MRAID Best Practices

UPDATED JULY 2014
ORIGINAL RELEASED APRIL 2014

This document was developed and is maintained and updated by IAB's MRAID Working Group.

Contents

Introduction

This document provides best practices and guidelines for designers building MRAID creatives, as well as clarifying requirements for implementing MRAID in containers/SDKs. Ambiguities in the MRAID 2.0 spec can lead to error prone creatives. We hope that this document will facilitate greater consistency of use of MRAID, while clarifying any misinterpretations that have sprung up across different development communities.

Common Mistakes

Networks and publishers supporting MRAID ads report seeing several common errors would-be MRAID creatives. In some cases, these errors result in a creative performing correctly in some MRAID containers but not in others, leading to confusion. Avoiding these errors will save time and maximize the odds that a rich creative will work across all MRAID implementations.

Failure to add an MRAID `ready` event listener to the creative

This could cause the creative to start making MRAID calls before the container is finished loading the MRAID libraries. This risks the creative asking the container for actions that the container is not yet prepared for, breaking the ad and causing it to behave in ways the designer does not intend.

Not optimizing for mobile devices when designing views

For example, this snippet of HTML for can be put into the ad to make the creative scale to the screen width. It controls the initial behavior of Webkit based browsers.

```
<meta name='viewport' content='width=device-width,initial-scale=1,maximum-scale=1'>
```

Navigating away from the ad with a URL instead of calling `mraid.open`

As per the MRAID 2.0 Spec, hyperlinks must not be used with for MRAID ads. Hyperlinks are not identified explicitly in MRAID and therefore MRAID-compliant containers will not treat hyperlinks in a consistent, predictable way. Some may ignore them entirely, some may leave the app and open them in the device's native browser. For consistent, predictable behavior, stick to mraid.open().

> "... ad designers should avoid using inline hyperlinks and window.location changes. mraid.open() is the appropriate and correct way for an MRAID ad to specify that a link should open a page in a separate browser. Loading a new web page in the ad view that is not written to the MRAID spec can leave the ad, and possibly the app, in an unusable state." - pg.25

Using 3rd party library bindings without using their `ready` event method

Third party libraries such as jQuery have a `ready` method which help facilitate a similar functionality as the mraid `ready` event listener. In the example of jQuery, putting the DOM click bindings **inside** of the `ready(handler)` method makes sure that the entire page has finished loading before starting execution of any other jQuery functions.

Binding to only DOM ready or only MRAID ready

If you are not using jQuery, you must still be aware that there are two ready states to your MRAID ad. Binding your initialization to just the `window.ready` event ignores that the MRAID libraries may not be available yet. Likewise, listening only for `mraid.ready` event ignores that the HTML DOM may still be rendering. Be sure to check for both ready states – and maybe even `isViewable` – before triggering initialization routines.

Inserting objects into the DOM in inappropriate locations.

Safari Mobile and iOS WebViews expect `document.appendChild` to be called on a specific node, and doesn't assume the `body` node otherwise. This means that if you would like to append a node to the body(or any other node for that matter), it should be clearly specified in the Javascript call as such: `document.body.appendChild()`. This is a common source for errors, as evidenced in StackOverflow

Failing to wait for isViewable before starting in-creative actions and animations.

For many legitimate user-experience-related reasons, apps may load ads off screen or otherwise out of the view of the end user. Most commonly this will happen to speed ad loads in cases with a page-style user interface, where the consumer swipes to load new pages of content. Ads that may be served into such environments should always check whether they are onscreen via MRAID's `isViewable()` method, before they begin any actions like animations. Not doing so risks the ad performs its bells and whistles without the user being aware of them, or that the ad comes into view with the creative partway through its behavior.

Misundertanding isViewable.

It's important to listen for **`isViewable`** to make sure that creative knows when to initiate actions. However it is also important to understand two limitations of **`isViewable`**. First, the MRAID spec was developed before the rise of the viewable impression as a new currency for buying digital advertising. As such, MRAID does not establish a specific definitional requirement for when an ad container should consider that an ad has gone from 'not viewable' to 'viewable,' or the other way. Creative designers, and ad servers should not attempt to use **`isViewable`** for the purposes of impression counting.

Second, MRAID doesn't require actions containers must take when the underlying app goes out of focus or is suspended. That is, MRAID does not require that opening a browser window or moving from one app to a different one should result in change to **`isViewable`**. The container as a whole should suspend all activities while the app is in background, resuming them if the app returns to the foreground. However, it may do so without an **`isViewable`** change to inform the ad that it was hidden for a while.

Attempting to resize or expand interstitials.
Interstitial ads in MRAID cannot change size and calls to `mraid.resize()` and `mraid.expand()` will result in an error. This also means that the state of an interstitial is `default` and NOT `expanded`. To hide the close button in the interstitial, `useCustomClose(true)` should be the first action when the `ready` event fires.

Attempting to use expand() multiple times in a creative.
To maintain simplicity and prevent ads opening large numbers of views, MRAID ads can only `expand()` once. If an ad is in an expanded state and attempts to call `expand()` again, MRAID will ignore that second call. For creative needs involving multiple size changes, do not use `expand()` at all. Use `resize()` instead.

Attempting to resize an expanded ad.
Similar to the previous point, ads in the expanded state cannot then call `resize()` to further change size. If a creative calls `mraid.expand()` and then subsequently calls `resize()` from the expanded state, the MRAID container will ignore the resize attempt.

Attempting to use open() rather than playvideo() for video URLs.
For platforms that do not reliably support the HTML5 video tag, you should use the mraid.playVideo(url) method.Using mraid.open can cause unpredictable behaviors and should not be used.

Missed opportunities to design ads for graceful degradation.
We see many examples of creatives where if an error happens they break totally when they don't have to. Designers should write MRAID ads such that if an error happens, the creative can try an alternative path or behavior, rather than simply failing.

MRAID Creative Requirements

Where the previous section highlighted common errors, this section offers some concrete advice, both for MRAID ads generally and for specific ad types supported by MRAID.

Initializing/Starting an MRAID Ad

- Always include or add "mraid.js" to the creative as early as possible. MRAID permits this either by including a script tag in the HTML, or via DOM insertion. This is a requirement for a

creative to be a proper MRAID ad. Some ad designers assume that the container will automatically inject the MRAID libraries (and some containers do actually do this) but the script tag must always be included to ensure proper ad behavior in all MRAID implementations.

- HTML technique

```
<html> <head> <script src="mraid.js"></script>
```

- DOM insertion technique

```
<script type="text/javascript">
    var head = document.getElementsByTagName('head').item(0),
        js = document.createElement('script'),
        s = 'mraid.js';
    js.setAttribute('type', 'text/javascript');
    js.setAttribute('src', s);
    head.appendChild(js);
</script>
```

- Start with the `MRAID.addEventListener` for `ready` as shown below. Put the rest of the MRAID code in `displayAd` or similar initialization function. The state must be "ready" before any MRAID APIs can be used. Failure to observe this requirement risks unpredictable failures for the ad when it tries to use MRAID functionalities that are not yet available to it.

Occasionally the ready event is fired before the creative has an opportunity to register a listener. Therefore using logic like this example represents a best practice.

```
function init() {
    var success = false;

    if (document.readyState === 'complete') {
        if (typeof mraid !== 'undefined') {
            if (mraid.getState() === 'loading') {
                mraid.addEventListener('ready', displayAd);
            } else if (mraid.getState() === 'default') {
                displayAd();
            }
            success = true;
        }
    }
    return success;
```

Two-Part Expandable Ads

- When creating 2-piece expandables the second "piece" must be a full HTML document and cannot be an HTML snippet.
- Also in the case of 2-piece expandables, ad designers must include the mraid.js script tag using the HTML method.
- Although the `mraid.expand(url)` method accepts a URL, using expand this way does not count as a click through. The end-user is still within a single ad experience. Only `mraid.open(url)` can be used to identify click through traffic.

Interstitial Ads

- An MRAID interstitial ad will include a close control and indicator, just like an expandable ad. In the case of interstitials where the close indicator would disrupt the user experience (e.g., interstitial ads in a magazine-style "page-turning" user interface, ad designers may use `usecustomclose(true)` on interstitial ads to suppress the default close indicator.

Resizeable Ads

- Be aware of how the creative is positioning itself within the container. Typically ad designers will position the ad at the top left of the container. In such cases, a resizeable ad that moves UP from the bottom of the screen will cause the banner to jump up to the new top-left corner of the resized container. Ad designers should always specify how they are anchoring the creative in the container, and to do so in such a way that the banner does not jump around the screen.

- Except in special cases, ad designers should ensure that a resized ad continues to cover the original banner space the default ad occupied.

- Except in special cases, ad designers should not use `mraid.resize()` to cover the full screen area of the device. If the ad needs to change size to a full-screen area, use `mraid.expand()`.

- Always use `setResizeProperties` before calling `resize`.

```
var resizeAd = function(){ var screenSize =
mraid.getScreenSize(); var resizeProperties = { "width":
screenSize.width, "height": screenSize.height/2, "offsetX": 0,
```

```
"offsetY": screenSize.height/10, "allowOffscreen": false }
mraid.setResizeProperties(resizeProperties); mraid.resize(); };
```

Miscellaneous MRAID Features/Capabilities

- When a creative is changing in several ways at once it is important to listen for both or all change events before proceeding. Different MRAID implementations may fire these events in a different order, so it is not safe to assume that one event firing means the sequence of activities is completed.

- If the creative needs to request a size, for example after changing size via `resize()`, don't listen for a statechange event, listen for a sizechange event. Even if the size is changing due to a statechange, listening for the sizechange event is more reliable and therefore the best practice.

- Always call `supports()` before calling any specific MRAID feature like `createCalendarEvent` and `storePicture`.

Debugging Tips

Developer Tools

For debugging JavaScript code, use the web debugging tools from browser vendors. One example of these tools is Chrome Developer Tools. There a variety of other similar tools as well, like Safari Developer Tools, and Firebug. Use the 'Console' section of the developer tools to get access to a live JavaScript interpreter with the current context loaded. This enables you to test out existing functions and add new things to the context to test. It can be also used for directing debugging logs to, from, or within your functions.

Console Logs

For testing expected output and other general debugging, Console Logs can be used through the function `console.log()`. There are various ways to format and output your messages to the console, and the full details are outlined in the article *Mastering Console Logging* by Alex Young for DailyJS

IAB MRAID WebTester

For testing creatives without an app, use IAB's open source MRAID WebTester at http://webtester.mraid.org/. This container is web-based and passes the IAB certification for MRAID v2. Because this tool is browser-based, you can continue to use your developer tools when using the WebTester.

In-App Testing

For testing creatives in-app, use tethering techniques for iOS and Android.

- Webplatform.org provides a useful overview of the many debugging resources available to the mobile developer.
  Instructions for remote debugging on Android devices can also be found at Google's Developers site.
  This useful post at Prototest.com provides detailed instructions for enabling remote debugging on both iOS and Android devices.
- Sometimes the usual remote debugging techniques will not work in-app, such as when debugging code running in a third-party, production-release application, or when using older iOS versions. In these cases, you can include the weinre libraries in your code to provide a suite of debugging tools similar to those available in Google's Chrome browser.

SDK Vendor-Focused Requirements

While the majority of this guide focused on advice for creative designers using MRAID, in some cases problems making MRAID creative work are the fault of the SDK or container, not the creative designer. The entire ecosystem benefits when MRAID implementations are uniform and consistent, and SDK vendors reading this document should be aware of the following:

Pass the compliance test.

Vendors claiming MRAID v2 compliance must be able to prove their container correctly runs the IAB's official certification suite ads, the documentation and links to which are located at iab.net/mraid. Providing consistent behavior is essential to maintaining a healthy ecosystem for vendors, clients, and consumers. The compliance test was officially ratified by the IAB in July 2014, and vendors that were previously certified as MRAID v2 compliant are or will soon be re-certified against the test.

Any vendor that has passed the official IAB test should have no problems with the notes called out below; however, the following two items are meant to clarify points that may be ambiguous.

Never fire an event before making the associated change.

An ad creative that requests a resize should be able to listen for a size change event as an indicator that the container has successfully completed the requested change. That is, the container should always complete a change and update any associated values before firing the associated event.

In the case of changes requested by an ad that result in multiple events being fired, MRAID does not specify a particular required order for those events. However, vendors should ensure that all required changes take place before firing the series of associated events.

Ensure containers report positioning reliably.

Vendors should take care that containers report creative positions accurately, particularly for resizeable ads. Ad designers that believe their coding is right and yet the ad is not getting the right values on a resize, should try using **getcurrentposition/getdefaultposition** to test where the ad is and how it is being moved.

About the MRAID Working Group:

The IAB MRAID Working Group has worked over the past four years to develop MRAID versions 1 and 2 and currently works to raise awareness of, support for, and compliance with this important industry specification. The MRAID Working Group is currently beginning efforts to collaborate with IAB's working groups managing the Video Suite and SafeFrame specifications to establish best practices for understanding when to use each of those options, and how best to deploy them in tandem to create rich, scalable ad experiences.

The MRAID Working Group includes representatives from the following companies:

- 4th Screen Advertising
- AccuWeather.com
- AdMeld
- Adobe
- ADTECH
- Amazon.com
- AOL
- Celtra
- CNN.com
- Conversant Media
- Crisp Media
- ESPN
- FreeWheel
- Goldspot Media
- Google
- IDG
- ImServices Group
- InMobi
- Innovid
- Medialets
- Microsoft Advertising
- Millennial Media
- Mixpo
- Mocean Mobile
- NBC Universal Digital Media
- The New York Times Company
- Nexage
- Opera MediaWorks
- PointRoll
- PricewaterhouseCoopers
- Rhythm NewMedia
- Sizmek
- Spongecell
- Time Inc.
- Turner Broadcasting System
- Univision Communications Inc.
- Velti
- The Weather Channel
- Xaxis
- Yahoo!

IAB Contact Information:

Joe Laszlo, Senior Director, IAB Mobile Center
Mobile@iab.net

Acknowledgement: The original source material for this guide comes from a similar document prepared by Millennial Media in 2013.