Manage Flicker

Flicker is a visual artifact that renders for a site visitor as content loads. It's most often caused by switching (or hiding) a section of your native site content with content served via another front-end technology, such as Monetate.

As the page loads, a visitor sees the native site content beginning to load. Simultaneously, another front-end technology running elsewhere in your platform begins to load and deliver an alternative piece of content to the same section of the page. The resulting content seems to flicker from the original content to a new piece of content. This content switch can be confusing and irksome to a visitor and could negatively impact their behavior on your site.

Here's an example of content rendered without masking.

This example shows content rendered with masking.

Fortunately, Monetate has solutions you can implement to reduce or even completely eliminate flicker:

- Monetate tag-based implementation (front-end)
- Server-side integration (back-end)

Tag-Based Implementation

There are two steps to reduce flicker on your page if your site uses a tag-based implementation of the Monetate platform.

- Tag selection and proper tag placement
- Content masking

You must use both tactics in tandem if you want to reduce flicker.

> If you do not take both steps, you cannot reduce flicker on your page.

Using the Right Tag in the Right Location

There are two types of tags available for a front-end Monetate implementation:

- The asynchronous tag
- The synchronous tag

See Tag Comparison to better understand the differences between the two tag types.

> Only the synchronous tag supports content masking. During page load, masks are served in entry.js, which is not called by the asynchronous tag.

Using the right Monetate tag in the right location in your page's code lays the foundation for properly masking elements on your site. Monetate recommends that you use the synchronous tag in the `<head>` element of your page code in the highest priority possible.

> To avoid potential flicker-related issues, don't modify the Monetate tag in any way. Doing so can cause performance issues.

Monetate performance is based on more than just tag placement. The placement of JavaScript API calls (if your site is implemented with the Monetate JavaScript API) also significantly impacts performance. You should also define these calls as early in the page as possible.

If you use a tag management system, ensure that you configure it to deliver Monetate in the highest priority possible. Any delay in calling the Monetate tag may result in additional load-time delays and potential flicker.

Using Content Masking to Reduce Flicker

Once you have the correct tag in the correct location on your page, you must use content masking to reduce flicker on your page. Simply put, masking runs as fast as possible to hide the area where a piece of native site content would load, accomplishing this task with some simple CSS. A mask runs independently of any audience targeting and only uses information available in the request URL to decide whether or not to run.

If your site uses clearly defined, regularly structured URL paths, leveraging content masking is easy.

For example, your site URL is structured as follows:

```
/product/pid1234
```

You could use a regular expression such as this one to run a mask on all product pages:

```
^\/product\/pid[0-9]{4,6}
```

How Content Masking Works

A content mask applies the style `visibility: hidden;` to the specified element selector(s). The CSS `visibility` property keeps the allotted area of the element in the page flow so that the layout doesn't noticeably shift as Monetate inserts alternate content.

Monetate applies the masking CSS as quickly as possible to the masked elements outside of the normal delivery method of most actions, events, and targets.

Content masking does have some limitations. CSS selectors are the only elements that can be masked. Sizzle is not supported. Additionally, masks aren't restricted by any audience targeting and run on all URLs that they're

able to match.

> See Applying Content Masks in Content Masking to learn how to configure content masking in Action Builder. Refer to When Masks Execute in that same documentation for the technical details of content masks.

Content to Mask

You should mask any piece of native site content that will be hidden or replaced by content that Monetate serves as part of these actions:

- Hide actions
- Insert or replace image actions
- Insert or replace HTML actions

See Content to Mask in this documentation to learn more about the action types that may require content masking.

Site Structure

Because content masks are applied using CSS selectors, you must consider your site structure when you build actions. You should be able to easily identify `ID` or `class` attributes on sections of content that are "above the fold" or highly visible for a visitor to effectively apply masks for any content changes you plan to make within that section.

> Learn more about the basics of CSS selectors, IDs, and classes in Element Selectors.

If you're unsure about how your site content is structured, then you should consult your IT department to learn what is and isn't consistent in your site's page templates. Certain pages that use elements that are dynamically generated or pages on which content structure regularly varies are more difficult to effectively mask.

You can also consult your Customer Success Manager (CSM). Every client's site is structured differently, and you can discuss any technical specifics of your site with your CSM.

Testing Content Masking

Testing content masks is straightforward, but you cannot test masks in Preview Mode. You must test them live on your site, and they take up to 30 minutes to cache after you activate an experience to become visible.

The Monetate Inspector browser plug-in lets you easily determine which actions, targets, and events are running on any given page. Launch the tool on your site, click the **Components** tab, and then click **Content Masking** to view which element selectors have been masked on the page.

See Activate an Experience for Preview and Testing for more information.

Server-Side Integration

A server-side integration is the best option for clients concerned with flicker. This back-end solution can completely eliminate flicker. However, it requires a much deeper and more technical integration with the platform.

With a full server-side integration, you can use Monetate to determine what content you should serve to a site visitor as if it were native content. That means there's no competing content to render in a visitor's browser.

See What Is the Engine API? for more information about server-side integration.

If your company doesn't have the ability to implement a back-end solution at this time, you can still do a lot by properly using content masks to drastically reduce the effects of flicker.