Use JavaScript to Poll for Necessary Resources

In a very basic sense, a timing issue is when you attempt to use a library (like jQuery), a selector, or another resource before it has fully loaded.

Timing issues are a constant concern for developers. Ensuring resources are available before they can be edited or altered is particularly important when you use JavaScript (and especially anything in jQuery, which needs to be called) to alter a webpage.

Timing issues occur when resources you reference are not loaded in the DOM (Document Object Model) by the time your JavaScript fires. Timing issues will vary from site to site and by each individual visitor's network speed.

Some visitors will have fast load times and are unlikely to experience any trouble if certain resources are referenced even when unavailable in the DOM initially because they will load resources very rapidly.

A visitor with a slower connection may run into timing issues when JavaScript references a resource or selector which has not yet rendered on the page. The JavaScript will likely fail to execute properly.

How Monetate Actions Overcome Timing Issues

Most Monetate actions are selector-based. This means they look for a selector within the DOM to act on. Hiding, inserting, or otherwise altering content relative to that selector.

Monetate uses backend JavaScript to re-check or poll for matched elements that in turn searches the DOM for the existence of the selector over a set period of time until it finds the selector or times out. If you enable polling/re-check, Monetate actions check for the selected element for up to 3 seconds even if it does not initially appear on the page.

This is a good option if you have certain elements that don't fire immediately when the page loads (for example, any AJAX– or JavaScript–generated content). If you disable polling, Action Builder only checks for the element once when your page loads.

Avoiding Timing Issues When Using Monetate's Insert JavaScript Action

Monetate's Insert JavaScript action makes it simple to add jQuery or pure JavaScript to a page. Like any on-page JavaScript, it's possible to run into timing issues with the Insert JavaScript action since this action is not fired based on a selector and does not have polling built in by default. Fortunately, there is a solution available.

If you want to utilize Monetate's append JavaScript actions effectively, the template below enables you to poll or re-check for a selector and for jQuery as well.

The Code

```
//Maximum time limit to continue looking for your object.
var timeout = 3000;

//How frequent to check for your object.
var interval = 50;
var isReady = function() {
    //Three examples provided below:
    //      Note that you can only ever have one of these return.
    //      Remove those you are not using for production ready code.

    //Does jQuery exist on the page yet?
    //return window.jQuery;

    //Do jQuery AND a SELECTOR exist on the page yet?
    //return window.jQuery && window.jQuery('.YOUR_SELECTOR')[0];

    //Without jQuery does a SELECTOR (ID, Class or Attribute selectors only) exist on the page yet?
    //return document.querySelectorAll('.YOUR_SELECTOR')[0];
};

//Code you want to run when isReady conditions exist in the DOM.
var onReady = function() {
    alert("jQuery is loaded."); //REMOVE THIS ALERT, It is for testing purposes only.

    //Insert your code here.

};

//Contingency plan code you want to run if isReady conditions DO NOT exist in the DOM by timeout
var opt_onTimeout = function() {
    alert("jQuery did not load."); //REMOVE THIS ALERT, It is for testing purposes only.

    //Optional Fallback JavaScript here! (http://youmightnotneedjquery.com/)

};

const checkDepLoading = function() {
    if (isReady()) {
        onReady();
    } else {
        if (timeout > 0) {
            timeout -= interval
            setTimeout(checkDepLoading, interval)
        } else {
            if (opt_onTimeout) {
                opt_onTimeout();
            } else {
```

```
            console.log("jQuery did not load before timeout.");
        }
    }
  }
}
// execute recursive checkDepLoading function
checkDepLoading()
```

The Code Explained

timeout

First, set your timeout variable. This should be an integer. It refers to the maximum amount of time in milliseconds that you will continue to check for the existence of a selector, jQuery, or both. The default is 3,000 milliseconds (3 seconds).

interval

Second, set the frequency variable, or how often Monetate checks for `isReady` conditions. This should be an integer. The default is 50 milliseconds.

isReady

The `isReady` function is where you determine what conditions you are looking for. There are three options listed (and commented out) in the template:

- `return window.jQuery;` — Look only for the existence of jQuery on your page. It asks "has jQuery loaded yet?," which will return either true (yes jQuery has loaded at this time) or false (no jQuery is not yet loaded in the DOM at this time). You now know whether jQuery is on the page and if you can use it.
- `return window.jQuery && window.jQuery('.YOUR_SELECTOR')[0];` — Looks for jQuery the same way, and also asks a follow-up question, "Is this selector also present?" You now know whether jQuery is on the page, if your selector is present in the DOM, and if you can use jQuery to target that selector.
- `return document.querySelectorAll('.YOUR_SELECTOR')[0];` — Uses vanilla JavaScript to look for a selector on the page. It asks "Is this selector rendered in the DOM yet?" You now know whether your selector is present in the DOM and if it is available to target with your JavaScript function. It is important to note `document.querySelectorAll` only supports IDs, Classes, or Attribute based selectors. This means you cannot use `:has(thing)`, `:contains(thing)` or `:not(thing)`.

You can only return one of the provided examples, and multiple returns will invalidate this check. Remove those snippets of code you are not using for production-ready code.

onReady

Place your code here. At this point, you have determined that your `isReady` conditions are true, and depending on which you have used, you now confirmed that jQuery, jQuery and a selector, or just a selector are present in

the DOM. You have a simple alert message in the template to confirm it works as expected. You should remove this from any production-ready action.

opt_onTimeout

This is your failsafe or fallback input. It is an optional input. In this section you can put any code you wish to execute if your `isReady` conditions return false. Again, you have a simple alert message in the template to confirm it works as expected. You should remove this from any production ready action.

checkDepLoading() function

Call the recursive `checkDepLoading` function, which checks for the `isReady` condition and then, if it returns true, execute the `onReady` function. If dependencies aren't yet loaded, wait 50 miliseconds (interval) and then call the `checkDepLoading()` function again. Repeat this up to 3 seconds (timeout).

Avoid Using setTimeout

`setTimeout` might seem like a perfectly logical and much simpler solution; however, there are some serious flaws with using this method.

`setTimeout(function(){ jQuery("body").toggleClass("main"); }, 1000);` does not check to see if jQuery is available to use. It simply says, wait one second and then attempt to run the jQuery `toggleClass`.

This code only runs once, whereas the template code provided above continues re-checking periodically. There is no fallback option either. If jQuery has not loaded after 1 second, the `toggleClass` will fail to execute properly.

You should consider the user experience first and foremost when developing. As stated earlier, each site's load times will vary by visitor based on many factors. If you don't include checks for resources and fallback options, visitors with slower connections may run into timing issues. This means a bad user experience or failed test and ultimately weakens your brand's online presence with particular users.

To run successful experiences and mitigate errors, check for references libraries and selectors before you attempt to alter them with JavaScript or jQuery.