SafeFrame 2.0

Enabling safe, rich ad interaction without direct access to the webpage

Draft for Public Comment: July 29, 2020 – September 28, 2020 (60 days)

This document has been developed by the IAB Ad Technology Council

The SafeFrame specification was created by a working group of volunteers from 43 IAB member companies.

The SafeFrame Working Group was led by:

- Lucas Silva, Google
- Marian Rusnak, Verizon

The following IAB member companies contributed to this document:

Adform	Criteo	Line
Admiral Adblock Publisher Solutions	Cyber Communications Inc.	LinkedIn
Admixer EU Gmbh	Digital Advertising Consortium Inc.	Liquidus Marketing
AdPushup	Digitas LBI	Microsoft Advertising
Alliance for Audited Media (AAM)	DoubleVerify	Powerinbox
AMC Networks	ESPN	Protected Media
Audit Bureau of Circulations UK	Forensiq	Sharethrough
Axel Springer SE	FreeWheel	Starcom Worldwide
BARC India	Google	Terragon Group
Blue 449	GumGum	The New York Times Company
Bonzai	HyperTV	TripleLift
Browsi	IAB Japan	Unruly
BuzzFeed	Insticator	Verizon Yahoo Media
CBS Interactive	Integral Ad Science	Yomedia Network

In addition to member companies, guests from PreBid.org also contributed.

The IAB lead on this initiative was Katie Stroud

Table of Contents

Summary

SafeFrame 2.0 defines an API-managed iframe that enables an ad to interact with a webpage without accessing sensitive page data.

SafeFrame was originally released in 2013, with a minor update in 2014. It offered publisher page security while enabling rich ad interaction on web and web-based apps. It also enabled third party ad measurement and verification. However, in the last 6 years ad tech has seen advances in browser features, more advanced solutions for measurement, innovations in ad creative, and continued increase in technical integrations. SafeFrame in its original form had become implementation heavy and creatively restrictive.

SafeFrame 2.0 is a significant update from its last release. More lightweight with support for advances in browser security, measurement technology, and ad interaction, SafeFrame 2.0 is easier to implement. Among some of the changes are: removal of the required host implementation, guidance for sandboxing, guidance for use with header bidding, alignment with Mobile Rich Media Ad Interface Definition (MRAID) for easier ad conversion, and focus on ad interaction features with guidance for use of other measurement solutions. A more granular list of updates is provided in the following sections.

SafeFrame 2.0 enables safer programmatic placement at scale. Widespread adoption simplifies ad buys for agencies and brands while enabling publisher yield without sacrificing safety. Achieving adoption will require support from multiple parties. Publishers need to build an API to support the technical features described here. Ad developers need to design ad function to make use of SafeFrame 2.0 features. Programmatic vendors and other ad delivery platforms need to accept and support SafeFrame, usually communicating to other technical platforms about the use of SafeFrame in a requested or delivered ad.

IAB Tech Lab and the SafeFrame Working Group continue to work on resources to support SafeFrame implementation and adoption. Increasing adoption creates a safer ad experience for users.

Audience

SafeFrame 2.0 requires a host API that the publisher implements and an ad SDK that communicates with the API. Most of the technical details in this documentation is targeted toward ad developers. Additional guidance is provided for publishers on how to implement the host API.

Web technology vendors (SSP, DSP, measurement providers, brand safety, programmatic platforms, etc.) should also be familiar with SafeFrames and guidance around serving SafeFrame ads, including using any SafeFrame features and settings in the programmatic tools used to request and serve ads. Vendors may also be able to offer SafeFrame capabilities on behalf of publishers or ad developers and can use this documentation to build third party tools that offer SafeFrame functionality.

Updates

The requirements for updating SafeFrame to version 2.0 includes simplifying implementation while enabling flexibility.

- **Header Bidding:** Supporting SafeFrames in header bidding with previous versions was complex and only achieved with specialized modifications for vendors willing to make it work. While SafeFrame is absent of any specific features to support header bidding, guidance is provided to simplify the use of SafeFrame in a pre-bid context. In addition, work with header bidding technology providers has been done to encourage new features that communicate the presence of SafeFrame and operational conduct that guides the handling of SafeFrames.
- **Security:** Guidance for host implementation highlights taking advantage of browser security features such as HTML sandboxing and feature policy. These security features further prevent ad access to sensitive page data while enabling additional ad interactions.
- **MRAID Alignment:** SafeFrame 2.0 was modified to align with the nomenclature and syntax of MRAID to enable easier conversion between MRAID ads used in mobile native and SafeFrame used in web and web-based apps.
- **Measurement:** SafeFrame 1.0 was developed to support third party measurement in addition to ad functionality while encapsulated in an iframe. SafeFrame 2.0 was designed to focus on ad interactions while recommending other solutions for measurement.
- **Simplicity:** The scope of SafeFrame 2.0 is reduced to only focus on ad interactions in web and web-based apps while allowing for the support of other specialized services, such as measurement. To enable more flexibility for host implementations, the host requirements have been replaced with guidance for API implementation.
- **Compatibility:** The updates in SafeFrame 2.0 make it incompatible with previous versions. While this incompatibility was approved, guidance is to be provided on supporting version 1.1 simultaneously with version 2.0 during a transition period up until previous versions are deprecated.

Scope

The scope of this documentation for SafeFrame 2.0 is limited to technical feature description for the ad SDK. Additional guidance is provided for publisher implementation. Some guidance is also provided to help vendors support ad request and delivery, especially with regard to header bidding and other programmatic services.

Out of Scope

For the sake of clarity on what SafeFrame 2.0 offers, the following is considered out of scope:

- **Ad request and delivery:** SafeFrame doesn't specify how the ad is requested or delivered. However, guidance is offered around best practices for efficient delivery.
- **Integration:** SafeFrame does not define any objects, methods, or properties other than those to create, manipulate, and manage an iframe container. However, guidance to support further functionality, such as additional security, UI elements, etc. may be covered to enable adoption, but are not defined as part of SafeFrame 2.0.
- **Non-Browser Implementations:** This version of the SafeFrame is limited to JavaScript-formatted, browser-based implementations. SafeFrame may function within applications, such as those used in mobile devices (native), but details for non-browser based implementations are not included.

Summary of Release Versions

A history of updates for SafeFrame is summarized in the following table.

Version number	Date	Summary
2.0	3/24/2020	Several updates for simplicity and flexibility.
1.01	4/16/2013	Minor name corrections
1.0	3/18/2013	Original

Benefits

Since its initial release in 2013, SafeFrame offered secured ad operation on a page without adding additional code. Instead of slowing down a page and risking broken page function with added creative code, ads could include the API calls that communicate with a page-hosted SafeFrame.

SafeFrame 2.0 continues to offer all the benefits that come from this simplified arrangement for ad placement, but improvements with this release were designed to encourage more widespread adoption.

Page Security

While SafeFrame opens up communication between the page and the ad, the SafeFrame host controls what information is accessed or shared, if any, and to whom.

Stability

As with standard iframes, the clear delineation between a publisher's webpage and the ad creative reduces the chance for bugs in rendering content or interference with the publisher's JavaScript and HTML code. Because the ad is rendered with its own HTML document, its own set of CSS rules, and its own JavaScript, it cannot directly interact or override the publisher's JavaScript, HTML, or CSS.

Since code interference between the two parties is contained within the SafeFrame, the two parties can interact in a controlled and transparent way using the SafeFrame API.

Transparency

Data shared using the SafeFrame API enables ad insight for functionality and performance reporting.

Scale

The more an ad requires interaction with a publisher's page, the more ops-heavy the arrangement is to certify partners who are allowed to add script directly on the webpage. Despite lengthy certification and added cost, direct placement on a webpage compromises page security. Even if the advertiser and their vendors mean no ill will, ad script can still interfere with page function. The safety provided with a SafeFrame implementation scales a publisher's ability to increase yield without sacrificing safety. The increased supply may also help reduce costs to advertisers.

How It works

A SafeFrame is a cross-domain iframe hosted by a publisher and used as a container for ad interaction that prevents the ad from directly accessing publisher page content. Using an API, the publisher responds to requests the ad makes using the SafeFrame SDK.

For example, when a user interacts with the ad to prompt an expansion, the ad can request that the host expand the SafeFrame to new specified dimensions. Likewise, the ad can request feedback on the resulting current size and position of the SafeFrame.

This interaction is made up of a few working parts: the host main page, a cross-origin iframe, the API, and the ad creative (previously known as the external party).

Host Main Page

The SafeFrame host is typically a publisher platform implemented to restrict direct access to a webpage. The protected webpage is a “parent page” on which a cross-domain iframe is used to contain ad content.

Cross-Origin iFrame

The host sets up a cross-origin iframe in a publisher-hosted secondary domain. Taking advantage of the restrictions created by containing content in a cross-origin iframe, SafeFrame uses an API to allow communication between the contained ad content and the parent page without allowing direct access.

SafeFrame API

The SafeFrame API is a protocol that enables communication between ad scripts running inside a host-implemented iframe and the parent page on which the iframe is placed. The API is flexible enough to enable most rich ad function while protecting the parent page.

Ad Creative

Known as the “external party content” in the previous versions of SafeFrame, the ad creative is the content served into a SafeFrame-supported iframe. Developers who design ads can use a SafeFrame SDK to enable interaction on a webpage using the SafeFrame API. Each SafeFrame implementation can be customized to a publisher’s unique requirements. The ad can design for a full set of features with the ability to gracefully degrade when certain features aren’t supported.

Using Browser Security Features

Browser technology has introduced many new features over the years since the first SafeFrame release. While SafeFrame offers a secure interactive ad experience, unwanted behaviors such as auto-redirects and automatic downloads are difficult to prevent with SafeFrame alone. On the other end of the security spectrum, enabling features such as access to the microphone for ads that respond to voice is difficult from an isolated iframe.

Sandboxing is an W3C feature that restricts certain behaviors while Feature Policy is a standard web feature that allows selective access to certain features. SafeFrame 2.0 encourages the use of these existing features to extend the security and functionality of SafeFrame.

Sandboxing

The HTML5 sandbox feature adds an extra set of restrictions to an ad served into an iframe. It prevents malicious or otherwise unwanted ad behaviors such as: auto-redirect, modal dialog or new page pop-ups, automatic downloads, or plug-in exploits. Adding select values can maintain most of the sandbox restrictions while allowing certain interactions for improved ad function.

Sandboxing is applied in the cross-origin iframe set up for the SafeFrame implementation. The recommended setup is to enable the following properties:

- **Allow-top-navigation-by-user-activation:** if redirects are allowed only when the user initiates.
- **Allow-popups-to-escape-sandbox:** if the ad is allowed to open new windows without inheriting the sandbox.
- **allow-forms:** if the ad creative needs to support form submissions.
- **allow-scripts:** if the ad creative needs to run Javascript.
- **allow-same-origin:** if the ad creative needs to access same-origin resources hosted outside of the ad iframe, such as for ad measurement.
- **allow-popups:** a misnomer, this flag allows support for opening a new page only after the user clicks a link. The ad is still subject to the browser's pop-up blocking settings and any pop-ups cued by user interaction.

Feature Policy (Permission Policy)

The HTTP Feature Policy header by Mozilla enables publishers to allow or deny certain features such as access to camera, geolocation, document-domain, microphone and others. The full list of features can be accessed on Mozilla's developer site. Rather than restrict certain features such as with sandboxing, the feature policy allows access to certain features as set by the SafeFrame host.

By default, all available features are disabled to prevent malicious behavior. One example of malicious behavior is an ad that checks whether a device is charging before launching a malware download. On the other hand, an ad that uses voice control would need access to the microphone to function properly.

Enabled in the cross-origin iframe set up by the SafeFrame implementation, allowing access to most features protect the parent page, but some features may enable malicious behavior that affects the device. The host should consider how access to each feature might affect performance.

To allow access to select features, use the “allow” attribute in iframes to specify a “directive” (feature) and an “allowlist" to specify domains that may access the feature. Visit the Feature Policy page for more details about setup.

Header Bidding

Header bidding is a process that executes before SafeFrame is implemented. While SafeFrame ad placement is often rejected in the header bidding process, support for enabling SafeFrames in header bidding must be part of the process outside of the SafeFrame API.

Work is being done to support SafeFrame among some of the leading header bidding providers. Features have been proposed that communicate whether an ad will be placed in a SafeFrame, whether an ad is SafeFrame-equipped, and the details around the specific implementation (such as version number). By the time SafeFrame 2.0 is ready for full release, guidance will be available around a specific features to use and the best practices for using them.

Please contact support@iabtechlab.com to provide feedback and suggestions as we further develop the guidance around header bidding.

SafeFrame 2.0 Technical Details

A SafeFrame is an API controlled iframe that passes information between a webpage and an iframe. The iframe is on a cross-origin domain that the API host (typically the publisher) controls. Ads served into the SafeFrame can provide a rich ad experience for users with page interactions communicated using the API while the publisher blocks direct page access and protects sensitive data.

This documentation provides information on how ad developers can make use of the SafeFrame API to offer an interactive ad experience that honors the page on which it is run.

Change Log

SafeFrame 2.0 includes significant changes from its last release in 2014. These changes are detailed below.

Change	**Description**
Host Implementation	The host implementation details have been removed. The host may support SafeFrame using technology and operational practices best suited to company policies and capabilities.Guidance for implementation will be provided upon final release
Alignment with MRAID	Several functions have been modified, added, or deprecated so that SafeFrame aligns more closely with MRAID. This change enables a more simple transition from one ad standard to the other.
Modified features	The following existing features have been modified: ● **supports():** name changes for features supported ● **meta():** standardized header tags identified for simplified meta requests ● **expand():** operationally expands to maximum size allowed (the total viewport size in most cases), get/set ExpandProperties instead of to specified parameters

Change	**Description**
Removed features	The following features have been removed: • **register():** replaced with addEventListener() • **collapse():** replaced with close() • **_status_update():** replaced with addEventListener() • **geom():** replaced with getDefaultPosition(), getCurrentPosition(), getScreenSize() and getMaxSize() • **inViewPercentage():** replaced with recommendation to use IntersectionObserver.
Added features	The following features have been added: • **getVersion():** support for updates • **addEventListener():** replaces register() and status_update() • **removeEventListener():** complements addEventListener() • **close():** replaces collapse() • **unload():** supports request to have ad removed • **get/set ExpandProperties():** replaces setting parameters in expand() function • **getState():** replaces status() and partly _status_update() • **getCurrentPosition():** replaces geom().self • **getDefaultPosition():** new feature • **getMaxSize():** replaces geom().ext • **getScreenSize():** replaces geom().win

Managing the SafeFrame

Depending on the host implementation, an ad will either be served directly into a SafeFrame or the host may place a URL in the SafeFrame that calls for the ad as the SafeFrame is loaded.

Initialize

- Check the environment
- Check what the host supports
- Ask for metadata needed for ad operation

Monitor SafeFrame size and position

- Add an event listener for specific events
- Request details when an event occurs
- Set new properties and initiate a task

Resize the SafeFrame container

Resizing an ad in MRAID is executed differently than resizing an ad designed for SafeFrame. To allow for simplified conversion of an MRAID ad to a SafeFrame ad, we've added and changed components that support resizing as executed in MRAID. This support enables a simple namespace change for conversion.

While support for MRAID resizing enables easy conversion, ad developers may want to use the simpler approach designed for SafeFrame. If you are developing ads that won't be used in mobile and an MRAID conversion is not needed, support for the SafeFrame technique is also included.

The two resizing methods are detailed in the following sections.

Resize - MRAID Alignment

An MRAID ad treats resize and expand differently. Resizing allows for dimensions to be specified for the new ad size. Expanding simply expands to the full amount of space available.

MRAID-style Resize in SafeFrame

- Set resize properties
- Call resize() method
- Check resize properties: getCurrentPosition

MRAID-style Expand in SafeFrame

- Expand to maximum allowed size (usually the viewport size)

Resize - Web Only

If you're developing an ad that doesn't need to make use of MRAID, you can use the SafeFrame method to resize.

SafeFrame resize (expand)

- Call expand() with attributes that set the updated size

Using Intersection Observer

One of the guiding principles for developing SafeFrame was to allow for reporting on viewability. In SafeFrame 2.0, the focus is on ad interaction and security. Measurement and checking for viewability is often managed by vendors, and now with the use of Open Measurement for Web.

The inViewPercentage() method was removed in SafeFrame 2.0, but using the Intersection Observer API, you can observe the intersection between two elements--in the case of SafeFrame, the iframe element and the browser viewport. You can use the API to signal when the SafeFrame is a given percentage in view or whether it is close to being in view (using margins).

Using the returned values from the Intersection Observer API, you can trigger certain creative actions, such as animating a graphic when only a certain percent of the SafeFrame is in view. You can also create a parallax effect. While limited to working with large ads that intersect the edges of the browser viewport, the effect offers functionality that was difficult to achieve with the SafeFrame API.

Guidance on ways to use the Intersection Observer API in the context of SafeFrame is in development to help improve adoption.

Initialization

Initialization in SafeFrame 2.0 is asynchronous, which means that the API must load (event listener reports 'ready') before the ad can be executed.

Example

```
function safeFrameIsReady() {
  $sf.ext.removeEventListener('ready', safeFrameIsReady);
  doSomething();
}

if ($sf.ext.getState() === 'loading') {
  $sf.ext.addEventListener('ready', safeFrameIsReady);
} else {
  safeFrameIsReady();
}
```

Namespace

`$sf`

The SafeFrame namespace separates creative functions from other functions. For example, syntax for the function getVersion() is `$sf.getVersion()`.

SF_ENV

The ad creative can request information about the environment in which the SafeFrame is rendered. The meta() feature can also be used to request additional information that the publisher is willing to provide.

Example

```
SF_ENV = {
  version: ‘2.0’,
  sdk: ‘Host party implementation name’,
  sdkVersion: ‘1.0.0’,
}
```

getVersion()

Returns the SafeFrame version that the host is using.

Syntax	`var version = $sf.getVersion();`
Availability	Synchronous
Parameters	none
Returns	A string that represents the SafeFrame version that the host is using. Example: `"2.0"`
Compatibility	New in SafeFrame 2.0. Aligns with MRAID.
Related	SF_ENV

supports()

Returns an object with keys representing what features have been turned on or off for this particular container.

Syntax	`var bool = $sf.supports(feature)`
	`var supportedFeatures = $sf.supports()`

Availability	Synchronous
Parameters	feature (string)
Returns	Object containing a list of SafeFrame container features that are available (see Object Values list below)

Compatibility	Modified from SafeFrame 1.1. Available object values have been updated.
Related	

Object Values

The feature support list includes the following values:

`{Boolean} expand`
Whether or not ad expansion is allowed. Default value is `true`.

`{Boolean} resize`
Whether or not ad resize is allowed. Default value is `true`.

`{Boolean} resize-push`
Whether or not resizing is allowed in push mode, which pushes page content as the ad expands rather than expanding over the content. Support is dependent on host capabilities. Default value is `false`.

`{Boolean} read-cookie`
Whether or not the host allows external party content to read host cookies. The host must respect user privacy settings according to the policies in place before allowing cookie-read. Despite possible value of `true`, the host may reject cookie values if policy or privacy settings restrict 3rd party access to data. Default value is `false`.

`{Boolean} write-cookie`
Whether or not the host allows external party content to write cookies to the host domain.The host must respect user privacy settings according to the policies in place before allowing cookie-write. Despite possible value of `true`, the host may reject cookie values if policy or privacy settings restrict 3rd party access to data. Default value is `false`.

meta()

Used to retrieve metadata about the context of the SafeFrame specified by the host. This function allows access to certain meta tags without allowing page access.

Syntax	`$sf.meta(propertyName, ownerKey)`
Availability	synchronous
Parameters	**propertyName:** a string containing the metadata name for which value is to be read **ownerKey:** a string containing the owner object from which to read the property. Default value is "shared" meaning that it is common data.

Returns	String | Number | Boolean
Compatibility	SafeFrame 1.1.
Related	supports()

Example: 1 - Retrieve a shared metadata value

```
//External Party JavaScript code (inside SafeFrame container)

      var posID   = $sf.meta("pos");
```

Example: 2 - Retrieve a non-shared metadata value

```
//External Party JavaScript code (inside SafeFrame container)
//"rmx" == owner of metadata blob, "sectionID" is key to retrieve

      var sectionID     = $sf.meta("sectionID", "rmx");
```

SafeFrame 2.0 Properties

SafeFrame property functions allow for querying the host on the state, position, and size of the SafeFrame. You can also set expand and resize properties. These property functions are described in this section.

get/set ExpandProperties()

Checks expand width and height (get) or sets expand properties to desired width and height for running expand function (set).

Syntax	`var expandProperties = $sf.getExpandProperties()` `$sf.setExpandProperties({width, height})`
Availability	synchronous
Parameters	● width: number (in pixels) indicating adjusted width for SafeFrame container. Default is the full width of the viewport. ● height: number (in pixels) indicating adjusted height for SafeFrame container. Default is the full height of the viewport.
Returns	void

Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0.

Related	resize()
	getMaxSize()
	getState()

get/set ResizeProperties()

Resizes the container to the specified dimensions. This can be called multiple times with different dimensions.

Syntax	`var resizeProperties = $sf.getResizeProperties()`
	`$sf.setResizeProperties({width, height, x, y, allowOffscreen, push})`

Availability	synchronous
Parameters	● **width:** number (in pixels) that specifies width for SafeFrame container resize. ● **height:** number (in pixels) that specifies height for SafeFrame container resize. ● **x:** the horizontal delta from the current upper-left position to the desired resize upper-left position of the ad container. Positive integers move right; negative integers move left. ● **y:** the vertical delta from the current upper-left position to the desired resize upper-left position of the ad container. Positive integers move down; negative integers move up. ● **allowOffScreen:** (optional) Boolean value that indicates whether the resized ad container must be allowed to be drawn partially or fully offscreen. - true: offscreen positioning is allowed; host must refrain from repositioning ad container despite resulting in offscreen placement. - false: offscreen positioning must be avoided and the host must attempt to reposition the ad container. ● **push:** (optional) Boolean value that specifies whether the resizing of the ad container should “push” the content or overlay it. Push mode has to be explicitly supported by the host. Default: false.
Returns	void
Compatibility	New in SafeFrame 2.0, but provides similar functionality to expand() API in SafeFrame 1.1. Aligns with MRAID 3.0, with the following differences: ● “offsetX” and “offsetY” resize properties are renamed to “x” and “y”.

	• "push" property is added to ensure compatibility with push expansion in SafeFrame 1.1.
Related	resize()

getState()

Returns current state of the SafeFrame container.

Syntax	`var state = $sf.getState()`
Availability	synchronous
Parameters	none
Returns	a string with one of the following values: • loading • default • expanded • resized • hidden
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0. Replaces status() in SafeFrame 1.1.
Related	

getCurrentPosition()

Returns information of the current position of the ad container in pixels. Coordinates (x, y) are from the top left corner of the rectangle defining getMaxSize().

Syntax	`var currentPosition = $sf.getCurrentPosition()`
Availability	synchronous
Parameters	none
Returns	Object with the following values: • **x:** the leftmost x-coordinate of the SafeFrame container • **y:** the uppermost y-coordinate of the SafeFrame container • **width:** the width of the SafeFrame container • **height:** the height of the SafeFrame container
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0.

	Replaces "self" property of the object returned in $sf.geom() in SafeFrame 1.1.
Related	inVewPercentage()

getDefaultPosition()

Returns information about the default position of the ad container in pixels. Coordinates (x, y) are the top left corner of the SafeFrame container assigned as the default position.

Syntax	`var defaultPosition = $sf.getDefaultPosition()`
Availability	synchronous
Parameters	none
Returns	Object with the following values: ● **x:** the leftmost x-coordinate of the SafeFrame container ● **y:** the uppermost y-coordinate of the SafeFrame container ● **width:** number in pixels representing the width of the SafeFrame container at its default size ● **height:** number in pixels representing the height of the SafeFrame container at its default size
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0. Replaces $sf.geom() in SafeFrame 1.1.

Related	getCurrentPosition()
	getState()

getMaxSize()

Returns the maximum width and height dimensions allowed for expanding the SafeFrame container. May be equal to the viewport size or smaller, depending on the host implementation.

Syntax	`var maxSize = $sf.getMaxSize()`
Availability	synchronous
Parameters	none
Returns	Object with the following values: ● **width:** number in pixels representing the maximum width allowed for expanding the SafeFrame container ● **height:** number in pixels representing the maximum height

	allowed for expanding the SafeFrame container
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0. Replaces “exp” property of the object returned from $sf.geom() in SafeFrame 1.1.
Related	expand()

getScreenSize()

Returns the size of the bowser window or viewport.

Syntax	`var screenSize = $sf.getScreenSize()`
Availability	synchronous
Parameters	none
Returns	Object with the following values: ● **width:** number in pixels representing the width of the view port. ● **height:** number in pixels representing the height of the view port.
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0. Note that the function uses "screen size" but the "screen" in web is the browser window.
Related	getMaxSize()

SafeFrame 2.0 Events

In SafeFrame 2.0, the register() function is replaced with an event listener. Using the event listener, you can specify events for which you'd like notification. You can also remove events previously added. The addEventListener() and removeEventListener() are described in the Methods section.

The following events can be added to the event listener:

- error
- ready
- sizeChange
- exposureChange
- metaChange

error

Provides information about failed attempts to perform a task. The ad must register a listener in order to receive error events. Any number of listeners can monitor for errors of different types so that the ad can respond as needed.

An error event offers two parameters: one for an error message and one for the action being attempted at the time the error occurred. Both parameters are optional and the message option is typically used for debugging pre-flight creative.

Syntax	`$sf.addEventListener('error', function(message, action) {});`
Parameters	● message: (string) description of the error that occurred ● action: (string) name of SafeFrame API invoked when the error occurred
Triggered by	Any SafeFrame API
Compatibility	Replaces “failed” status update in SafeFrame 1.1. Aligns with MRAID 3.0.

Related	addEventListener()
	removeEventListener()

ready

Provides notification when SafeFrame is ready.

Syntax	`$sf.addEventListener('ready', function() {});`
Parameters	none
Triggered by	SafeFrame API is fully loaded and ready to be used.
Compatibility	Replaces synchronous initialization mechanism in SafeFrame 1.1 using $sf.register() with asynchronous event. Aligns with MRAID 3.0.
Related	getState()

sizeChange

Syntax	`$sf.addEventListener('sizeChange', function(width, height) {});`
Parameters	● width: (number) the width of the ad container ● height: (number) the height of the ad container
Triggered by	Change in the ad container width and height dimensions resulting from resize, expand or close.
Compatibility	Replaces "geom-update" status update in SafeFrame 1.1. Aligns with MRAID 3.0.

Related	expand()
	resize()
	getCurrentPosition()
	addEventListener()
	removeEventListener()

stateChange

If registered with an event listener, notifies ad creative when a change in the state of the SafeFrame container changes and provides the resulting updated state.

Syntax	`$sf.addEventListener('stateChange', function(state) {});`
Parameters	**state:** (string) indicating either "loading", "default", "expanded", "resized", or "hidden"
Triggered by	Calling expand(), reize(), close(), or when the host does something that changes the state of the ad container.
Compatibility	Replaces "expand" and "collapse" status updates in SafeFrame 1.1. Aligns with MRAID 3.0.

Related	getState()
	addEventListener()
	removeEventListener()

metaChange

Syntax	`$sf.addEventListener('sizeChange', function(propertyName, propertyValue) {});`
Parameters	propertyName: propertyValue:
Triggered by	
Compatibility	
Related	

SafeFrame 2.0 Methods

SafeFrame methods are sent to the host to request an action to which the host then executes.

addEventListener()

Method that initializes an event listener that provides notification when a specified event has occurred. This function replaces the process that used $sf.register() and $sf._status_update() in SafeFrame 1.1. The event listener allows for more granular event listening.

Syntax	`$sf.addEventListener(event, listener)`
Availability	synchronous: the event listener is added synchronously, but the events are provided asynchronously
Parameters	● event: (string) the SafeFrame event for which the listener should listen ● listener: (function) the function that the listener should use
Returns	void
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0. Replaces the process that used $sf.register() and $sf._status_update() in SafeFrame 1.1.
Related	removeEventListener()

RemoveEventListener()

Removes listeners for SafeFrame events previously registered using addEventListener().

Syntax	`$sf.removeEventListener(event, listener)`
Availability	synchronous
Parameters	● event: (string) the SafeFrame event the listener should remove ● listener: (function) the function for removing the listener
Returns	void
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0.

Related	addEventListener()
	error()
	ready()

close()

Moves previously expanded or resized ad into its default state. When called in the default state, it changes the state to "hidden" and hides the ad. Replaces $sf.collapse() in SafeFrame 1.1.

Syntax	`$sf.close()`
Availability	asynchronous
Parameters	none
Returns	void
Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0.
Related	resize()

unload()

A request to the host to unload the ad from the SafeFrame. The host may respond by removing the ad and replacing it or by removing the ad placement altogether.

Syntax	`$sf.unload()`
Availability	asynchronous
Parameters	none

Returns	void
Compatibility	New in SafeFrame 2.0
Related	

expand()

Expands the SafeFrame container to the maximum available space (see getMaxSize()). This can be called only when in the default state.

Syntax	`$sf.expand()`
	`$sf.expand(expandProperties)`

Availability	asynchronous
Parameters	none
Returns	void
Compatibility	In SafeFrame 1.1 the method supports parameters indicating size and direction of the expansion. Those were removed and behavior is changed to expand to the maximum available space. Aligns with MRAID 3.0.

Related	resize()
	getMaxSize()
	getState()

resize()

Resizes the container to the specified dimensions using the resizeProperties object. This can be called multiple times with different dimensions.

Syntax	`$sf.resize()`
	`$sf.resize(resizeProperties)`

Availability	synchronous
Parameters	resizeProperties: an object (optional) that specifies settings for the resize.
Returns	void

Compatibility	New in SafeFrame 2.0. Aligns with MRAID 3.0.
Related	get/set ResizeProperties()

Example: resizeProperties object

```
{
  width: 300,
  height: 250,
  x: 0,
  y: 10,
  allowOffscreen: true,
  push: false}
```

cookie(cookieName, cookieData)

If the host supports cookie read/write, cookie data may be passed through the SafeFrame API. The host must respect user privacy settings according to its policies and the governing region in which it operates. Even if the host allows this feature, third party access to users' personal data may be restricted, in which case the host will reject any cookie read or write requests.

Syntax	`var cookieValue = $sf.cookie(cookieName)`
	`$sf.cookie(cookieName, {})`

Availability	asynchronous
Parameters	**cookieName:** a string for the cookie being requested **cookieData:** (optional) An object that contains the value, and potentially an expiration date, of a cookie to be set. Without the cookieData object, a read only is being requested. To write a cookie, details are provided as follows: ● cookieData.info: a string for the cookie to be written ● cookieData.expires: (optional) an expiration date for the cookie
Returns	string | void
Compatibility	No change from SafeFrame 1.1.
Related	supports()

Example 1: Reading a Host Cookie

```
//Sample JavaScript implementation
var w = window, sf = w[“$sf”], sfAPI = sf && sf.ext, myPubCookieName =
“foo”, myPubCookieValue = “”, fetchingCookie = false;

function register_content()
{
      var e;
      try {
            if (sfAPI) sfAPI.register(300,250,status_update_handler);
      } catch (e) {
            //console.log(“no sfAPI -- > “ + e.message);
      sfAPI = null;
      }
}

function get_host_cookie()
{
      var e;

      try {
                  if (sfAPI && sfAPI.supports(“read-cookie”)) {
           fetchingCookie = sfAPI.cookie(“foo”);
                  }
            } catch (e) {
                  fetchingCookie = false;
            }
}

function status_update_handler(status, data)
{
      if (status == "read-cookie") {
            myPubCookieValue = data;
            //now do whatever here since you have the cookie data
      }
}
```

Example 2: Writing a host cookie

```
//Sample JavaScript implementation
var w = window, sf = w[“$sf”], sfAPI = sf && sf.ext, myPubCookieName =
“foo”, myPubCookieValue = “”, settingCookie = false;

function register_content()
{
      var e;
      try {
            if (sfAPI) sfAPI.register(300,250,status_update_handler);
      } catch (e) {
            //console.log(“no sfAPI -- > “ + e.message);
      sfAPI = null;
      }
}

function set_host_cookie(newVal)
{
      var e, cookieData = {value:newVal,expires:new Date(2020, 11, 1)};

      try {
                  if (sfAPI && sfAPI.supports(“write-cookie”)) {
           settingCookie = sfAPI.cookie(“foo”, cookieData);
                  }
            } catch (e) {
                  settingCookie = false;
            }
}

function status_update_handler(status, data)
{
      if (status == "write-cookie") {
            myPubCookieValue = data.info;
            //now do whatever here since the write was successful
      } else if (status == “failed” && data.cmd == “write-cookie”) {
            //data.cmd contains original command sent
      //data.reason contains a description of failure
            //data.info contains the object of information sent to host
            settingCookie = false;
            //cookie not allowed to be set
      }
}
```

--End--