SafeFrame

Version 1.1 Draft

Released: August 2014

This document has been developed by the IAB Ad Technology Council

The SafeFrame specification was created by a working group of volunteers from 21 IAB member companies.

The SafeFrame Working Group was led by:

- Sean Snider, Yahoo!
- Prabhakar Goyal, Microsoft

The following IAB member companies contributed to this document:

Adobe Systems Inc.	HealthiNation
AOL & ADTECH	Media Rating Council - MRC
Auditude	Microsoft
C3 Metrics	NBC Universal Digital Media
CBS Interactive	Network Advertising Initiative - NAI
Disney Interactive Media Group	OpenX Limited
Dotomi	Time Inc.
Editorial Projects in Education	Turner Broadcasting System, Inc./CNN.com
FDG	Undertone
FreeWheel	Yahoo!
Google	

The IAB leads on this initiative were Chris Mejia and Katie Stroud

Contact adtechnology@iab.net to comment on this document. Please be sure to include the version number of this document (found on the bottom right corner of this page).

INTELLECTUAL PROPERY NOTICE: Companies participating in the SafeFrame Version 1.0 Working Group made no patent commitments in the process of producing SafeFrame Version 1.0. Future versions of SafeFrame will be produced under the auspices of the forthcoming IAB Intellectual Property Rights Policy.

Details and resources for the SafeFrame initiative can be found at http://www.iab.net/safeframe.

About the IAB Ad Technology Council

The Ad Technology Council consists of more than 70 IAB member companies that have a primary business in advertising technology. It collaborates with the IAB Ad Operations Council in developing important technical standards and operating best practices for the digital ad industry. A select group of leading member businesses in the space also takes part in the Ad Technology Leadership Board, advising IAB's management and Board on top ad tech priorities.

The mission of the Advertising Technology Council is to develop and foster the adoption of technical guidelines and specifications that will reduce costs, open new marketplace opportunities, and ensure the long-term growth of the digital advertising industry.

A full list of Council member companies can be found at:
http://www.iab.net/advertising_technology_council

Table of Contents

Executive Summary

The SafeFrame1.0 technology is a managed API-enabled iframe that opens a line of communication between the publisher page content and the iframe-contained external content, such as ads. Because of this line of communication, content served into a SafeFrame is afforded data collection and rich interaction, such as ad expansion, that is unavailable in a standard iframe.

To avoid disruptive ad behavior and the potential security risks of serving ads inline with the page, publishers may choose to have ad content served into an iframe.

An iframe is a sort of mini HTML page within the publisher-hosted page. Using the iframe, ad content is sequestered within the boundaries of the iframe and unable to access any information about the page where it is served. Without access to page content, ad content within the iframe cannot expand, interact dynamically with site visitors, or collect any data necessary in determining ad effectiveness.

The iframe solution protects the publisher, but it also limits ad capabilities and decreases the value of inventory that is restricted to iframes.

SafeFrame's API-enabled iframe opens a line of communication between webpage code and the ad content in a controlled and transparent way. This communication allows for rich interaction while protecting the publisher's page from undetected changes that might otherwise damage page integrity.

Some key benefits of SafeFrame for digital advertising include:

Consumer Protection
While SafeFrame shares information with ad content served to its API-enabled iframe, the publisher chooses what to share and can protect sensitive consumer information like personal email addresses, passwords, or even banking information.

Publisher Control
The isolation between publisher code and ad code enables publishers to maintain control of the page layout and limit interference from ads while still allowing rich interaction and limited data collection. Using the SafeFrame API, publishers also have the ability to decide what website information (if any) should be exposed to which advertisers and vendors.

Publisher Efficiency
With the implementation of SafeFrame, publisher's can allow rich interaction from ads served to an iframe while maintaining control that prevents ad code from breaking page function. Enabling rich media inventory within SafeFrame improves revenue potential while keeping operational costs under control.

Standardized Advertiser Layouts
Advertising technology providers may standardize their rich media ad code so that it can run on any publisher network that adheres to the SafeFrame API protocol, reducing operational costs.

Support for Viewability and other Industry Initiatives
SafeFrame 1.0 offers mechanisms to support viewable impressions under development by 3MS as well as the DAA's AdChoices and other privacy initiatives. In fact, SafeFrame offers increased privacy controls previously unattainable in standard iframes. Also, the transparent communication enabled by SafeFrames establishes a foundation onto which support for other industry initiatives can be built.

The benefits that SafeFrames offer cannot be fully realized until several publishers have implemented the technology on their pages, and ad developers and technology vendors have made necessary modification, if any, to support serving ads to publisher-implemented SafeFrames.

The SafeFrame working group has built an open-source reference implementation to encourage swift adoption in the marketplace, but ad developers and technology vendors should be patient as publishers make the transition to SafeFrame 1.0.

Intended Audience

Technical details in this specification are primarily intended for site owners who would like to implement SafeFrames and for ad developers who develop rich ads that will use the SafeFrame protocols. Specifically, website and ad content developers can use the specifications in this document to develop SafeFrame protocols that enable communication between website content and any externally-served ads or other content.

Web technology vendors should also become familiar with the SafeFrame specifications to determine whether they need to make any modifications to support SafeFrame technology in the marketplace.

Also, SafeFrame is not limited to digital advertising and can be used by any client/server relationship.

Specification Updates

Version number	Date	Summary
1.0	3/18/2013	Original
1.01	4/16/2013	Minor name corrections
1.1	3/14/2014	Support for communicating whether top browser window is "in focus" (changes in 5.1 and added 5.11)

1 Overview

SafeFrame specifies a framework that creates a container around HTML content served to a webpage and establishes an API to enable communication between the webpage and the served content. With SafeFrame 1.0, specified objects and functions are used to manipulate and interact with created SafeFrame containers, allowing for rich-interactions. The primary use for SafeFrames is to encapsulate external HTML content with SCRIPT tags or other markup while protecting the host page from content that could otherwise inadvertently or purposefully affect the host site in unexpected ways.

As a solution for content served into iframes, SafeFrame offers visibility and functionality to served content where it was once impossible in a standard iframe.

1.1 The SafeFrames Components

SafeFrame manages the interactions between two parties: the host and the external party. The host owns the domain where content is displayed and the external party provides content, such as ads, to be displayed on the Host domain. In addition, SafeFrame interfaces with a secondary host domain to where external party content is served and from where it interacts with the Host by way of controlled SafeFrame API features.

These four key components if SafeFrame are described in more detail in the following sections.

1.1.1 The Host

The host, for purposes of this document, is the site-owned domain (or domains) on which content is displayed for an end user who accesses the content typically by way of a Web browser. In online advertising, the host is synonymous with "Publisher" but may be known by other names in other industries. The host implements the SafeFrame framework, including the API and a set of static resources (JavaScript™ and HTML files) used by the SafeFrame. The host is also responsible for rendering external content served to the SafeFrame container.

1.1.2 The External Party

The External Party, for the purposes of this document, is a content provider that serves content, or data that redirects to content, that originates from a source outside of the Host domain. In online advertising, the External Party may be the ad server, an ad exchange, ad network, or any technology organization that pushes ads to the Host domain.

In many cases, the external content provider may not need to modify content code at all, but when the content needs to interact with the host site in some way, such as to expand content, then the external party needs to include SafeFrame API details in a JavaScript-formatted tag.

General Implementation Note	The managed container technology offered with SafeFrame 1.0 only requires code modification to the external content if the external content needs to interact with the host site where it is served. For example, any expanding or floating behavior requires some code modification and must be done in JavaScript. Any rich interactions that remain confined to the SafeFrame container need no modification.

1.1.3 The API

SafeFrame specifies an API that provides communication protocols between the host site and external content. Using this API, the host site can provide information to the external content as necessary, and the external content can request services from the host site (i.e. expansion).

With implementation and industry adoption, the expectation is that other industry specifications and initiatives will extend the functionality of SafeFrame API. For example, the current specification provides building blocks for supporting Viewable impressions as being developed by the 3MS initiative and may be extended to support the AdChoices program of the DAA.

1.1.4 The Secondary Host Domain

SafeFrame operates in a secondary domain provided by the Host and ideally established on a content delivery network (CDN), which enables performance and improves availability. This secondary domain serves as an agnostic processing space between the host and external party. Any information that the External Party needs to know about the Host domain is accessed by request and provided using the SafeFrame API.

1.2 Benefits

SafeFrame offers benefits to owners and operators of either the host site or the external content.

1.2.1 Transparency between External party and Host

SafeFrame offers mechanisms for sharing information between the host site and external content. Some possibilities for information that can be shared include details like: metadata specified by the host site, geometric location of the SafeFrame container, and whether SafeFrame container is in view.

1.2.2 Unified and Standardized API

With industry adoption, having a common API that all advertisers and other external content providers can have to communicate with the host is a foundation for cleaner interaction between the two.

1.2.3 “Sandboxing” External content

SafeFrame renders content into a container that creates a clear delineation between 3rd party content and the host content. This barrier creates a "sandbox" that automatically gives the host some peace of mind, providing the following features:

Base Level Security
SafeFrame is essentially an iframe with an API that enables communication between the host and external content and offers the same base level security that an iframe does. While the API opens up communication between the two parties, the host controls what information is accessed or shared, if any, and to whom.

Stability

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

Also, the external content cannot be rendered out of place. For example, delivered raw, external content can render itself anywhere within a host's page, overlapping host content and functionality or displaying under host content and out of view.

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

Performance Measurements

Clear delineation enables hosts to measure how long external content takes to load, which is already possible for content served into iframes. But in the case where external content is served inline with host page code, load times of external content cannot be measured because it's not contained and can render in multiple places within the page. With SafeFrame's interaction capabilities made possible with its API, a viable option is made available to move external content once served inline to a SafeFrame.

The Best of Both Worlds: Rich Media and Data Collection

Website owners are faced with the decision to either allow rich media to serve inline with page code on their site, allowing unlimited access to page data, or serve the content into an iframe, which prevents most rich media interactions and restricts most or all data collection.

Serving content into a SafeFrame enables both rich media interaction and controlled data collection, enabling new functionality for iframe-served content and offering a viable alternative for serving rich content into an interactive SafeFrame instead of inline on the host page.

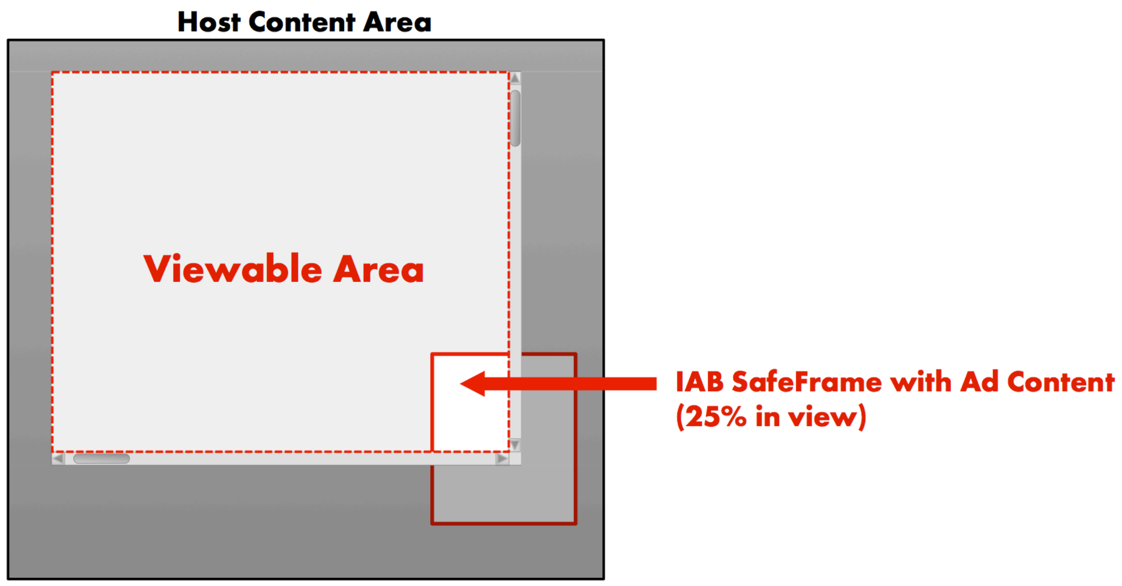
1.2.4 Host Customization & Control

The host can customize the SafeFrame API to add functions that control what features and rich-interactions are allowed. Hosts can also render new content or unload content at any time.

1.3 SafeFrame and Viewable Impressions

SafeFrame creates a container around ad content, preventing that content from directly accessing information about the host webpage or application. The content can, however, request information and interact with the host by sending and receiving messages using the SafeFrame API. These messages enable the host to share select information with the external content, including geometrical information that enables the external content to determine whether it is in view.

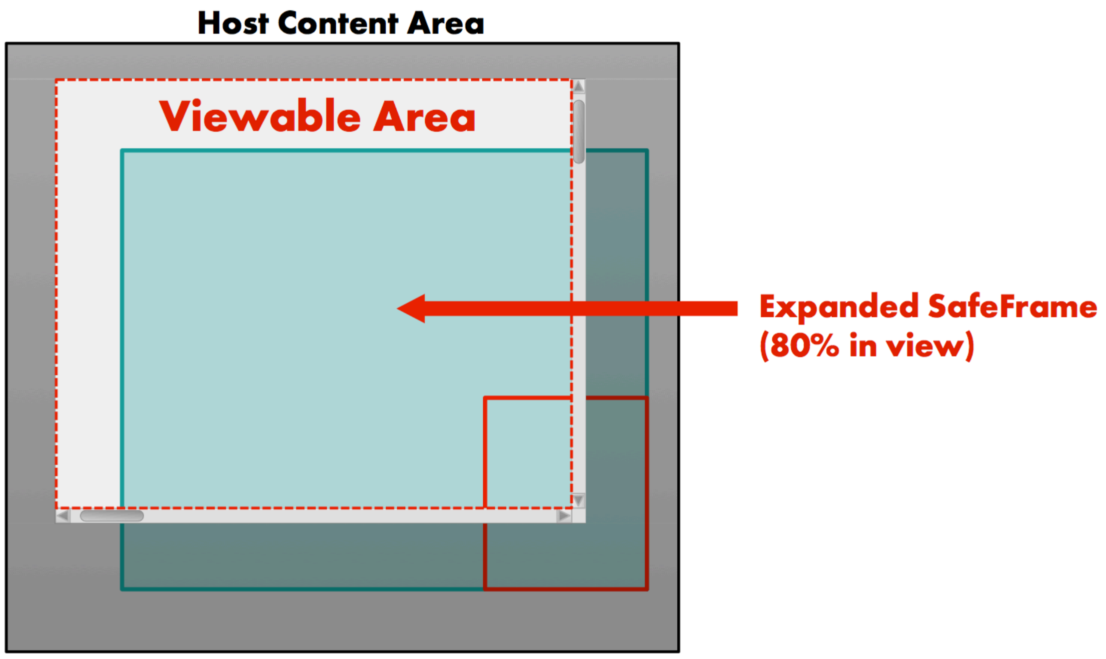
The following diagram illustrates how SafeFrame content may display on a Webpage and shows that the ad content served within the SafeFrame is 25% in view.



Using the SafeFrame external call `$sf.ext.geom`, the external party serving the ad content can determine where the SafeFrame is in relation to both the viewable area and the total host content area. Using the supplied dimensions, the external party can determine how much of its content is in view.

Other SafeFrame calls enable external content to expand beyond the boundaries of the SafeFrame. The `$sf.ext.geom` call also enables the external party to determine how far it can expand and how much of the expanded content is in view.

The following diagram illustrates an example of how SafeFrame content might expand within a webpage and how much of it is in the viewable area.



For more information, please see section 0, which covers the `$sf.ext.geom` call in detail.

1.3.1 Supporting 3MS Viewability

The SafeFrames specification is working to align with other industry initiatives surrounding viewability and improving the quality of advertising metrics, namely the Making Measurements Make Sense (3MS) initiative. As of the release of SafeFrame 1.0, viewability was still being tested and a formal recommendation had yet to be established. As SafeFrames and 3MS viewability recommendations are established and adopted in the industry, updates to SafeFrame can be made to more seamlessly support 3MS recommendations.

Until then, SafeFrame 1.0 offers geometric values that hosts and external parties can use to determine how much content was in view. Hosts and external parties should collaborate and come to an agreement on how much content should be in view in order to claim a "viewable" impression.

1.3.2 Viewability Features Optional to External Party

The SafeFrames framework allows external content to determine if it is in-view by calling the `$sf.ext.inVewPercentage` method. This feature, as with all SafeFrame features, are optional to the external party who serves content to the Host. However, the host is required to provide viewability data whenever the external party requests it.

1.4 SafeFrame and In-Stream Video

SafeFrame wasn't designed for in-stream video ads, but companion banners served in a VAST tag can be served into SafeFrames. The IAB Video Suite will need an update to fully support SafeFrame rendering, but video publishers can already use SafeFrame in place of any iframes they use on their video webpages.

Any VAST companion banners specified as an iframe resource can be rendered within the implemented SafeFrame. To ensure VAST companion banners can be served into a SafeFrame, the external party must specify the creative resource as an iframe. Any SafeFrame protocols included in the VAST companion banner content can be used by any video publisher that supports SafeFrame 1.0.

Possibilities exist for rendering VPAID content within a SafeFrame, but implementing a solution requires modification from both the video host and external party. Until the IAB VSuite is updated to more readily support SafeFrame, technical operation for using SafeFrame to render VPAID content must be addressed by parties who wish to implement the solution.

1.5 SafeFrame and Mobile

Any SafeFrame content that is rendered in a webpage can also be rendered in a mobile device just as the webpage would. Browser-based Web applications, including those designed for mobile, can also fully benefit from SafeFrame implementation. However, while SafeFrame can work in a non-browser application, such as those developed specifically for mobile devices (native apps), details for non-browser support is excluded from SafeFrame 1.0 and will be considered for a future SafeFrame release.

1.6 Reporting SafeFrame Data

SafeFrame 1.0 provides data you can use in reports, but the mechanism for reporting data is not done using SafeFrame. Reporting systems must be configured to collect and report SafeFrame data in a format that recipients expect.

1.7 Differentiation from Other Specifications

IAB SafeFrame solves a problem that no other IAB specification or guideline solves. The following sections help differentiate SafeFrame from other specifications in the industry that solve different problems.

1.7.1 IAB Friendly iframes

The IAB SafeFrame specification is different than that of the IAB Friendly iframes documented in "Best Practices for Rich Media Ads in Asynchronous Ad Environments."

The IAB Friendly iframe best practice is used to support rich media ads served with JavaScript calls that don't work with dynamic coding frameworks like AJAX. Using a Friendly iframe, content from an ad server is rendered into a frame originating from the same server as the host content.

While the Friendly Iframe solution addresses cross-platform barriers to supporting certain rich media formats, it does not isolate external content from the host content. The rich media content in a Friendly Iframe is served directly from the same server as the host's.

In contrast, SafeFrame enables isolation between the host and external content and provides an API to enable controlled and transparent interaction while providing a minimal layer of security and stability control for the host. With SafeFrame, ad content is served from a neutral domain instead of from the same source as the host content. Access to the host site domain where the ad content is eventually rendered is allowed only through an API specified in the SafeFrame guideline.

The SafeFrame guidelines remove many of the security risks associated with serving rich media into host-originated iframes. The SafeFrame API also allows for more transparency between host and ad content as well as more controls and monitoring tools over rich interactions. This added control also ensures improved rendering capabilities of rich media.

1.7.2 Cross Origin Resource Sharing (CORS)

Cross Origin Resource Sharing (CORS) is a mechanism used to enable cross-origin HTTP requests. Without CORS, the Same Origin Policy prevents imbedded code from one domain to request potentially damaging content from another domain. Exceptions exist for benign content such as images.

Unfortunately, the Same Origin Policy prevents content such as that in rich media ads from requesting scripted files to support proper rendering and interaction. Using CORS, the content can allow the browser to make cross-domain requests to trusted domains. While this solution is effective, it also grants the external content full access to the website, enabling the external content to access and change website content without the website's knowledge.

CORS simply manages whether one domain can make requests to another; it cannot manage interactions between the two.

An alternate solution is to have content served into an iframe. In this solution, the content is processed within the external server that is serving the content. But with this solution, all access to the website is denied, disabling any rich interactions with the external content.

In summary, CORS has no control over what requested content can access, and a standard iframe bars any access at all.

Using an iframe and an API, SafeFrame enables rich interaction between the website and externally-served content while allowing controlled access to website information.

1.8 Out of Scope

For the purposes of defining this specification the following items are considered out of scope:

Fetching / Retrieving External Content

SafeFrame doesn't specify how external content is retrieved. Once retrieved, external content is placed into the SafeFrame `$sf.host.Position` object as a string or URI with whatever metadata the host chooses to use.

Additional Features for Specific Implementations

SafeFrame does not define any objects, methods, or properties other than to create, manipulate, and manage a container. Further functionality, such as additional security, UI elements, etc. may be added but are not defined as part of SafeFrame 1.0.

Non-Browser Based Implementations

This version of the SafeFrame is limited to JavaScript-formatted, browser-based implementations. SafeFrame 1.0 may function within applications, such as those used in mobile devices (native), but details for non-browser based implementations are not included in this version of SafeFrames. Browser-based applications, for mobile and any other device, are supported.

1.9 Operational Considerations

Using SafeFrames may alter certain familiar operations. Before implementing SafeFrames, consider some of the following operational effects and determine whether your technology and processes need any modifications to take advantage of SafeFrame benefits.

The following considerations are intended to provide a starting point for an implementation analysis. Further evaluation and testing may be needed to ensure a smooth implementation and transition.

Contextual Display Advertising

Ad tags that programmatically serve ads based on page content access host page data to identify the type of content it displays and then serve an ad appropriate to the content. With SafeFrames, the Host API has to pass this data to the ad tag since the ad tag can't access the host page directly.

The ad tag provider should work with the host page owner (the publisher) to negotiate what information should be passed in order for the ad tag provider to serve contextual display ads.

For more information, see section 5.8 for details about the `$sf.ext.meta` function.

Access to Host URL

When External Party content is initially loaded onto a host site using an iframe, HTTP headers generally indicate the URL for the host site, but not very accurately. Since SafeFrame is an iframe, this URL inconsistency also occurs in SafeFrame. To retrieve an accurate Host URL, use the `document.referrer` property in JavaScript as you would in a standard iframe.

Setting Cookies

With SafeFrame, external content is rendered within the SafeFrame domain, which is different than the host domain. Cookies can be set and read in this secondary domain, but to set and read cookies in the host domain, the host must declare whether it supports the `$sf.ext.cookie` feature. Also, even if cookie read and write ability is supported, the host controls when and to whom cookie data is shared. If external content requires setting and reading cookies directly in the host domain, this access must be negotiated with the host page owner.

Third Party Data

Using the SafeFrames API, any data shared from the website (host) domain is provided by the host. In business models where a third party collects data from the host on behalf of another party, the third party in a SafeFrame implementation has to rely on data provided by the host (first party) rather than collecting it directly.

While first party-provided data may raise concerns about integrity, the data being shared is used to render served content correctly. Sharing incorrect data goes against the best interests of the host because it could cause incorrect rendering and interaction, which interferes with the host's page content. In addition, website owners who implement SafeFrames can be audited to ensure the integrity of data shared across the SafeFrame API.

Supporting Ad Content of Unknown Dimensions

Some ad serving models involve allocating ad space for ads of a particular set of criteria without knowing the details about which ad will be served when the call is made. In these cases, width and height are unknown at the time the call is made for the ad. SafeFrame 1.0 does not support this model directly, but can be supported using existing SafeFrame features combined with support for "push" expansion technology.

If the host declares support for push expand technology, ad content of unknown dimensions may be accepted by providing initial dimensions that can be resized to larger actual dimensions using the `$sf.ext.expand` feature to expand the SafeFrame container. The push method, if supported, is the optimal method for expanding in this scenario, but using the overlay expand method is an option.

Other technology and processes may need modification to accommodate SafeFrame operations. Consider running a thorough analysis and product testing before going live with SafeFrames.

2 Host Implementations

In a browser based SafeFrame implementation, the Host side of the API is written in JavaScript, and must provide the list of functions and namespaces listed in section as defined. The mechanism for browser based implementations is to use an iframe tag to create a container for external content, along with additional JavaScript code for facilitating functionality and communication with the external content.

Browsers are graded on the level of functionality they support. A-grade browsers are known, capable, modern, and common. All A-grade browsers with JavaScript activated shall be supported. C- & X-grade browsers are more rare, less capable, and antiquated. Host parties may support these browsers at their own discretion. SafeFrame relies mainly on the HTML5 "postMessage" function as the low-level means of communication between the iframe and the host. While the `postMessage` function offers optimum performance, other mechanisms may be used to facilitate communication between the two parties, especially in the cases of C- & X-graded browsers.

For more information on the HTML5 `postMessage` function, please visit:

http://dev.w3.org/html5/postmsg/#posting-messages

2.1 How SafeFrame Works

The goal of SafeFrame is to deliver content from an external source (external content) into a SafeFrame container and rendered onto the host site. External content can be delivered in one of two ways:

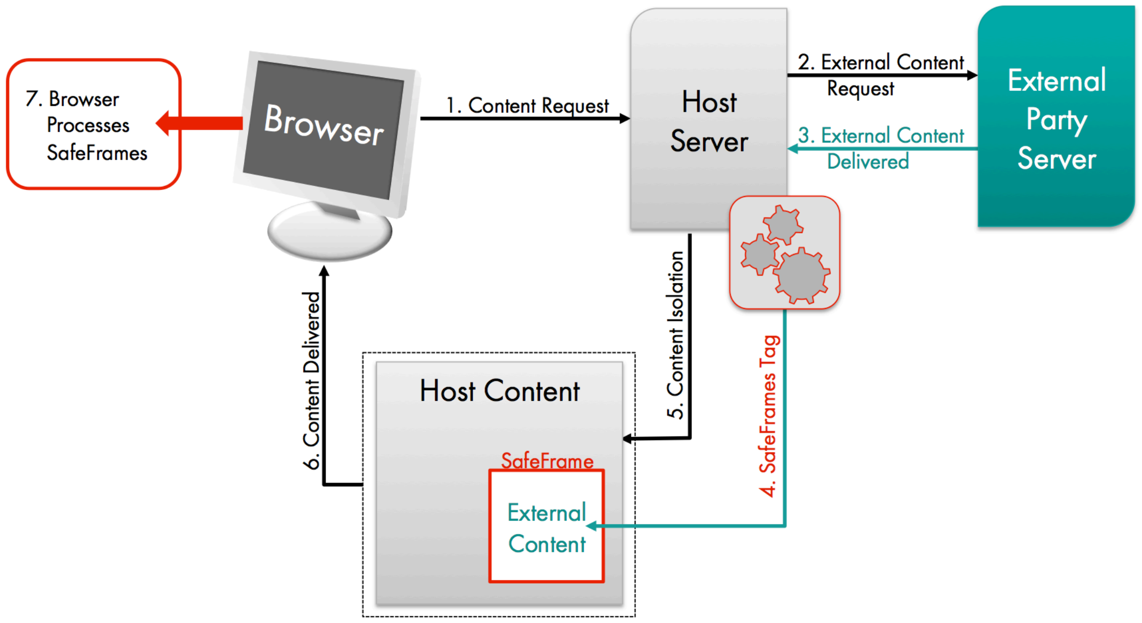
Delivery Mode A: Host code transforms external content and renders in SafeFrame
When the browser contacts the host webserver, the host may retrieve external content from its own backend systems. In this delivery mode, the host can transform external content into inline JavaScript structures using SafeFrame tags that can then be rendered by a SafeFrame implementation.

Delivery Mode B: External content is delivered directly into the SafeFrame
Hosts may not have mechanisms in place to transform external content on their own webservers. In this delivery mode, they still provide the same type of inline JavaScript structure with SafeFrame tags, but instead of placing the external content directly into the structure, they provide a URL to the content. In this case external content is delivered directly into the SafeFrame container using a SCRIPT tag and the URL that was specified.

The following sections describe this process in more detail.

2.1.1 Delivery Mode A: Host transforms external content

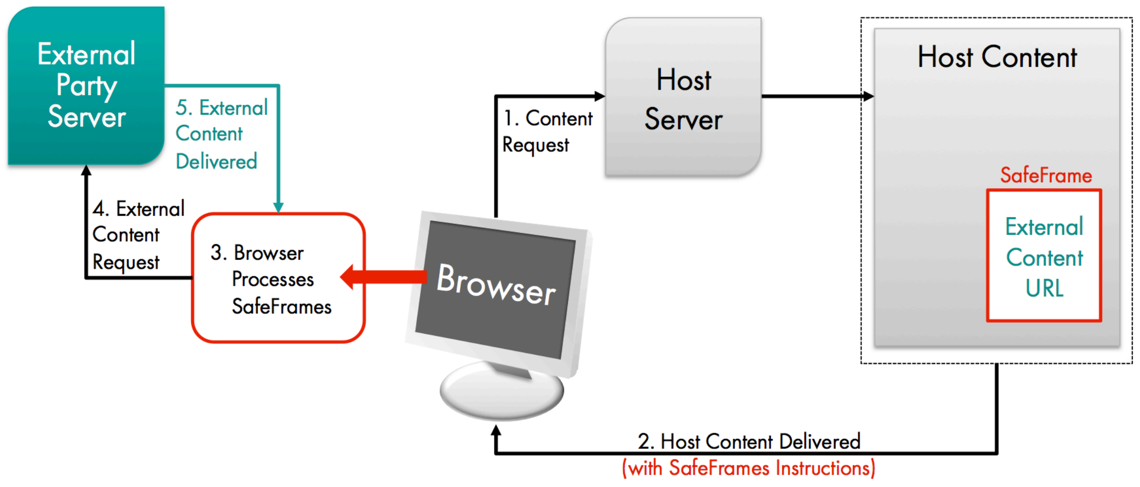
The following diagram illustrates the process for delivering external content into the SafeFrame container using delivery mode A.



1. **Content Request:** When an end user visits a website, the browser requests content from the host server.
2. **External Content Request:** The host requests content data from the external server.
3. **External Party Delivers:** The external party delivers HTML content as data.
4. **SafeFrame Tag:** The host transforms external content data to be served into the SafeFrame container using SafeFrame tags.
5. **Content Isolation:** Host content is isolated from external content.
6. **Content Delivered:** Host content, along with external content wrapped in a SafeFrame container, is served to the browser.
7. **Browser Processes SafeFrame:** The browser uses SafeFrame instructions from the delivered host content to process interactions between host content and the external content.

2.1.2 Delivery Mode B: External content delivered directly

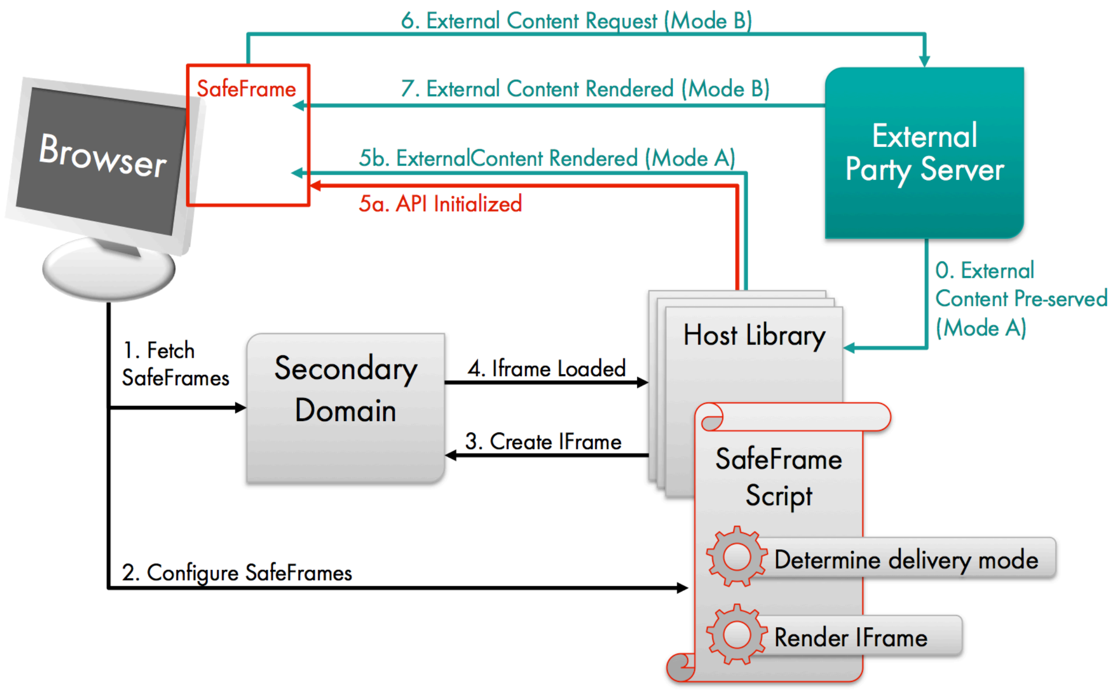
The following diagram illustrates the process for delivering external content into the SafeFrame container using delivery mode B.


External Party Server
5. External Content Delivered
4. External Content Request
3. Browser Processes SafeFrames
Browser
1. Content Request
Host Server
Host Content
SafeFrame
External Content URL
2. Host Content Delivered
(with SafeFrames Instructions)

1. **Content Request:** When an end user visits the host's website, the browser requests content from the host's server.
2. **Host Content Delivered:** The host delivers HTML content for its site along with SafeFrame instructions and a URL to the external content.
3. **Browser Processes SafeFrame:** The browser uses SafeFrame instructions from the delivered host content to process interactions between host content and the external content.
4. **External Request:** The browser requests content from the external server using the URL that the host provided.
5. **External content Delivered:** The requested external content is delivered directly into the SafeFrame iframe.

2.1.3 Rendering the SafeFrame

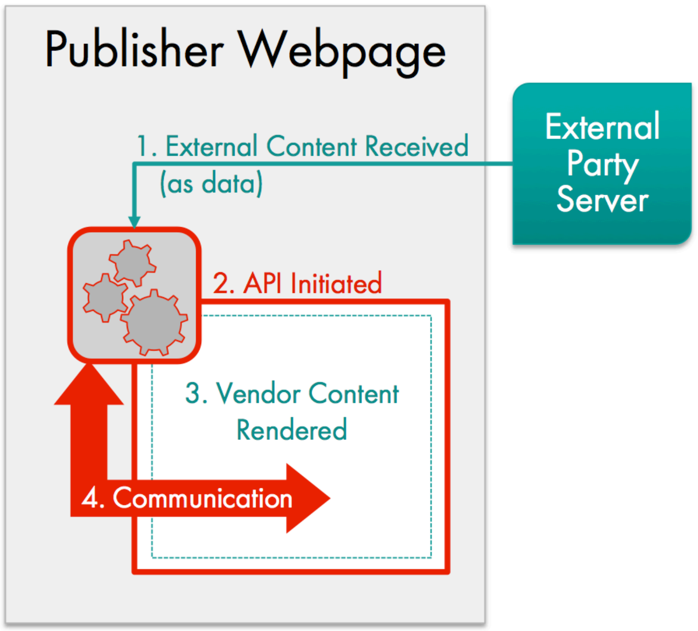
The figures in sections 2.1.1 and 2.1.2 illustrate the difference between delivery mode A and delivery mode B, respectively. The figure that follows describes at a high-level, how the browser uses SafeFrame instructions sent from the host server to initialize the SafeFrame API and render the external content within it.



1. **Fetch SafeFrame:** After receiving instructions from the host server, the browser requests and receives SafeFrame from a secondary domain.
2. **Configure SafeFrame:** The browser initiates SafeFrame code accessed from the host library. Using the SafeFrame class `$sf.host.Position`, the delivery mode is identified because either the HTML external content is either included (delivery mode A) or a URL is referenced instead (delivery mode B). The SafeFrame function `$sf.host.render()` is then used to render the iframe.
3. **Create iframe:** The iframe is then created in the secondary domain. If external content is delivered using delivery mode A, the content data will be loaded with the iframe.
4. **iframe Loaded:** The iframe is loaded (with external content if delivered using delivery mode B) into the host library.
5. **API Initialized:** The SafeFrame API is initialized and external content is rendered if delivered using delivery mode A.
6. **External Content Request (Mode B):** If external content is being delivered using delivery mode B, then the URL supplied within the SafeFrame is used to request content from the external server.
7. **External content Rendered (Mode B):** External content is delivered and rendered directly in the SafeFrame iframe.

2.1.4 In-Page Communication with the API

The following diagram illustrates how communication is initiated between host and external party content.



1. **External Content Received (as data):** The host receives external party data for the content to be rendered in the host's Webpage.
2. **API Initiated:** Using the SafeFrame class `$sf.host.Position`, the Webpage creates a container for the external data received and then configures the SafeFrame host API.
3. **External Content Rendered:** External party data is rendered as content within the SafeFrame.
4. **Communication:** Once the external content is rendered it can use SafeFrame external API code, if implemented, to make calls to the host API.

2.2 Requirements

The following sections describe the requirements for implementing SafeFrame.

2.2.1 JavaScript Host Library and API

The JavaScript host library and API is used to control and render SafeFrame containers. Namespaces, classes, and functions are provided in this library and further described in section 4.

Host Implementation Note	A SafeFrame may include: a WebView (mobile), an embedded browser, an HTA (Microsoft HTML Application), or a raw consumer web browser.

2.2.2 Secondary Host Name

The SafeFrame host API creates, renders and manages external party HTML content by creating an iframe that has a different-origin, hostname, and domain than the host Webpage. This secondary host name creates a "cross-domain-barrier," preventing external party HTML and JavaScript from directly accessing anything in the host Webpage.

Web browsers follow the "same-origin" policy where code from 2 different origins is not allowed to communicate (with certain exceptions). Therefore the host MUST have a secondary host name, from where the SafeFrame resources can be served. Secondary host names can be provided using a CDN (Content Delivery Network).

2.2.3 Resource conventions

For the host, SafeFrame defines 2 types of resources: base-rendering files and JavaScript files.

Base HTML Files (static HTML Files)

HTML Files are used to provide a base-level HTML document into which external party HTML content is rendered. In cases where HTML5 messaging is unsupported, an HTML file may also be used to act as a proxy to facilitate sending messaging between the host and external party.

HTML files used with SafeFrame must adhere to the following conventions:

- Browsers and proxies must also be able to publicly cache the HTML files.
- More than one HTML file may be used for rendering to provide additional functionality.
- All rendering HTML files MUST include support for external party API functionality in order to render external content.
- HTML files used for rendering must first include the SafeFrame external party JavaScript library.

JavaScript Files

The Host SafeFrame API is implemented with JavaScript and must adhere to the following conventions:

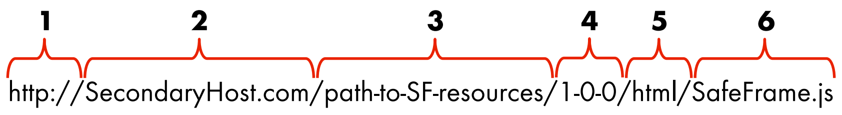
- SafeFrame API JavaScript files MUST be static.
- The external party API is ALWAYS implemented in JavaScript.
- Browsers and proxies must also be allowed to publicly cache all files provided.

2.2.4 URI conventions for SafeFrame

Since browsers and proxies must be able to cache JavaScript and HTML resource files, these files should only change with the release of a new version and URIs for accessing the files cannot contain query-string parameters or anything that would make a browser or proxy not cache the file.

Version consistency must be maintained. For example, HTML resources for version 2-2-0 shouldn't be served to a JavaScript implementation of version 2-3-5. Version consistency must be maintained. For example, HTML resources for version 2-2-0 shouldn't be served to a JavaScript implementation of version 2-3-5.

The following URI format for accessing host resources for SafeFrame enable the use of static URIs and relative paths within resources.


1
2
3
4
5
6
http://SecondaryHost.com/path-to-SF-resources/1-0-0/html/SafeFrame.js

The sections of a URI used to access SafeFrame resources are defined entirely by the host but must be provided in the order specified and are described in detail below:

1. The protocol (i.e. http, https, etc.)
2. Secondary Host Name (and port if applicable)
3. Root path to SafeFrame resource (multiple directories allowed, seperated by /)
4. SafeFrame version number in the format "n-n-n"
5. Lowercase "html" for HTML files or lowercase "js" for JavaScript files (other resource types may be included as a future SafeFrame release)
6. File name

2.3 Implementation Notes

The following notes are important for the host implementation of SafeFrame.

SafeFrame and iframe Nesting

SafeFrame containers are ALWAYS rendered in the top-level HTML document. A SafeFrame container cannot be rendered inside another SafeFrame container. The iframe that is created by the SafeFrame Host API can only communicate with external HTML content if it is implemented as part of the host's top-layer HTML that the browser executes. SafeFrame JavaScript code must be able to detect improper nesting.

Given the following improper nesting cases, SafeFrame JavaScript code should take the listed actions.

1. SafeFrame host JavaScript code loaded into an iframe:
   a. SafeFrame host JavaScript namespaces are nullified.
   b. One and only one call to Function $sf.host.boot is allowed so that the Class $sf.host.Position object's content can be rendered.
   c. SafeFrame JavaScript code will NOT respect or process configuration calls (see Class $sf.host.Config, and Class $sf.host.PosConfig).
   d. SafeFrame JavaScript code will not respect or process metadata defined in a SafeFrame tag that is within an iframe.
2. SafeFrame tags (see section 3) placed inside an iframe:
   a. SafeFrame JavaScript code will parse and read the SafeFrame tags to retrieve content.
   b. SafeFrame JavaScript code will render the content.
   c. SafeFrame JavaScript code will NOT respect or process configuration calls (see Class $sf.host.Config, and Class $sf.host.PosConfig).
   d. SafeFrame JavaScript code will not respect or process metadata defined in a SafeFrame tag that is within an iframe.
   e. If the iframe provided is a SafeFrame container, then content is rendered with its own JavaScript code and can access the external party SafeFrame API.

The SafeFrame JavaScript Library

SafeFrame containers ALWAYS contain a JavaScript library which responds to the SafeFrame host JavaScript library and is ALWAYS the first JavaScript file included in an HTML rendering file.

SafeFrame HTML Management

SafeFrame manages ALL HTML elements it creates. Neither the host nor the external party should attempt to manipulate HTML nodes that the SafeFrame creates and manages. Doing so results in unexpected behaviors that can break the interface.

SafeFrame may add a CSS class attribute to HTML elements that it controls, but the host and external party should refrain from adding CSS rules or selectors for the following CSS class names:

- **sf_data:** used to denote inline SafeFrame elements which contain data to be rendered
- **sf_position:** used to denote the starting HTML element tree of a rendered SafeFrame container
- **sf_lib:** used to denote SCRIPT HTML elements that contain SafeFrame JavaScript code
- **sf_el:** generic used to denote other HTML elements maintained by SafeFrame JavaScript code

Handing content to the SafeFrame Host API for rendering

Besides configuring external content for delivery into the SafeFrame container, the host should verify the content is in an acceptable form for the API to process. The host page receives external content as either raw HTML or as a URL that the SafeFrame must fetch and render on its own.

In either case, SafeFrame is using JavaScript to process the information. Raw HTML content may need to be encoded before sending to SafeFrame. While no adjustments are needed for a URL, the external party should be notified that a JavaScript response is necessary for rendering and adding content into the SafeFrame container.

The following implementation notes provide more detail and correspond to the two delivery modes discussed in section 2.1.

Raw HTML as JavaScript String (delivery mode A)

Since JavaScript handles certain characters and SCRIPT tags differently than HTML does, the host may have to modify any raw HTML content before sending it to be processed in the SafeFrame API.

For example, the following HTML string produces a syntax error when processed as JavaScript:

```
<script type="text/javascript"> document.write('Hello "Dave"');
</script>
```

To enable the above HTML string to be process in the SafeFrame API, it must be reformatted as follows:

```
var html = "<scr"+"ipt type=\"text\/javascript\">
document.write('Hello \"Dave\"'); </scr"+"ipt>";
```

The above JavaScript-formatted HTML string can be processed in the SafeFrame `$sf.host.Position` object.

Using a URL to fetch 3rd party content (delivery mode B)

The host may provide a URL to the external content instead of providing the HTML content itself. In this case, the host page cannot render the external content directly into the SafeFrame `$sf.host.Position` object, leaving it to SafeFrame to fetch and render the content in container.

Since the host page cannot encode the external content for processing in SafeFrame, the response from the request that SafeFrame makes should be in JavaScript format. Once the SafeFrame container is generated, a SCRIPT tag response from the external content URL is generated and can deliver additional content. A SCRIPT tag is used so that content delivered from the provided URL can access the External Party API within the SafeFrame container, instead of using a nested iframe tag.

External Party Implementation Note	While external party HTML content may use additional iframe tags, any content within extra iframe tags is prevented from accessing the API because of the "same-origin" policy. See section 3 on SafeFrame tags for more information.

2.4 SafeFrame Rendering Details

The Host JavaScript library inserts an HTML iframe element into the host page. The Host API controls the rendering process in order to also control the external content to be rendered in the iframe.

The iframe is created with the following guidelines.

The `src` Attibute

The "src" attribute of the iframe is a URL that points to a static, publically cacheable HTML file. The URL provided must be from an origin different than the host domain, which is typically a content delivery network (CDN).

The `name` Attribute

The "name" attribute of the iframe must contain a serialized string of data, containing configuration attributes of: a particular SafeFrame position configuration, metadata, and the content to be rendered.

Using a data string in the `name` attribute enables one-way, synchronous message passing such that JavaScript code inside the HTML file can: read the corresponding `window.name` property, de-serialize the string of data, setup the environment for the external party content, and finally render the content.

This technique allows for HTML SCRIPT tags that contain JavaScript `document.write` commands to work properly.

Width and Height Properties

The width and height of the iframe needs to be set to the same values given in the `w` and `h` fields of the `$sf.host.PosConfig` object. Typically the width and height of the iframe should match the known width and height of the content to be rendered.

The SCRIPT Tag

A SCRIPT tag should exist inside the HTML document within the created iframe. This SCRIPT tag is the first, initial JavaScript within the HTML document that reads in the data to be processed and rendered. A relative URL to access this JavaScirpt file may be used as long as version consistency is maintained (see section 2.2.4 for details on URL conventions in SafeFrame).

This JavaScript file must do the following (in order):

1. Check to see that the HTML document is a directly within (is a child of) the top level HTML document. If it is not, an error should be produced and no content should be rendered.
2. Read and de-serialize data passed in the `window.name` property.
3. Validate the data passed in to the `window.name` property, which is usually done by making sure the de-serialized object contains all required information, including a GUID. If validation fails, an error should be produced and no content should be rendered.
4. Set the `window.name` property to an empty string (`""`) so that the external party cannot read data from this point on.
5. Initialize the ability to send cross-domain messages up to the host SafeFrame JavaScript.
6. Attach any additional markup and metadata passed in the `name` attribute.
7. Attach any proprietary logic, event handlers, or other details to the HTML document (such as `onload` events).
8. Render external party content.

The HTML file should contain the following:

- CSS rules that set the margin and padding of the `BODY` element in the document to 0px.
- A single, absolutely positioned `DIV` element, as a direct descendant of the `BODY` element, initially positioned at top 0, left 0. This element is used in the cases where the external party content wishes to expand in a given direction so that proper alignment can be maintained.
- A single, `SCRIPT` element, placed within the given `DIV` element, which contains the logic in section 4 above. The SCRIPT may be from an external source or may be defined in-line.

  Example:

```
<html>
   <head>
     <style type=”text/css”>
         BODY { margin:0px; padding:0px }
     </style>
      </head>
      <body scroll=”no”>
          <div id=”sf_align”
style=”position:absolute;top:0px;left:0px;” class=”sf_el”>
            <script type=”text/javascript” src=”../js/ext.js”
class=”sf_lib”></script>
      </div>
   </body>
</html>
```

2.5 Communication Mechanism Details

In order to facilitate high-performing, secure communication between the host and the external party within a SafeFrame container, HTML5's "postMessage" function is the primary method used. This method allows an implementer to send a string from an HTML document of one origin to an HTML document of a different origin, thus getting around the "same-origin" policy. External content is prevented from calling this same method with the intent to trick the Host API into doing something wrong or malicious because the message (string sent) is validated.

For more information on the W3C same-origin policy, please visit:
http://en.wikipedia.org/wiki/Same_origin_policy

Whenever a message is received from a container, the following steps are taken to make sure that it is allowed:

1. **Secondary Domain / Origin Checking**
   The origin domain for a message sent from external HTML content should match the domain of the URL passed in the `renderFile` field of the `$sf.host.Config` class used to create the SafeFrame container. Should the origins not match the message is ignored.
2. **GUID Checking**
   A GUID is defined when the SafeFrame container is rendered and should be included with any messages the external party API sends. When the GUID is either absent or unknown when provided, the message is ignored.
3. **HTML window object reference Checking**
   The window reference source of the external party object should point to an iframe window reference created when the SafeFrame was rendered. If the window reference of the object does not match any known SafeFrame container that has been rendered, the message is ignored.
4. **Sequential Message Handling**
   Messages are handled on a first come, first served basis and a response denoting success or failure should always be attempted.

3 SafeFrame Tags

The high-level goal for a SafeFrame tag is to encapsulate external content as data so that the host can manage content rendering and control. The following sections describe SafeFrame tags, how to structure them, and how to process them.

3.1 SafeFrame Tags Structure & Requirements

A SafeFrame tag is a standardized set of HTML tags. It must be constructed with the following elements:

- SCRIPT tag provided inline with the host configuration and containing the external content metadata.
- JavaScript in the SCRIPT tag for processing the data tag(s)
- (Optional) NOSCRIPT tag for fallback on HTML when JavaScript is unsupported
- (Optional) DIV tag to specify where the SafeFrame container will be rendered

Only the primary host should insert the SafeFrame tag within page content. Nested SafeFrame tags are not supported. Any SafeFrame tags that are included in tags from exchanges, intermediaries, proxies, or any other secondary publishing partner or vendor are ignored.

If a SafeFrame tag is rendered within a SafeFrame container that has already been created, the rendering process assumes that the container has already been created and skips over to rendering the external content. Using only the SafeFrame container directly on the host Webpage ensures that content is rendered properly, the data is shared, and the API can be accessed.

3.1.1 The SCRIPT Tag

The SCRIPT tag must be specifically constructed inline with the host content and include the data tag for the external party content to be rendered in the SafeFrame.

The SCRIPT tag must contain the following:

- A class attribute with the value of iab_sf_data
- The type attribute set to `text/x-safeframe`
- A JavaScript or JSON-like data structure that is a representation of both `$sf.host.Position` and `$sf.host.PosConfig`. This structure is defined in literal JavaScript syntax and may include additional metadata values that are converted into an `$sf.host.PosMeta` object.

Example:

```
<script type='text/x-safeframe' class='iab_sf_data'>
      {
      id: "LREC", // ID of position object
      html: "<h1>Hello World</h1>", //3rd party HTML content
      conf:
            {
            size: "300x250" //The size conf is required and denotes the
            }
            meta: //optional shared meta information
            {
                  rmx:
                  {
                        sectionID: "14800347",
                        siteID: "140509"
                  }
            }
      }
</script>
```

The attributes of the SCRIPT tag above are defined as such for the following reasons:

type="text/x-safeframe"
Since the data is an inline JavaScript structure, issues could occur in the syntax and possibly break the host web page if treated as JavaScript code. Setting script type to "text/x-safeframe" ensures that the data structure is not picked up by the browser's JavaScript engine.

class="sf_data"
In a SCRIPT tag, the class attribute is not processed; however, specifying the sf_data class name enables the SafeFrame host API to identify data tags that it can then process.

3.1.2 Using JavaScript to Process Data Tags

The SCRIPT tag must include the JavaScript code that will process the data tag(s). Sections 3.1.2.1 to 3.1.2.3 provide examples for three different ways to process data tags.

- Example 3.1.2.1: Process All Tags at Once
- Example 3.1.2.2: Define the Library First
- Example 3.1.2.3: Process Tags One at Time

3.1.2.1 Example: Process All Tags at Once

When the data tags are included in the SCRIPT tag before the SafeFrame host library along with the `$sf.host.boot` method, the host library is loaded and the bootstrapping feature finds and renders all the data tags listed above the bootstrapping feature in the code.

The following example provides two SafeFrame data tags followed by instructions to load the SafeFrame host library and bootstrapping feature.

```
<table>
      <tbody>
            <tr>
                  <td valign='top'>

                  <-- SafeFrame Inline Tag 1 -->
                        <div id='tgtLREC'>
                              <script type='text/x-safeframe'
                              class='iab_sf_data'> {
                                    id: "LREC",
                                    src:
                                    "http://extserver.com/data-tag",
                                    conf:
                                    {
                                          w: 300,
                                          h: 250,
                                          dest: "tgtLREC"
                                    },
                                    meta:
                                    {
                                          rmx:

                                          {
                                                sectionID: "14800347",
                                                siteID:  "140509"
                                          }
                                    }
                              }
                              </script>
                              <-- b/c a "dest" tag exists (the overall
                              div container) container tags will be
                              rendered here -->

                              <-- optional noscript section for fall
                              back -->
                              <noscript>
                                    <img src=
                                    "http://ext.server.com/img.gif"
                                    />
                              </noscript>
                        </div>
                  </td>
                  <td valign='top'>

                        <-- SafeFrame Inline Tag 2 -->
                        <script type='text/x-safeframe'
                        class='iab_sf_data'>
                        {
                              id: "LREC2",
                              src: "http://externalserver.com/data-
                        tag",
                              conf:
                              {
                                    w: 300,
                                    h: 250
                              }
                        }
                        </script>
                        <-- b/c a "dest" tag exists (the overall div
                        container) container tags will be rendered here
                        -->
                        <-- optional noscript section for fall back -->
                        <noscript>
                              <img src=
                              "http://ext.server.com/img.gif"
                              />
                        </noscript>
                  </td>
            </tr>
      </tbody>
</table>
<-- SafeFrame Host library / API -->
<script type='text/javascript' src='sf-api-boot.js'></script>
<-- script code in external file will automatically 'boot' and read
data tags -->
```

3.1.2.2 Example: SafeFrame host library before data tags

Another way to implement a SafeFrame tag is to first define the host library, then provide the data tags, and finally call `$sf.host.boot` explicitly.

The following example demonstrates how this scenario might be coded.

```
<html>
      <head>
<script type="text/javascript" src="http://cdn.example.org/v1/sf-
host.js"></script></script>
            <script type='text/javascript'>

                  <-- SafeFrame Host API configuration -->
                  (function()
                        {
                        var pubAPI = $sf.hostpub, conf;
                        function handle_start_pos_render(id)
                        {

                        }
                        function handle_end_pos_render(id)
                        {
                        }
                        conf = new pubAPI.Config(
                        {
                              auto: true,
                              cdn: "http://l.yimg.com",
                              renderFile: "r.html",
                              root: "/SafeFrame/v1/html",
                              ver: "2-3-4",
                              positions:
                              {
                                    "LREC":
                                    {
                                          dest:    "tgtLREC",
                                          w: 300,
                                          h: 250
                                    }
                              },
                              onStartPosRender:
                              handle_start_pos_render,
                              onEndPosRender: handle_end_pos_render
                        });
                  })();
            </script>
      </head>
      <body>
            <div id='tgtLREC'>
                  <script type='text/x-safeframe' class='sf_data'>
                        {
                              id: "LREC",
                              src:
                              "http://externalserver.com/data-tag",
                              meta:
                              {
                                    rmx:
                                    {
                                          sectionID: "14800347",
                                          siteID: "140509"
                                    }
                              }
                        }
                  </script>
```

```
                 <-- b/c a "dest" tag exists (the overall div
                container) container tags will be rendered here -->
                <noscript>
                      <img src= "http://ext.server.com/img.gif"
                      />
                </noscript>
          </div>
          <script type='text/javascript'>
                $sf.host.boot();
          </script>
    </body>
</html>
```

3.1.2.3 Example: SafeFrame Data Tag with Sibling Auto-Bootstrapping

In the following example, each data tag submitted is accompanied by a secondary tag that calls `$sf.host.boot` to load the tag listed above it in the code.

```
<table>
      <tbody>
            <tr>
                  <td valign='top'>
                        <-- SafeFrame Inline Tag 1 -->
                        <div id='tgtLREC'>
                              <script type='text/x-safeframe'
                              class='sf_data'>
                                    {
                                          id: "LREC",
                                          src:
                                          "http://externalserver.com/da
                                          ta-tag",
                                          conf:
                                          {
                                                w: 300,
                                                h: 250,
                                                dest: "tgtLREC"
                                          },
                                          meta:
                                          {
                                                rmx:
                                                {
                                                      sectionID:
                                                "14800347",
                                                      siteID: "140509"
                                                }
                                          }
                                    }
                              </script>
                              <-- b/c a "dest" tag exists (the overall
                              div container) container tags will be
                              rendered here -->
                              <-- optional noscript section for fall
                              back -->
                              <noscript>
                                    <img src=
                                    "http://ext.server.com/img.gif"
                                    />
                              </noscript>
                              <script type='text/javascript'>
                                    (function() {
                                          var w = window, s = w["$sf"],
                                          b = s && s.boot;
                                          if (!s) s = w["$sf"] = {};
                                          if (b && typeof b ==
                                          "function") {
                                                try { b(); } catch (e)
                                                { }
                                          } else {
                                                document.write("<scr","
                                                ipt
                                                type='text/javascript'
                                                src='sf-
                                                host.js'></scr","ipt>")
                                                ;
                                          }
                                    })();
                              </script>
                              <-- Above script code will only load in
                              host library one time, call boot for each
                              tag -->
                        </div>
                  </td>
                  <td valign='top'>
                        <-- SafeFrame Inline Tag 2 -->
                        <script type='text/x-safeframe'
                        class='sf_data'>
                              {
                                    id: "LREC2",
                                    src:
                                    "http://externalserver.com/data-
                                    tag",
                                    conf:
                                    {
                                          w: 300,
                                          h: 250
                                    }
                              }
                        </script>
                        <-- b/c a "dest" tag exists (the overall div
                        container) container tags will be rendered here
                        -->
                        <-- optional noscript section for fall back -->
                        <noscript>
                              <img src= "http://ext.server.com/img.gif
                              />
                        </noscript>

                        <script type='text/javascript'>
                              (function() {
                                          var w = window, s = w["$sf"],
                                          b = s && s.boot;
                                          if (!s) s = w["$sf"] = {};
                                          if (b && typeof b ==
                                          "function") {
```

```
                                          try { b(); } catch (e)
                                          { }
                                    } else {
                                          document.write("<scr","
                                          ipt
                                          type='text/javascript'
                                          src='sf-
                                          host.js'></scr","ipt>")
                                          ;
                                    }

                              })();
                        </script>
                        <-- Above script code will only load in host
                        library one time, call boot for each tag -->
                  </td>
            </tr>
      </tbody>
</table>
```

4 Host API Implementation Details

The SafeFrame host API uses the namespaces, functions, and classes defined in sections 4.1 to 4.11.

4.1 Namespace $sf.host

This namespace is used to define the JavaScript classes, objects, and methods that the host Webpage can use for interaction with the SafeFrame container.

The `$sf.host` namespace is SafeFrame initiation point for configuring, rendering, inspecting, and interacting with a SafeFrame container. Everything defined in this space is public unless otherwise called out.

4.2 Namespace $sf.host.conf

Specifying the host conf namespace inline will allow SafeFrame containers (including SafeFrame tags) to be loaded (or bootstrapped) with configuration options you specify. This object is a literal version of the `$sf.host.Config` object.

Related Sections

3 SafeFrame Tags

4.4 Class $sf.host.Config

Example 1

```
<script type='text/javascript'>

//JavaScript inline host config, used mainly for SafeFrame tags which want to
auto boot the SafeFrame host API and render 3rd party content.

       var w = window, sf = w["$sf"], pub = sf && sf.host;
       if (!sf) sf = w["$sf"] = {};
       if (!pub) pub = sf.host = {};

       host.conf =
       {
              debug:     true,
              ver:       "2-3-4",
              positions:
              {
                     LREC:
                     {
                            id:   "LREC",
                            dest: "tgtLREC",
                            tgt:  "_self",
                            w:    300,
                            h:    250
                     }
              }
       };

       //Assuming a SafeFrame tag is placed below this configuration, it will
       read the config defined and use those values as the logic for the tag.
</script>
```

Example 2

```
<-- SafeFrame Inline Tag -->
<div id="tgtLREC">
       <script type='text/x-safeframe' class='sf_data>
              {
                  id:        "LREC",
                  src:"http://ext.server.com/sf",
                  conf:
                  {
                     dest:   "tgtLREC",
                     size:   "300x250"
                  }

                  meta:
                  {
                     rmx:
                     {
                            sectionID:    "14800347",
                            siteID:   "140509"
                     }
                  }
                }
       </script>
       <script type='text/javascript'>
              try {
                  $sf.host.boot();
              } catch (e) {  }
       </script>
</div>
```

4.3 Namespace $sf.info

The info namespace is reserved for storing information about SafeFrame containers.

> `<static> {Array} $sf.info.errs`
>
> Contains information about any errors that occur in the host side of the SafeFrame API; details are ***Read-Only***
>
> `<static> {Array} $sf.info.list`
>
> Contains information about each SafeFrame container that is either: to be rendered, is rendered, or is being processed; details are ***Read-Only***

Whenever the SafeFrame host API creates a container, it updates these namespace fields appropriately, allowing for inspection and/or debugging of the current state.

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>

(function() {

      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
&& host.Config,

      CONF_CDN    = "http://l.yimg.com",
      CONF_ROOT   = "/sf",
      CONF_VER = "2-3-4",
      CONF_RFILE  = "/html/render.html",
      CONF_TO  = 30;

      function on_endposrender(posID, success)
      {
            //a render action success
      }

      function on_posmsg(posID, msg, data)
      {
            //listen for messages
      }

      w.render_content    = function()
      {
            var conf, posConf, pos,confDesc;

            if (Config) {
                  conf = Config();
                  if (!conf) {
                        confDesc =
                        {
                              debug:            true,
                              cdn:              CONF_CDN,
                              root:             CONF_ROOT,
                              ver:              CONF_VER,
                              renderFile:           CONF_RFILE,
                              to:               CONF_TO
                              onEndPosRender:       on_endposrender,
                              onPosMsg:         on_posmsg
                        };
                        conf = new Config(confDesc);
                  }
                  if (conf) {
                        posConf = new host.PosConfig("LREC","tgtLREC");
                        posConf.w   = 300;
                        posConf.h   = 250;
                        posConf.z   = 1000;
                        pos      = new host.Position("LREC","<h1>Hello
                            World I'm an Ad<h1>",null,posConf);
                        host.render(pos);
                  }
            }
      }
      w.remove_content    = function()
      {
            var skipID = "LREC",  // we want to skip the LREC position,
      and leave it in the page
                  list   = $sf.info.list,
                  cnt    = list.length,
                  to_rem = [],
                  idx    = 0,
                  pos;
            while (cnt--)
            {
                  pos = list[idx++];  //$sf.host.Position
                  if (pos.id == skipID) continue;
                  to_rem.push(pos.id);
            }
      $host.nuke(to_rem); //remove all but the LREC position; } })();
</script>
```

4.4 Class $sf.host.Config

`$sf.host.Config(conf)`

The host config class is used to describe the configuration options for the SafeFrame Host API. This class configures the overall features and settings used by the Host.

Host Implementation Note	The host class `$sf.host.Config` should only exist once in the SafeFrame host API and should be constructed to initiate configuration options at a time when SafeFrame containers are inactive. When constructed, details will write to the inline `$sf.host.conf` namespace if not previously defined.

If no initial argument is specified, existing configuration is returned. If return value is null, then no valid configuration exists.

When `$sf.host.Config` is constructed, other SafeFrame host classes read from the resulting configuration to determine how containers are rendered. Any resulting values are added to the inline `$sf.host.conf` namespace if not previously defined.

Parameter

`{Object}` **conf**

A list of key value pairs to use for the configuration.

Fields

The following fields may be returned in the `conf` parameter:

`{String}` **conf.cdn**

Host of the CDN used to fetch SafeFrame resources. This value should always be a different domain than your web page

Sample value: "`http://l.yimg.com`"

`{String}` **conf.ver**

The version number of the SafeFrame to be used, provided in the format [number]-[number]-[number].

Sample value: `"2-3-4"`

`{String}` **conf.renderFile**

The partial path and filename of the file from the `cdn` property that is used as the base document for external party content to be rendered using the SafeFrame.

`{String}` **conf.hostFile**

The URL string to the Host-side JavaScript file to be used.

`{String}` **extFile**

The URL string to the External Party-side JavaScript file to be used.

`{String}` **bootFile**

The URL string to the External Party-side JaveScript file to be used for bootstrapping the SafeFrames library, processing SafeFrames tags, and rendering content.

`{Number}` **conf.to**

The maximum amount of time (in seconds) that a render process can take before the operation can be aborted.

Rendering the external party content in a SafeFrame container is an asynchronous process, which is done by rendering an x-domain iframe tag. This number defines the maximum amount of time that the render operation can spend in the "loading" state before a time-out error is generated.

Sample value: `30`

`{Object}` **conf.positions**

An object defining literal representations of `$sf.host.PosConfig` objects, keyed by id, to be used to configure each position in the page

`{Boolean}` **conf.auto** *(Optional)*

Whether or not automatic bootstrapping and rendering of SafeFrame tags should occur. Default is true. If set to false, SafeFrame tags will just add to the `$sf.info` object.

`{String}` **conf.msgFile** *(Optional)*

The partial path and filename of the file from the `cdn` property that is used to as a proxy for x-domain communication. Only required for older browsers that do not support HTML 5.

`{Boolean}` **conf.debug** *(Optional)*

Whether or not to run the SDK in debug mode, which will also use un-minified JS code, separated files, etc.

Events

`onBeforePosMsg(id, msgName, data)`

A function that gets called each time a position sends a request for some functionality. Returning `true` cancels the command request.

Parameters:

`{String}` **id**

The id of the position that has started its render process

`{String}` **msgName**

The type of message being sent

`{String}` **data** *(Optional)*

Data that gets passed through

`onEndPosRender(id)`

A function which gets called each time a position has finished rendering

Parameters**:**

`{String}` **id**

The id of the position that has started its render process

`onFailure(id)`

A function which gets called anytime a render call has failed or timed out

Parameters**:**

`{String}` **id**

The id of the position that has started its render process `onPosMsg(id, msgName, data)`

A callback function which gets called each time a position sends a message up to your web page

Parameters:

`{String}` **id**

The id of the position that has started its render process

`{String}` **msgName**

The name / type of message being sent

`{String}` **data** *(Optional)*

Data that gets passed through

`onStartPosRender(id)`

A callback function which gets called each time a position is about to be rendered

Parameters:

`{String}` **id**

The id of the position that has started its render process

`onSuccess(id)`

A callback function which gets called anytime a render call has successfully completed.

Parameters:

`{String}` **id**

The id of the position that has started its render process

Related Sections:

4.2 Namespace $sf.host.conf

4.5 Class $sf.host.PosConfig

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>

(function() {
      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
&& host.Config,

      CONF_CDN    = "http://l.yimg.com",
      CONF_ROOT   = "/sf",
      CONF_VER = "2-3-4",
      CONF_RFILE  = "/html/render.html",
      CONF_TO     = 30;

      function on_endposrender(posID, success)
      {
            //a render action success
      }

      function on_posmsg(posID, msg, data)
      {
            //listen for messages
      }

      w.init_SafeFrame    = function()

      {
            var conf, confDesc;

            if (Config) {
                  conf = Config();
                  if (!conf) {
                        confDesc =
                        {
                              debug:            true,
                              cdn:              CONF_CDN,
                              root:             CONF_ROOT,
                              ver:              CONF_VER,
                              renderFile:           CONF_RFILE,
                              to:               CONF_TO
                              onEndPosRender:       on_endposrender,
                              onPosMsg:         on_posmsg,
                              positions:
                              {
                                    "LREC":
                                    {
                                          id:       "LREC",
                                          w:        300,
                                          h:        250,
                                          z:        1000,
                                          dest:     "tgtLREC"
                                    }
                              }
                        };
                        conf = new Config(confDesc);
                        if (conf) {
                              alert("SafeFrame Host Config
                        successful");
                        }
                  }
            }
      }
})();
</script>
```

4.5 Class $sf.host.PosConfig

`$sf.host.PosConfig``(posIDorObj, destID, baseConf)`

The position configuration class describes how an `$sf.host.Position` object should be rendered. Only one `PosConfig` object per unique ID can exist. If more than one `PosConfig` object is constructed with the same id, previous values of the original PosConfig are overwritten. The host can still support multiple ad positions (i.e. two unique LRECs) with the same characteristics; they just need to have different IDs (i.e. LREC1 and LREC2)

The class construction can be thought of as a factory where, internally, all the instances constructed are monitored so that automatic linking with overall configuration options and data can occur.

Parameters:

`{String|Object}` **posIDorObj**
If this value is provided as a string, then it is used as the id property of the instance. If the value is returned as an object, then it is a descriptor that populates the properties of the instance.

`{String}` **destID**
The HTML element ID attribute string into which the content is to be rendered.

`{Object}` **baseConf,** (Optional)
An optional object that defines a representation of an `$sf.host.Config` object and is used in cases where no initial Host configuration was pre-defined. This option enables a shortcut for automatic host configuration if necessary and is usually used in conjunction with SafeFrame tags. If specified when a Host configuration already exists, this parameter is ignored.

Fields

`bg`
The background color to be used inside the safe frame. Default value is `"transparent"`.

`css`
Style-sheet text or a URI to a CSS file that defines additional CSS to be used inside the SafeFrame iframe. Default value is `""`.

`dest`
The HTML element ID into which the content is to be rendered.

`H`
The height (in pixels) of the SafeFrame iframe to be created for the content specified.

`id`
A unique identifier for the position or content. Used to link the `$sf.host.Position` object with a configuration. Specifying the `id` as `"DEFAULT"` means that this configuration will be used as the default values for other `$sf.Position` objects created.

`size`
A string representing the width and height (in pixels) of the safe frame to be created for the content specified. Setting this value also sets the `w` and `h` properties respectively Example: `"300x250"`

`tgt`
The target window name for where hyperlink clicks should be routed to unless otherwise specified. Default value is `"_blank"`. If a URL is provided, it opens in a new window. The values `"_self"` and `"_parent"` are NOT allowed and if provided the value `"_top"` is used instead.

`w`
The width (in pixels) of the SafeFrame iframe to be created for the content specified.

`z`
The z-index value to be used for the SafeFrame iframe.

`supports`
An object identifying the features that the host supports relative to the content specified.

Methods

`toString()`
A method that serializes the position into a string using query-string encoded syntax.

Example

```
//See $sf.host.Config example
```

4.6 Class $sf.host.Position

`$sf.host.Position``(posIDorObj, html, meta, config)`

A class used to describe the HTML content that is to be rendered inside a safe frame.

Parameters:

`{String|Object}` **posIDorObj**
REQUIRED, if is a string, used as the id property of the instance. If is an object, it is used as a descriptor to fill out the properties of the instance.

`{String}` **html**
REQUIRED, the string content to be rendered into the safe frame described by this instance

`{Object}` **meta Optional**
An object with key/value pairs defining customizable metadata about the position

`{Object}` **config Optional**
An object representing position config overrides

Fields:

`{Object}` **config**
Config information defines how SafeFrame renders a position. This object can override values already set in the associated config.

`{String}` **html**
The HTML content to be rendered inside the safe frame, or a URL to HTML content returned that is returned using a SCRIPT tag.

`{String}` **id**
A unique identifier for the position. If present, this value is used to lookup a `$sf.host.PosConfig` object.

`{Object}` **meta**
Metadata information in the form of an object of any number, combination key, or value pairs to store host or content-related metadata.

`{String}` **src**
A URI to be used as a SCRIPT tag that renders the contents in the SafeFrame. Setting this value changes the value of the HTML property and is used mostly for short-hand purposes.

The purpose of this field is to enable content to be fetched when the HTML content is no readily available. Setting this property creates an HTTP request for content to the URI specified. Because the URI provided is in a SCRIPT context, content must be returned in JavaScript form. This process prevents the creation of other iframes that would otherwise damage the system because content within any created iframes is denied access to the external content API.

The URI provided may contain MACRO place holders that SafeFrame will populate. This feature can be used to gather information from a Web browser that can be passed in the HTTP request and is useful for cases when retrieved content requires information about the Web browser environment only available to the host.

SafeFrame populates the following values:

`{String} ${sf_ver}`
The string representation of the current version of SafeFrame

`{Number} ${ck_on}`
Indicates whether cookies are enabled on the browser: 1 for true, 0 for false.

`{String} ${flash_ver}`
Identifies which version of Flash is enabled in the browser. If Flash is not detected, the value is set to 0.

Example

```
function define_content()
{
       var pub = $sf.host, PosConfig = host.PosConfig, PosMeta = host.PosMeta,
         Pos = host.Position, pos, posConf, posMeta;

    posConf     = new PosConfig("LREC", "tgtLREC");
    posConf.w   = 300;
    posConf.h   = 250;
    posConf.z   = 1000;

       posMeta      = new PosMeta({"context":"Music"});

    //a shared meta object will now contain
    //  context:    "Music"
    //  sf_ver:     "1-0-1",
    //  flash_ver:  11

    pos         = new Pos("LREC",
"http://getsomeads.com?pos=LREC&f=${flash_ver}&sf=${sf_ver}", posMeta,
posConf);
    //note that the ${flash_ver} and ${sf_ver} macros will get filled out
automatically
    //
    //so if flash 11 is installed, and we are using SafeFrame version 1
    //the URI for the script tag created will be
    //
    // "http://getsomeads.com?pos=LREC&f=11&sf=1-0-1"

    host.render(pos);
```

Method

`toString()`

A method that serializes the position into a string using query-string encoded syntax

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>

(function() {

      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
         && host.Config,

      CONF_CDN    = "http://l.yimg.com",
      CONF_ROOT   = "/sf",
      CONF_VER = "2-3-4",
      CONF_RFILE  = "/html/render.html",
      CONF_TO     = 30;

      function on_endposrender(posID, success)
      {
            //a render action success
      }

      function on_posmsg(posID, msg, data)
      {
       //listen for messages
      }

      w.init_render   = function()
      {
       var conf, confDesc, posConf, pos;

            if (Config) {
                  conf = Config();
                  if (!conf) {
                        confDesc =
                        {
                              debug:           true,
                              cdn:             CONF_CDN,
                              root:            CONF_ROOT,
                              ver:             CONF_VER,
                              renderFile:         CONF_RFILE,
                              to:              CONF_TO
                              onEndPosRender:     on_endposrender,
                              onPosMsg:        on_posmsg
                        };
                        conf = new Config(confDesc);
                        if (conf) {
                              posConf      = new
                        host.PosConfig("LREC","tgtLREC");
                              posConf.w = 300;
                              posConf.h = 250;
                              posConf.z = 1000;
```

```
                                        pos       = new
                                          host.Position("LREC","<h1>Hello World,
                                          I'm an Ad</h1>");
                                        //note that b/c you constructed a
                                          PosConfig object already with an id of
                                          "LREC", the configuration will be
                                          grabbed

                                        host.render(pos);
                                  }
                           }
                    }
             }
})();
</script>
```

4.7 Class $sf.host.PosMeta

`$sf.host.PosMeta``(shared_obj, ownerKey, obj)`

This class defines metadata for a particular position. Metadata can be shared, or keyed, to specific data owners (which allows for hiding if needed). Values stored in this object, cannot be changed; they are set when constructed and are read-only. Typically data stored in this object is used for proprietary purposes.

When a SafeFrame container is constructed and rendered, the information stored here will be available to the external party API. Shared and non-shared internal objects are created for cases where certain metadata needs to be protected. For example, the `ownerKey` property could be a signature generated from a server.

Inside the SafeFrame container, a function is used for accessing this metadata, so external parties cannot use iteration to discover it. In this case the signature used as the `ownerKey` could be used inside the container to access it, allowing access only to trusted parties.

Whenever a `$sf.host.PosMeta` object is constructed the following information will always appear by default in the "shared" section.

> `{String} sf_ver`
> The string representation of the current version of SafeFrame
>
> `{Number} ck_on`
> Identified whether cookies are enabled on the browser: 1 for true, 0 for false.
>
> `{String} flash_ver`
> Identifies which version of Flash is enabled in the browser. If Flash is not detected, the value is set to 0.

Also see the `$sf.host.Position` "src" property. The above values are defined when the `PosMeta` object is constructed and can be automatically passed on the URL for the "src" property as macro fields.

Parameters:

`{Object}` **shared_obj** (Optional)
An object containing key /value pairs for shared metadata

`{String}` **ownerKey** (Optional)
A key name to identify the owner or a particular set of metadata.

`{Object}` **obj** (Optional)
An object containing the key value pairs of metadata

See the related Function $sf.ext.meta section for details on passing metadata.

Method
`{String|Number|Boolean}` **`value`**`(propKey, ownerKey)`

A method retrieves a metadata value from this object.

Method Parameters:

`{String}` **propKey**
The name of the value to retrieve

`{String}` **ownerKey** (Optional)
The name of the owner key of the metadata value. By default, it is assumed to be shared, so nothing needs to be passed in unless looking for a specific proprietary value

Returns:
`{String|Number|Boolean}`

Example
```
<-- Host Side tags -->
<div id='tgtLREC'></div>
<script type='text/javascript'>

var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub &&
host.Config, conf, posConf, posMeta, shared, non_shared, pos;

if (Config) {

    conf = Config();
      if (!conf) conf = new
         Config({debug:true,cdn:"http://l.yimg.com",root:"/sf",ver:"2-
         3-4",renderFile:"/html/render.html",to:30})
    if (conf) {
        posConf    = new host.PosConfig("LREC","tgtLREC");
        posConf.w  = 300;
        posConf.h  = 250;
        posConf.z  = 1000;
        shared     = {"context": "Music"};
        non_shared = {spaceID: 90900909090, adID: 3423423432423};
             posMeta   = new host.PosMeta(shared,"y",non_shared); //Use a
                  signature for a key name (instead of "y"), if you don't
                  want 3rd parties accessing this data
```

```
            pos   = new host.Position("LREC","<Hello World I'm an
                 Ad>",posMeta,posConf);
        host.render(pos);
    }
}
</script>
```

```
<-- External Party tag -->
<script type='text/javascript'>

var w = window, sf = w["$sf"], ext = sf && sf.ext, cntxt = ext &&
ext.meta("context"), yspaceID = ext && ext.meta("spaceID","y");

alert(cntxt); //will say Music;

alert(yspaceID); //will say 90900909090
</script>
```

4.8 Function $sf.host.boot

The boot function is used to look for, process and automatically render data tags. It returns a Boolean response that indicates whether or not any new, unprocessed items have been found. Once processed, the resulting SafeFrame data is added to the `$sf.info`. And if the auto field is set to true in the `$sf.host.config` class, the boot function initiates the render process for content defined in the data.

Returns

`{Boolean}`
Indicates whether any new, unprocessed items have been found

Related Sections

3 SafeFrame Tags
4.3 Namespace $sf.info
4.2 Namespace $sf.host.conf

Example

```
<-- SafeFrame Inline Tag -->
<div id="tgtLREC">
      <script type='text/x-safeframe' class='sf_data'>
            {
               id: "LREC",
               src: "http://secondarydomain.com/safeframe",
               conf:
               {
                 dest: "tgtLREC",
                 size: "300x250"
               }
               meta:
               {
                 rmx:
                      {
                           sectionID:"14800347",
                              siteID: "140509"
                        }
                  }
            }
      </script>
      <script type='text/javascript'>
            try {
                  $sf.host.boot();
            } catch (e) {  }
      </script>
</div>
```

4.9 Function $sf.host.status

The status function is used to determine the status of positions. It returns a Boolean response that indicates whether any positions in the page are currently in the process of being rendered or if some other operation, such as expansion, is occurring.

Parameter

`{Object}` positions

The optional Object parameter offers an empty object reference that can be populated with one of a list of keys representing each `$sf.host.Position` object (using its id property) that SafeFrame is currently managing. The value for each key contains an object with a status code string representing the current state of the container. In this release, possible values are:

- **ready:** the container is available for rendering but has not yet been rendered
- **loading:** the container is currently in the process of being rendered
- **expanding:** the container is currently in the process of expanding
- **expanded:** the container is currently in expanded state
- **collapsing:** the container is currently in the process of collapsing
- **error:** the container has experienced an error that is preventing any interaction

Returns

`{Boolean}`

Indicates whether or not the SafeFrame SDK is busy with an operation where the configuration cannot be updated

Related Sections

5.1 Namespace $sf.ext

5.7 Function $sf.ext.status

Example
```
<script type='text/javascript'>
      var posDetail = {};
      var isBusy    = $sf.host.status(posDetail);
      var posID     = "";
      var posInfo, posInfoStatus, posInfoDesc, posIDProc;

      if (isBusy) {
            //Cannot change configuration while operations are ongoing,
              inspect object to determine what is going on

            for (posID in posDetail)
            {
                posInfo = posDetail[posID];
                //object has "status", "id", and "desc" properties

                posInfoStatus = posInfo.status;
                switch (posInfoStatus)
                {
                   case "expanding":
                         posIDProc = posID;
                   break;
                   case "collapsing":
                         posIDProc = posID;
                   break;
                   }
                   if (posIDProc) break;
            }
            if (posIDProc) alert(posIDProc + ", is " + posInfoStatus);
          }
</script>
```

4.10 Function $sf.host.nuke

The nuke function is used to dismantle SafeFrame container positions from the page. This function can be called even if interaction is currently pending or taking place and will abort outstanding operations or rendering.

The nuke function is provided to accommodate situations where the SafeFrame container position cannot be easily removed from the page under regular circumstances. For example, the nuke function may be used to remove the SafeFrame container position in a native app that has no page that can be closed like it would in a Web browser.

Nuke is not needed to load new content into an existing position. The render function handles setting new content positions.

Parameter

`{String|String[]}` **id**

The id of the position to be removed; use "*" to remove all positions.

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>

(function() {
      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
      && host.Config,

      CONF_CDN    = "http://l.yimg.com",
      CONF_ROOT   = "/sf",
      CONF_VER= "2-3-4",
      CONF_RFILE  = "/html/render.html",
      CONF_TO  = 30;

      function on_endposrender(posID, success)
      {
            //a render action total failure
      }

      function on_posmsg(posID, msg, data)
      {
          //listen for messages
      }
      w.render_content   = function()
      {
          var conf, posConf, pos,confDesc;

          if (Config) {
            conf = Config();
            if (!conf) {
                confDesc  =
                {
                  debug: true,
                  cdn:       CONF_CDN,
                  root:      CONF_ROOT,
                  ver:       CONF_VER,
                  renderFile:    CONF_RFILE,
                  to:    CONF_TO
                  onEndPosRender:   on_endposrender,
                  onPosMsg:      on_posmsg
                };
                conf = new Config(confDesc);
            }
            if (conf) {
              posConf    = new host.PosConfig("LREC","tgtLREC");
              posConf.w  = 300;
              posConf.h  = 250;
              posConf.z  = 1000;
              pos      = new host.Position("LREC","<h1>Hello World I'm
                         an Ad<h1>",null,posConf);
              host.render(pos);
            }
          }
      }

      w.remove_content   = function()
      {
          host.nuke("*"); //will remove all positions rendered or in
                            process of rendering.
                         //could also pass "LREC" in this case, or
                            "LREC","SKY" if "LREC" and "SKY" ads were
                            configured.
      }
})();
</script>
```

4.11 Function $sf.host.get

The get function is used to obtain a reference to a SafeFrame container position config. When one of the SafeFrame callback functions notifies the host code of an event, this function is used to get a reference to the PosConfig object associated with the position in question.

Parameter

`{String}` **id**

The id of the position to get.

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>

(function() {
      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
      && host.Config,

      // Configuration omitted for brevity

      function on_endposrender(posID, success)
      {
            var adPos = host.get(posID);
            if(!success) {
              host.nuke(posID);
            }
      }
</script>
```

4.12 Function $sf.host.render

The render function is used to render one or more SafeFrame positions.

You can pass in one or more `$sf.host.Position` objects (or representations of objects) to render a group of containers at one time. If you pass in callback functions to the `$sf.host.Config` class, you will see these callbacks called in the following order:

1. onStartPosRender
2. onEndPosRender (success / failure)
3. onBeforePosMsg (if ad sends commands such as for expansion etc, allows you to return true to reject the message)
4. onPosMsg (if ad sends commands such as for expansion, etc.)

Host Implementation Note	The `onEndPosRender` callback cannot initiate when `$sf.host.nuke` has been called for a position that is currently rendering.

Parameter

`{Object|Object[]|$sf.host.Position|$sf.host.Position[]}` **data**

A representation of a `$sf.host.Position` object to be rendered

Related Sections

4.4 Class $sf.host.Config

4.5 Class $sf.host.PosConfig

4.6 Class $sf.host.Position

Example

```
<div id='tgtLREC'></div>
<script type='text/javascript'>  (

function() {
      var w = window, sf = w["$sf"], pub = sf && sf.host, Config = pub
      && host.Config,

      CONF_CDN    = "http://l.yimg.com",
      CONF_ROOT   = "/sf",
      CONF_VER = "2-3-4",
      CONF_RFILE  = "/html/render.html",
      CONF_TO  = 30;

      function on_endposrender(posID, success)
      {
            //a render action success
      }

      function on_posmsg(posID, msg, data)
      {
            //listen for messages
      }
      w.render_content    = function()
      {
          var conf, posConf, pos,confDesc;

          if (Config) {
            conf = Config();
            if (!conf) {
                confDesc =
                {
                  debug:        true,
                  cdn:              CONF_CDN,
                  root:             CONF_ROOT,
                  ver:              CONF_VER,
                  renderFile:           CONF_RFILE,
                  to:           CONF_TO
                  onEndPosRender:       on_endposrender,
                  onPosMsg:         on_posmsg
                };
                conf = new Config(confDesc);
            }
            if (conf) {
                posConf      = new host.PosConfig("LREC","tgtLREC");
                posConf.w = 300;
                posConf.h = 250;
                posConf.z = 1000;
                pos      = new host.Position("LREC","<h1>Hello World
                        I'm an Ad<h1>",null,posConf);
                host.render(pos);
            }
          }
      }

      w.remove_content    = function()
      {
          host.nuke("*"); //will remove all positions rendered or in
                           process of rendering.
                              //could also pass "LREC" in this case, or
                                "LREC","SKY" if "LREC" and "SKY" ads
                                were configured.
      }
})();
</script>
```

5 External Party API Implementation

The SafeFrame external party API uses the namespace and functions described in sections 5.1 to 5.10.

5.1 Namespace $sf.ext

`$sf.ext`

The ext namespace provides a series of methods for retrieving various types of information regarding the container. The external party uses this namespace to define the JavaScript classes and objects that the external party creative can use for interacting with the host content in the context of a SafeFrame.

SafeFrame methods that are used to execute interactions are asynchronous so that any success or failure can only be determined using callbacks from the API. These methods also maintain their state, which means that they are protected against repeated calls.

For example:

- The call `$sf.ext.expand` is initiated.
- In the background, SafeFrame processes `$sf.ext.expand` and sends a message to the host.
- If `$sf.ext.expand` is called again, before the first call is processed, *it is considered an error* because only one command can be processed at a time.
- If the `$sf.ext.expand` callback function was provided using `$sf.ext.register`, then the function is called and, once processed, notice of success or failure is sent.
- After a success or failure result is produced, `$sf.ext.expand` can be called again.

Event

`<static>  $sf.ext.__status_update(status, data)`

This event provides the status of the external party creative content. Event that is fired from the external party SDK, for which you can register a call back via `$sf.ext.register`.

Implementation Note	The `$sf.ext.__status_update` namespace is merely implied and does not exist in the JavaScript hierarchy but is called out here to document the parameters that are possible when the function is called and submitted to `$sf.ext.register`.

The callback function is called with at least two parameters: first, a string that denotes the state change and second, a string denoting the command that generated the status update event, which is issued by the external party API-initiated command that generated the status update event. If the second parameter is an empty string, the implication is that the host has forced a status update, rather than the command being initiated by the external party API.

Event Parameters:

`{String}` **status**

The status code string notifying external content of container updates. The following status codes are available:

expanded
The container has been expanded.

collapsed
The container is in the default collapsed state.

failed
A command initiated by the external party API did not succeed.

geom-update
The container geometry information has changed. Sent for events such when the browser window is resized, parent container scrolls, or other geometric changes.

focus-change
The browser window / tab has become active ("focus"), or become in-active ("blur").

`{Object}` **data** *(Optional)*
Contains information about the original message or action requested of the Host or supplied by the host as a result of changes in the page. The following objects may be issued:

cmd
The original command sent with possible values such as: `exp-ovr`, `exp-push`, `read-cookie`, `write-cookie`, etc.

reason
Description information about whether the command succeeded or failed.

info
The information sent as part of the command echoed back to the caller, such as dimensions for expansion, the data to set for a cookie, etc.

Related Sections
5.2

Function $sf.ext.register
5.5 Function $sf.ext.expand

5.2 Function $sf.ext.register

`$sf.ext.register(initWidth, initHeight, cb)`

Availability: Synchronous (can be requested at anytime)

The external party register function registers the SafeFrame platform to accept SafeFrame external party API calls. External party creative declares the initial (collapsed) width and height. Besides width and height, this function can also define a callback function, which informs the external content about various status details.

The initial width and height parameters are required in order for SafeFrame to notify the host of the display space needed to render the external content. The callback is a method that returns a success or error code for every command processed, notifying the external party of execution status for every command sent. The external party can then react accordingly. Commands should only be called once while waiting for success or failure notification. Any subsequent calls made before success or failure notification will be ignored.

Parameters

`{Number}`**initWidth**
The initial / original width of the 3rd party content

`{Number}` **initHeight**
The initial / original height of the 3rd party content

`{Event}` **cb**
An optional callback function that will be called as a notification of event status

Returns:
void

Related Section
Event details in section 5.1

Example

```
<-- External Party tag -->
<script type='text/javascript'>

var w = window, sf = w["$sf"], ext = sf && sf.ext;

function status_update(status, data)
{

}
if (ext) {
    try {
        ext.register(300, 250, status_update);

              alert(ext.meta("context"));
              //read some metadata passed in from the host side
    } catch (e) {
        alert("no SafeFrame available");
    }
}
</script>
```

5.3 Function $sf.ext.supports

`$sf.ext.supports()`

Availability: Synchronous (can be requested at anytime)

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

Returns

`{Object}`
An object containing a list of SafeFrame container features that are available, defined as follows:

`{Boolean} exp-ovr`
Whether or not expansion is allowed in overlay mode. Default value is `true`.

`{Boolean} exp-push`
Whether or not expansion is allowed in push mode. Push expansion, a method of content expansion in which Host content is "pushed" instead of expanding over the content, is not yet supported in SafeFrame but may be supported separately by the Host. Default value is `false`.

`{Boolean} read-cookie`
Whether or not the host allows external party content to read host cookies. Default value is `false`.

`{Boolean} write-cookie`
Whether or not the host allows external party content to write cookies to the host domain. Despite value of `true`, the host may reject cookie values when offered if deemed appropriate. Default value is `false`.

Example

```
//Sample JavaScript implementation
//Let's say that a 300x250 ad has been declared to fully expand to 400
pixels to the left and 200 pixels to the top.


function feature_check(which)
{
    var o = $sf.ext.supports();

    return (o && o[which]);
}


function expand()
{
    if (feature_check("exp_push")) {
        $sf.ext.expand({l:400,t:200,push:true});
    }
}
```

5.4 Function $sf.ext.geom

`$sf.ext.geom()`

Availability: Synchronous (can be requested at anytime)

The geom function enables an exchange of geometric dimensions and location of the SafeFrame container and its content in relation to the browser or application window and the screen boundaries of the device in which the host content is being viewed.

Host Implementation Note	If called, the Host is required to return the requested values.

This information can be used to:

- Determine available direction and dimensions for content expansion
- Determine whether or not the SafeFrame container is "in view"

Viewability Note	SafeFrame provides information that can be reported in terms of availability according to accepted industry recommendations; however, SafeFrame does not directly report viewability metrics. One metric necessary for reporting viewability is duration, which must be derived by registering a status update listener to determine the duration for how long the `self.iv` is registers as `true`. See section 5.2 for details on the `$sf.ext.register` function.

Returns

`{Object} g`

An object containing sub objects with geometric information about the container. Geometric information may be returned as described in the following lists.

win

Identifies the location, width, and height (in pixels) of the browser or application window boundaries relative to the device screen.

- `{Number} t`
  The y coordinate (in pixels) of the top boundary of the browser or application window relative to the screen
- `{Number} b`
  The y coordinate (in pixels) of the bottom boundary of the browser or application window relative to the screen
- `{Number} l`
  The x coordinate (in pixels) of the left boundary of the browser or application window relative to the screen
- `{Number} r`
  The x coordinate (in pixels) of the right boundary of the browser or application window relative to the screen
- `{Number} w`
  The width (in pixels) of the browser or application window (win.r – win.l)
- `{Number} h`
- The height (in pixels) of the browser or application window (win.b – win.t)

self

Identifies the z-index and location, width, and height (in pixels) of the SafeFrame container relative to the browser or application window (win). In addition, width, height, and area percentage of SafeFrame content in view is provided, based on how much of the container is located within the boundaries of the browser or application window (win).

- `{Number} t`
  The y coordinate (in pixels) of the top boundary of the SafeFrame container
- `{Number} l`
  The x coordinate (in pixels) of the left side boundary of the SafeFrame container
- `{Number} r`
  The x coordinate (in pixels) of the right side boundary of the SafeFrame container (self.l + width of container)
- `{Number} b`
  The y coordinate (in pixels) of the bottom boundary of the SafeFrame container (self.t + height of container)
- `{Number} xiv`
  The percentage (%) of width for the SafeFrame container that is in view (formatted as `"0.14"` or `"1"`)
- `{Number} yiv`
- The percentage (%) of height for the SafeFrame container that is in view (formatted as `"0.14"` or `"1"`)
- `{Number} iv`
  The percentage (%) of area for the SafeFrame container that is in view (formatted as `"0.14"` or `"1"`)
- `{Number} z`
  The Z-index of the SafeFrame container
- `{Number} w`
  The width (in pixels) of the SafeFrame container
- `{Number} h`
  The height (in pixels) of the SafeFrame container

exp

Identifies the expected distance available for expansion within the host content along with information about whether controls allow the end user to scroll the page. If "scrollable," the SafeFrame content can expand to dimensions greater than those provided.

- `{Number} t`
  The number of pixels that can be expanded upwards
- `{Number} l`
  The number of pixels that can be expanded left
- `{Number} r`
  The number of pixels that can be expanded right
- `{Number} b`
  The number of pixels that can be expanded down
- `{Number/Boolean} xs`
  A response that indicates whether the host content is scrollable along the x-axis (1 = scrollable; 0 = not scrollable)
- `{Number/Boolean} yx`
  A response that indicates whether the host content is scrollable along the y-axis (1 = scrollable; 0 = not scrollable)

Since calculating geometric information and exchanging messages can impact performance, geometric information should only be updated during the following times:

First render of the SafeFrame container

When the SafeFrame container is first rendered, `$sf.ext.geom` should be processed and results sent along with external content to be rendered

When changing the size or location of the SafeFrame container

The `$sf.ext.geom` function should be processed when the container size or location is changed using one of the following functions:

- `$sf.ext.expand`
- `$sf.ext.collapse`

When outside updates from host are received

- Upon receiving a message from the host side where the container geometry has been updated by the host itself, such as forcing the content to collapse. See the registration callback messages.
- Upon a scroll of the over all viewable area but only one update per second is allowed (throttling)
- Upon resize of the over all viewable area, but only one update per second is allowed (throttling)

Host Implementation Note	For scroll or resize events, the Safe Frames host implementation should only listen for these events for the first parent HTML element above the SafeFrame container that is either clipped or scrollable.

Example

```
//Sample JavaScript implementation
//Let's say that a 300x250 ad has been declared to fully expand to 400 pixels
to the left and 200 pixels to the top.

function expand()
{
        var w = window, sf = w["$sf"], ext = sf && sf.ext, g, ex;

        if (ext) {
            try {
                g   = ext.geom();
                ex  = g && g.exp;
                if (Math.abs(ex.l) >= 400 && Math.abs(ex.t) >= 200) {
                        ext.expand({l:400,t:200});
                }
            } catch (e) {
                //do not expand, not enough room
            }
        } else {
            //api expansion not supported
        }
    }

 function status_update_handler(status)
    {
    if (status == "expanded") {
            // The ad has finished expanding
        }
    }
```

5.5 Function $sf.ext.expand

`$sf.ext.expand(obj)`

Availability: Asynchronous (only first request is accepted; additional requests are rejected until initial request is processed)

This method expands the SafeFrame container to the specified geometric position, allowing intermediary expansions. The pixel per direction is the absolute position in respect to the original offset declared by the `init` register method. If this method is called without the initialization method, an error may be thrown and it will be ignored. The expand method can only be called from initial size in order to maintain performance.

Tweening or animation is not supported in the SafeFrame so any animation must be processed by the external party within the container and call this method whenever it needs to expand to its maximum size.

At least one of the offset parameters is mandatory. If all of the parameters are missing, the call is ignored and an error may be thrown. At the end of this method the external party registers the status of execution. If the SafeFrame iframe is already at the maximum size, the call is ignored.

Parameters:

`{Object}` **obj**
A descriptor object that defines the top, left, bottom, right coordinates for expansion. At minimum, 1 value must be specified.

`{Number}` **obj.t**
The new top coordinate (y) relative to the current top coordinate.

`{Number}` **obj.l**
The new left coordinate (x) relative to the current left coordinate.

`{Number}` **obj.r**
The new right coordinate (x+width) relative to the current right coordinate (x+width).

`{Number}` **obj.b**
The new bottom coordinate (y+height) relative to the current top coordinate (y+height).

`{Boolean}` **obj.push**
Whether or not expansion should push the host content, rather than overlay.

Implementation Note	"Push" functionality is an expand feature that "pushes" host content in the direction(s) in which external content expands. Technology for the supporting the push expand feature *is not directly specified in SafeFrame 1.0.* The Host must explicitly declare whether Push is allowed in the `supports` property of the `$sf.host.posConfig` object. If allowed, the Host must be able to technically support the functionality.

Returns:

void

Example

```
//Sample JavaScript implementation
//Let's say that a 300x250 ad has been declared to fully expand to 400
pixels to the left and 200 pixels to the top.

var expansionPending = false;
var expanded         = false;

function expand()
{
       var w = window, sf = w["$sf"], ext = sf && sf.ext;

       if (ext) {
           ext.expand({l:400,t:200});
       } else {
           //api expansion not supported
       }
    }

 function status_update_handler(status)
    {
    if (status == "expanded") {
           // The ad has finished expanding
    }
}
```

5.6 Function $sf.ext.collapse

`$sf.ext.collapse()`

Availability: Asynchronous (only first request is accepted; additional requests are rejected until initial request is processed)

This method collapses the SafeFrame container to the original geometric position. This initial size should have been declared in the initialization register method prior to calling this method. If this method is called without the initialization register method, it may throw an error, and will be ignored. If already at the initial size, the call will be ignored.

Returns:

Void

Example

```
//Sample JavaScript implementation

function collapse()
{
       var w = window, sf = w["$sf"], ext = sf && sf.ext;

       if (ext) {
           ext.collapse();
       } else {
           //api expansion not supported
       }
   }

 function status_update_handler(status)
   {
   if (status == "expanded") {
           // Expanded
       } else if (status == "collapsed") {
           //we called collapse
       }
   }
```

5.7 Function $sf.ext.status

`$sf.ext.status()`

Availability: Synchronous (can be requested at any time)

Returns information about the current state of the container, such as whether or not an expansion command is pending, etc. The following are a list of status code strings that can be returned (more may be added in subsequent versions). Some strings are analogous to status updates received in the function that you provide upon calling `$sf.ext.register`.

Returns:

`{String}` One of the following strings may be returned

expanded
Denotes that the container has been expanded.

expanding
Denotes that an expansion command is pending.

collapsed
Denotes that the container is in the default collapsed state.

collapsing
Denotes that a collapse command is pending.

Related Sections

5.2

Function $sf.ext.register

5.8 Function $sf.ext.meta

`$sf.ext.meta(propName, ownerKey)`

Use to retrieve metadata about the SafeFrame position that was specified by the host. The host may specify additional metadata about this 3rd party content. The host specifies this metadata using the `$sf.host.PosMeta` class.

Because the host may want to use some of this data for its own purposes and not share it with the external party, the external party content must use this function to access the metadata information. This way, external party content cannot scan for any values the host does not wish to share.

Parameters:

`{String} propName`
The name of the metadata value you want to read

`{String} ownerKey` (Optional)
The name of the owner object from which to read the property. By default this value is "shared" meaning look in common data.

Returns:

`{String|Number|Boolean}`

Example: 1 - Retrieve a shared metadata value

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

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

Related Sections

5.8 Function $sf.ext.meta

5.9 Function $sf.ext.cookie

`$sf.ext.cookie(cookieName, cookieData)`

Availability: Asynchronous (reading/writing require passing a function to `$sf.ext.register`)

Sends a message to the host to read or write a cookie in the host domain. Note that if host supports this functionality, cookie data is not returned directly from this function as it is asynchronous. You must pass a function to `$sf.ext.register`, which will be then called when the cookie data is set or retrieved.

Host Implementation Note	Allowing an external party to read or set cookies poses a security risk on certain secure pages, such as login pages. Consider whether allowing cookie reading or setting is safe for the page before allowing it.

Parameters:

`{String} cookieName`
The name of the cookie to set or read.

`{Object} cookieData` (Optional)
An object that contains the value, and potentially an expiration date, of a cookie to be set. If not set, the Host assumes that External Party content is only interested in reading the Host cookie value. If set but no expiration date is given, the Host assumes that any cookie written to the Host domain is intended to remain indefinitely.

If offered the following parameters are available:

`{String} cookieData.info` (Required)
A string value for the cookie.

`{Date} cookieData.expires` (Optional)
A date for when the cookie should expire.

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
    }
}
```

5.10 Function $sf.ext.inViewPercentage

`$sf.ext.inViewPercentage()`

Availability: Synchronous (can be requested at anytime)

Returns the percentage of area that a container is in view on the screen as a whole number between 0 and 100.

Implementation Note	The information provided in this function is available in the `$sf.ext.geom` function return as the `self.iv` value. This additional function is offered as a convenience for easier access to the information.

Returns:

`{Number}` The percentage of area that a container is in view on the screen

Industry Standard Viewability

Industry-accepted viewability metrics may require a duration component for reported viewable impressions. Duration can be determined by calculating how long the value for `$sf.ext.inViewPercentage` meets or exceeds the minimum-accepted percentage for a viewable impression.

The following code sample demonstrates how a registered "listener" might determine duration (values in **bold** to be replaced with industry-accepted viewability values):

```
var viewableTimerId = 0;
var viewableFired = false;

function nodifyViewablePassed()
{
    if(viewableFired) return; // fire beacon
        viewableFired = true;
        viewableTimerId = 0;
}

function status_update(status, data)
{
    // notify if 50% in view for 1 second
    if($sf.ext.inViewPercentage() > 50)
    {
        if(viewableTimerId == 0){
            viewableTimerId = setTimeout(function()
            {notifyViewablePassed(); }, 1000);}
    }
        else{
        clearTimeout(viewableTimerId);
        }
}

$sf.ext.register(160, 650, status_update)
```

5.11 Function $sf.ext.winHasFocus

`$sf.ext.winHasFocus()`

Availability: Synchronous (can be requested at anytime)

Returns whether or not the browser window or tab that contains the SafeFrame has focus, or is currently active.

Returns:

`{Boolean}` True if the browser window / tab has focus, otherwise false

Version Requirements:

"`1.1`" Requires specVersion 1.1 as opposed to original functionality in "1.0".

Relationship to Viewability

In addition to geometric coordinates, the content within a SafeFrame may like to know that the main window is currently active, or in focus. This function provides that information and may be considered when reporting viewable metrics.

Viewability Note	The `winHasFocus` function provides information that can be considered as part of a viewable metric. The information this function reports does NOT determine or report viewability. Viewability metrics are determined by the industry and the parties involved in a media deal in which viewability is reported.

The following code sample demonstrates how a registered listener might determine if the main browser window or tab has focus.

```
var win_has_focus = false;

function status_update(status, data)
{
      // notify if 50% in view for 1 second
      if(status == "focus-change") {
          win_has_focus = $sf.ext.winHasFocus();
      }
}

$sf.ext.register(160, 650, status_update)
```

End