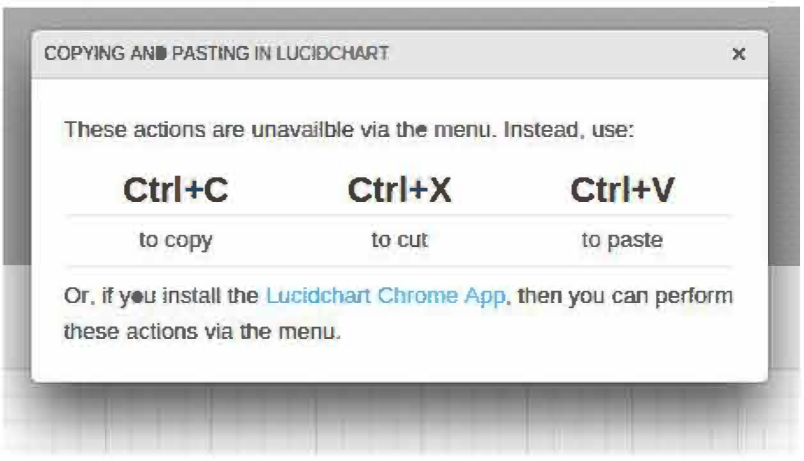
```
var focusHiddenArea = function() {
    hiddenInput.val(' ');
    hiddenInput.focus().select();
};

$(document).mouseup(focusHiddenArea);

['cut', 'copy', 'paste'].forEach(function(event) {
    document.addEventListener(event, function(e) {
        console.log(event);
        focusHiddenArea();
        e.preventDefault();
    });
});
```

We have a hidden text area that is always set to have some text selected. This way, the cut, copy, and paste events are always fired in any browser. However, you'll notice that this Fiddle doesn't do anything when the user types—and after the user types, the clipboard events won't fire until the hidden text area is refocused by clicking. In the case of an application like Lucidchart, this is an insufficient solution. We can't just throw away user keyboard input.

I won't go into all of the details with handing our input, because it turns out that implementing your own text editing is a rather detailed problem. But just to give you an idea, this fiddle is able to successfully capture input and remain ready for a cut, copy, or paste event.

Context Menu Copying and Pasting

So far, we only support context menu copying and pasting for Chrome and IE. Users of other browsers are limited to keyboard shortcuts. Chrome can grant us clipboard access via our Chrome extension, and Internet Explorer lets you access the clipboard any time (via `window.clipboardData`), but will prompt the user if it is outside the system event. Other than those two cases, we just can't support menu copying and pasting (though you'll find plenty of people trying to get around this, the only real way I know of is to is to use Flash) . Our only consolation is that Google can't do it either. Go ahead and try it with Google Docs. It only works in Chrome and IE. Use it in Firefox or Safari and you'll notice a nice little dialog telling you to use keyboard shortcuts.



Step #2 — Get Data in Multiple Formats To and From the Clipboard

We now have a reliable way of receiving clipboard events. Now we'll address the problem of moving data to and from it.

Let's start by describing what we want. Desktop applications can read and write any type of data on the clipboard. If it can be represented as a stream of bytes, it can go on the clipboard. For example, if I select multiple shapes and text areas in Microsoft PowerPoint, then paste into Microsoft Word, all the shapes and text are correctly pasted using some Object representation of shapes and text internal to MS Office. What's more, the application can put multiple data forms onto the clipboard. So, those shapes from PowerPoint can likewise be pasted into Photoshop or Gimp as an image. When you copy in PowerPoint, something like this probably happens.

```
clipboard.setData('text/plain', selection.getText());
clipboard.setData('application/officeObj', selection.serialize());
clipboard.setData('image/bmp', draw(selection));
clipboard.setData('text/html', ...);
...
```

PowerPoint is going to try to copy every possible useful data type to the clipboard so that other

applications can use whatever data type they recognize.

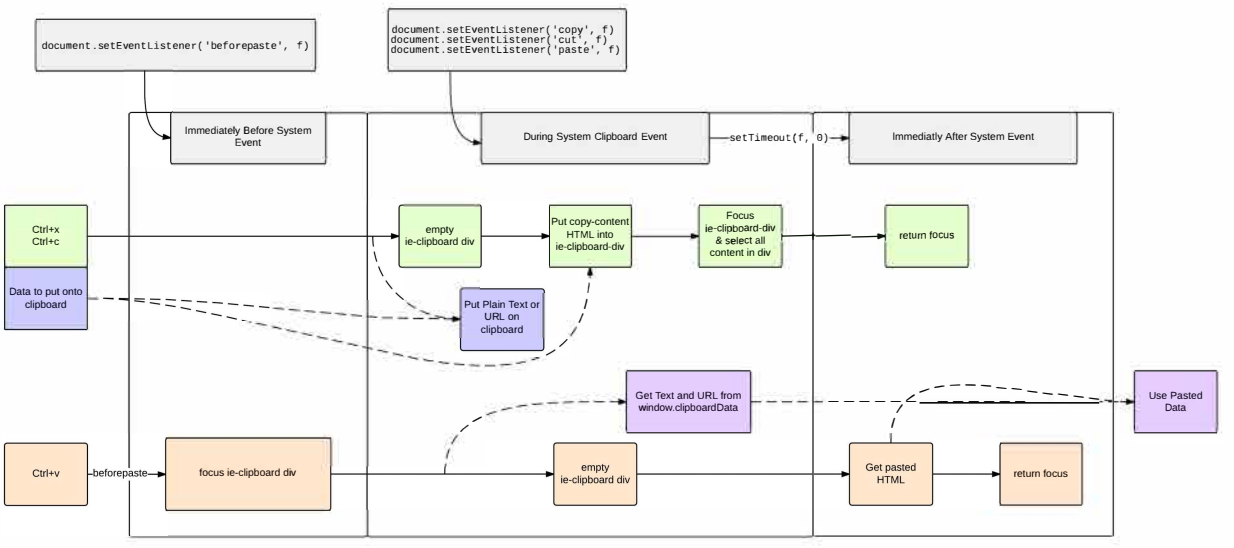
On the web, we don't have quite the same flexibility. While the standards describe a minimum support of a wide number of data types, we tend to be happy if we can just consistently access plain text on the clipboard. Right now, here is what the browsers actually support:

- **Chrome and Safari**: They support any content type on the `clipboardData`, including custom types. So, we can call `clipboardData.setData('application/lucidObjects', serializedObjects)` for pasting, and then call `var serialized = clipboardData.getData('application/lucidObjects')`
- **Firefox**: It currently only allows access to the data types described above. You can set custom types on copy, but when pasting, only the white-listed types are passed through.
- **Internet Explorer**: In true IE fashion, it supports just two data types: `Text` and `URL`. Oh, and if you set one, you can't set the other (it gets nulled out). There is a hack, however, that also allows us to indirectly get and set HTML.

The clipboard object in Internet Explorer doesn't expose `text/html` via JavaScript. It does, however, support copying and pasting HTML into `contenteditable` elements. We can leverage this if we let the browser perform its default copy and paste, but 'hijack' the events to get/put the HTML data we want. This would look something like the following code.

```
if (isIe) {
    document.addEventListener('beforepaste', function() {
        if (hiddenInput.is(':focus')) {
            focusIeClipboardDiv();
        }
    }, true);
}

var ieClipboardEvent = function(clipboardEvent) {
    var clipboardData = window.clipboardData;
    if (clipboardEvent == 'cut' || clipboardEvent == 'copy') {
        clipboardData.setData('Text', textToCopy);
        ieClipboardDiv.html(htmlToCopy);
        focusIeClipboardDiv();
        setTimeout(function() {
            focusHiddenArea();
            ieClipboardDiv.empty();
        }, 0);
    }
    if (clipboardEvent == 'paste') {
        var clipboardText = clipboardData.getData('Text');
        ieClipboardDiv.empty();
        setTimeout(function() {
            console.log('Clipboard Plain Text: ' + clipboardText);
            console.log('Clipboard HTML: ' + ieClipboardDiv.html());
            ieClipboardDiv.empty();
            focusHiddenArea();
        }, 0);
    }
};
```

Below is a diagram illustrating the entire process:



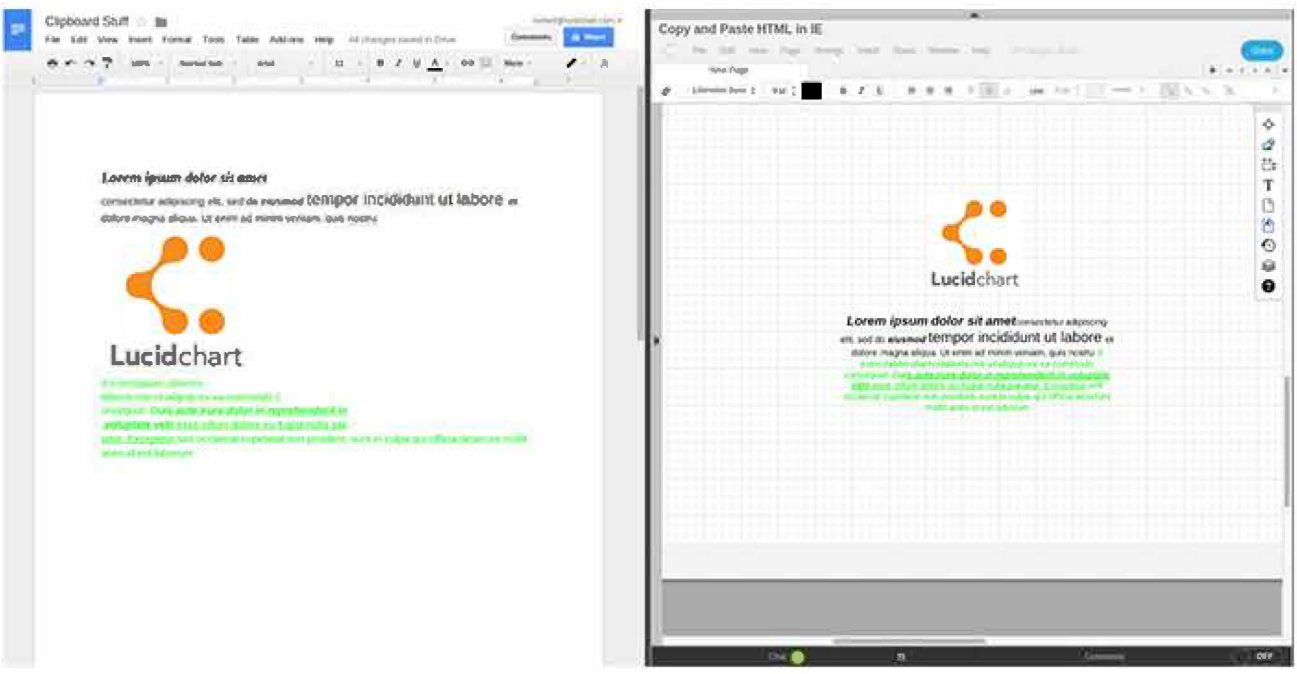
Here's what's going on:

- We have a hidden `contenteditable` div on the page just for copy and paste.
- During the `copy` event (before the system performs its default action), we set the div's HTML to whatever we want placed on the clipboard and programmatically select the entire content.
- Then when the system performs copy, it will get the desired HTML and it will be put on the system clipboard.
- To paste HTML, we shift the focus to the `contenteditable` element during the `beforepaste` event.
- Immediately after the system has performed its default paste, we just extract and clear the pasted HTML.

This final snippet gives a working example of how we copy and paste custom text and HTML across browsers using nothing but JavaScript. This is actually enough to support everything our old clipboard implementation was doing. We encode/decode our shapes to/from HTML so we can copy and paste shapes in the editor, and we get/set plain text on the clipboard so that we can copy and paste text to and from other applications.

Step #3 — Have Fun

Obviously, we didn't go through this mess solely to keep our previous functionality. Now that we have consistent access to the system clipboard, we are no longer limited to living entirely within our own application. Rather, Lucidchart users can share and receive data from other applications by using the system clipboard. For instance, it was an easy process to add the functionality for **pasting images** from a desktop screenshot or from another browser window. And we also now support the **pasting of formatted text** from Gmail, Google Docs, or other web pages.



We'd really love to hear what you think about this feature. Go ahead and try it for yourself!
If you have any ideas to improve it — or just want to commiserate about copying and pasting the web — do not hesitate to leave feedback on our support forums.

Useful resources

- Final Copy and Paste Example
- Pasting an image in IE11
- Pasting Images in Chrome