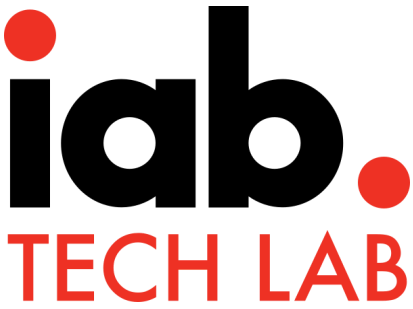

MRAID 3.0

Best Practices Guide

June 21, 2018

Executive Summary

MRAID, or “Mobile Rich Media Ad Interface Definitions,” is the common API (Application Programming Interface) for mobile rich media ads that will run in mobile apps. This is a standardized set of commands, designed to work with HTML5 and JavaScript, that developers creating rich media ads use to communicate what those ads do (expand, resize, get access to device functionalities such as the accelerometer, etc.) with the apps into which they are being served.

The last MRAID best practices documentation was released in 2014. Times have changed and the working group is excited to announced a new MRAID best practices document for 2018. This new best practices document clarifies many things that were either ambiguous or not well understood and gives helpful hints on how to use MRAID 3.0 to the fullest.

Audience

This best practices document is designed for Ad SDK developers and MRAID creative developers.

More information on MRAID is available at: https://www.iab.com/mraid

About IAB Tech Lab

The IAB Technology Laboratory is an independent, international, research and development consortium charged with producing and helping companies implement global industry technical standards. Comprised of digital publishers and ad technology firms, as well as marketers, agencies, and other companies with interests in the interactive marketing arena, the IAB Tech Lab's goal is to reduce friction associated with the digital advertising and marketing supply chain, while contributing to the safe and secure growth of the industry.

Learn more about IAB Tech Lab here: **https://www.iabtechlab.com/**

MRAID Working Group

MRAID working group has the following members. Special contributions to the best practices guide were made by ViralGains, Celtra, Yieldmo and Pandora.

Adform	Gruuv Interactive	Rocket10
AdGear	Homes.com	Sabio Mobile
Adobe	Hulu	Shazam
AdTheorent	IAB	Sizmek
AerServ	Improve Digital International B.V.	StackAdapt
Alliance for Audited Media (AAM)	InMobi	Taboola
Amazon	Innovid	Tapjoy
Anzu Virtual Reality	Integral Ad Science	Telaria
BARC India	Intowow	The Media Trust Company
Bonzai	Jukin Media	The New York Times Company
Celtra	Kiip	The Trade Desk
Chocolate	Leaf Group	Thinknear by Telenav

Conversant Media	Ligatus	Turner Broadcasting System
Cyber Communications Inc.	Liquidus Marketing	Twitter
Dentsu Aegis	LogoBar Enterprises	Vertebrae
Digital Advertising Consortium Inc.	Meredith Digital	Verve
Display.io	MGID	Videology
DoubleVerify	Microsoft Advertising	Vidooly
Flashtalking	mPlatform	ViralGains
Flipboard	NinthDecimal	Westwood One
FORTVISION	Oath	Yahoo
FreeWheel	OpenX	Yieldmo
Fyber	Pandora	YuMe by RhythmOne
Google	Pixalate	Zenith Media
	RhythmOne	Zentrick

* Working Group membership as of June 15, 2018

Learn more about the MRAID Working Group here (https://iabtechlab.com/working-groups/mraid-working-group/)

Table of Contents

Executive Summary **2**
Audience 2

About IAB Tech Lab **3**

MRAID Working Group **4**

Introduction **8**

Common Mistakes **9**
Failure to add an MRAID ready event listener to the creative 9
Not optimizing for mobile devices when designing views 9
Navigating away from the ad with a URL instead of calling mraid.open 9
Using 3rd party library bindings without using their ready event method 10
Binding to only DOM ready or only MRAID ready 10
Inserting objects into the DOM in inappropriate locations 10
Attempting to resize or expand interstitials 11
Showing two close buttons (useCustomClose is ignored) 11
Attempting to use expand() multiple times in a creative 11
Attempting to resize an expanded ad 11
Attempting to use open() rather than playVideo() for video URLs 12
Missed opportunities to design ads for graceful degradation 12

Ways of counting viewable impressions **13**

MRAID Creative Requirements **14**
Initializing/Starting an MRAID ad 14
HTML Technique 14
DOM Insertion Technique 14
Two-part expandable ads are deprecated 15
Interstitial ads 16
Resizeable ads 16
MRAID 2.0 event chains 17
Miscellaneous MRAID features 17

Debugging tips **19**
Developer tools 19
Console logs 19
SDK vendor focused requirements 19
Check your MRAID implementation with compliance ads 19
Never fire an event before making the associated change 20

Introduction

This document provides best practices and guidelines for designers building MRAID creatives, as well as clarifying requirements for implementing MRAID in containers/SDKs. While previous versions of MRAID had ambiguities and inconsistencies in the specification, MRAID 3.0 has clarified and removed many ambiguities, but this document serves as a reference for older implementations and to provide guidance on many common cases in the industry.

Common Mistakes

Networks and publishers supporting MRAID ads report seeing several common errors in 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 media 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 risk is 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. Creatives can also check the environmental variable for MRAID 3.0 to make sure they have been served into the proper container.

Not optimizing for mobile devices when designing views

For example, this snippet of HTML 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 3.0 Spec, hyperlinks must not be used with MRAID ads. Hyperlinks are not identified explicitly in MRAID, and 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()`.

When the user clicks on an HTML hyperlink (defined by an `<a href="">` tag) in an MRAID ad, there are two possibilities: the linked page could load in the existing webview, or the content could open a separate browser window and load the indicated HTML link there.

The open method must always use the native device or OS behavior or user setting for opening a URL from an MRAID ad. This will ensure the experience that the user expects and enable necessary device controls the user needs to navigate to and away from the external links. (From page 36 of MRAID 3.0 specification.)

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 the mraid.ready event ignores that the HTML DOM may still be rendering. Be sure to check for both ready states 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.

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.

Showing two close buttons (`useCustomClose` is ignored)

While MRAID 2.0 allowed creatives to suppress the SDK close button image, MRAID 3.0 has made the SDK provided close button image mandatory. This ensures a consistent experience across all ads in a SDK. It is important that any creative expecting to run in multiple versions of MRAID understand that MRAID 3.0 containers will always show a close button provided by the SDK. Even if the ad sets `useCustomClose` as true, the SDK will ignore this request and still show its own control.

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. The majority of environments that support MRAID 3.0 already support the HTML5 video tag. In these cases, using the video tag is the recommended way to play a video, within the ad, itself.

Missed opportunities to design ads for graceful degradation

There are many examples of creatives where, if an error happens, they fail 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 3.0 creatives that don't require specific MRAID 3.0 features should be able to run in a MRAID 2.0 container. If the ad requires MRAID 3.0 features, it is important to attempt to only serve in MRAID 3.0 placements.

Ways of counting viewable impressions

For purposes of viewable impression counting, creatives should use Open Measurement SDK first, then MRAID 3.0 `exposureChange` method, then can degrade to using MRAID 2.0 `isViewable` method and events to fallback. For MRAID 2.0 implementations, it is important to listen for `isViewable` to make sure that creative knows when to initiate counting viewable impressions. While MRAID did not specify exactly what the definition of `isViewable` was, it has been clarified to the point where `isViewable` is good enough to provide information that something is being shown on the screen.

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

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

DOM Insertion Technique

```
<script type="text/javascript">
(function(){
     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);

     js.addEventListener('load', function() {
         // mraid object has loaded, wait for mraid ready event
     });

     js.addEventListener('error', function() {
         // error loading mraid.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 are deprecated

As of MRAID 3.0, two-part expandable ads have been deprecated and should not be used. Only self-contained expandable creatives should be used for all MRAID ad experiences.

Interstitial ads

An MRAID interstitial ad will include a close control and indicator, just like an expandable ad. Like expandable ads, this close indicator is shown at all times by the SDK.

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 resizable 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 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();
};
```

MRAID 2.0 event chains

MRAID 2.0 ads should take care about making sure expected values from a series of chained events (expand causes a `sizeChange` and `stateChain`) occur at the last fired event. MRAID 2.0 containers were not mandated to fire in any order or make sure all variables were set for all chained events before firing any of them. All MRAID 2.0 mandated was that an event must fire after the variables it changed was done. For instance, in the case of an ad expanding, the stateChange event may fire and the state changes to expanded, but the `sizeChange` has not fired yet and the container size has not chained yet. It is important to listen to both events to make sure all changes have been done.

Miscellaneous MRAID features

In MRAID 2.0, when a creative is changing in several ways at once, it is important to listen for all chained 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.

MRAID 3.0 containers are now required to make all container size and state changes before firing any events that happen because of that change. This means that a creative can listen to the event it cares about.

For instance, if the creative wants to know when the container is expanded, it can just listen to `stateChange` and see that it was expanded or not. It does not need to also check `sizeChange`. The expectation is that the container makes all the needed changes and then fires the events.

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

Use the `unload()` command to prevent a poor user experience if the ad decides it will not or refuses to run.

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 (http://dailyjs.com/2012/02/02/console/)

SDK vendor focused requirements

While the majority of this guide focused on advice for creative designers using MRAID, in some cases problems making MRAID creatives 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:

Check your MRAID implementation with compliance ads

- Vendors claiming MRAID 3.0 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 for MRAID 2.0 was officially ratified by the IAB in July 2014. MRAID 3.0 compliance ads are in a public comment period. When released, these ads will provide SDK vendors the ability to make sure their containers are compliant with MRAID 3.0 specifications.

- MRAID 2.0 compliance ads for expansion (single part), resize, and interstitials should be used for compliance testing for MRAID 3.0 containers. During the public comment period, these MRAID 2.0 ads will be refreshed for use with MRAID 3.0.
- 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. tTe container should always complete a change and update any associated values before firing the associated event. This is required by the MRAID 2.0 specification. MRAID 3.0 also requires that any chained events be sent after all container changes and variables have been set.

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.