HTML5
FOR DIGITAL ADVERTISING

GUIDANCE FOR AD DESIGNERS & CREATIVE TECHNOLOGISTS

VERSION 1.0.1

UPDATE RELEASE NOTES

Published July 15, 2013

.

This document has been developed by the IAB Ad Operations Council and the Mobile Marketing Center of Excellence
The HTML5 for Digital Advertising (HTML5_DAv1.0) document was created by a working group of volunteers from 19 IAB member companies.

The HTML5 Digital Advertising Working Group was led by:

- Cory Hudson, AOL & ADTECH
- Keith Walter, JumpTap

The following IAB member companies contributed to this document:

AOL & ADTECH
CBS Interactive
CMG Digital (Cox)
Crisp Media
FreeWheel
Google & YouTube
InMobi
JumpTap
Medialets
Pandora Media Inc.
Pictela
PointRoll
Spongecell
SpotXchange
The Weather Channel
Time Inc.
Tribune
Turner Broadcasting System, Inc./CNN.com

The IAB leads on this initiative were Jessica Anderson and Sabrina Alimi

Contact adtechnology@iab.net to comment on this document.

About the IAB's Ad Operations Council and Mobile Marketing Center of Excellence

The Ad Operations Council is dedicated to improving the operational efficiency of interactive advertising. Ad Operations Council working groups regularly include agency-side representatives to help improve communication, understanding, and work process in many areas of the buyer-seller relationship. A full list of Council member companies can be found at: http://www.iab.net/ad_ops_council

The IAB Mobile Marketing Center of Excellence, an independently funded and staffed unit inside the IAB, is charged with driving the growth of the mobile marketing, advertising and media marketplace. The Mobile Center devotes resources to market and consumer research, mobile advertising case studies, executive training and education, supply chain standardization, creative showcases and best practice identification in the burgeoning field of mobile media and marketing. The agenda focuses on building profitable revenue growth for companies engaged in mobile marketing, communications and advertising, and helping publishers, marketers and agency professionals understand and leverage interactive tools and technologies in order to reach and influence the consumer. More information can be found at: http://www.iab.net/mobile

This document is on the IAB website at: http://www.iab.net/html5

Table of Contents

Executive Summary

HTML5 for Digital Advertising 1.0 is a starting point for creative technologists. This guidance focuses on how to package and optimize HTML5 ad code for delivery. Guidance on HTML5 ad formats and technical specification are part of additional steps that need to be taken in the IAB HTML5 initiative and are NOT included in this document. The goal for this document is to align creative technologists on some best practices for packaging HTML5 creative. Ideally, establishing some common ground will help reduce operational overhead as companies make the shift from graphical development tools like Flash™ to the more code-heavy format of HTML5.

The demand for HTML5-formatted ads increases as the promise for seamless support across desktop computers and mobile devices is propagated throughout the digital advertising marketplace. While HTML5 offers rich design features, it comes with the need for a different skill set.

Designing ads with common ad design packaging tools like Flash™ is done using a visual platform, but designing HTML5 ads relies heavily on code. This shift from visual design to code-based design will increase companies' operational overhead. These costs are magnified when publishers and ad developers lack a common framework for HTML5 ad optimization.

In an effort to improve ad effectiveness, companies are rising to adopt HTML5 and incurring the cost that comes with the shift. However, the operational cost associated with supporting the new format can be reduced if ad developers and publishers operate on a common framework for HTML5 ad optimization. Ultimately, wide adoption of the HTML5 ad development standard may improve and reduce operational costs, but the digital advertising industry will struggle with HTML5 until much needed guidelines are put in place.

As a framework for HTML5 ad optimization, this document addresses some of the common hurdles that have stifled HTML5 ad implementation. Until guidance for more complex issues in HTML5 adoption is available, adhering to the practices outlined in this document will help companies begin to build HTML5 ad development workflows that are more scalable and consistent while improving ad performance and user experience across all platforms.

Intended Audience

HTML5 ad developers who need a common framework for optimizing their ads for the best consumer experience and ad load performance will benefit most from this document. However, creative designers, publishers and those involved at any point in the ad technology supply chain will benefit from being familiar with the HTML5 ad optimization techniques offered in this document.

1 General Overview

Released as an official recommendation by the World Wide Web Consortium (W3C) in December 2012, HTML5 is the latest update to the Hypertext Markup Language (HTML) that includes new semantic tags for features such as video, audio, canvas, and other design features. HTML5 has grown into an industry buzzword that has come to encompass all the various web technologies and APIs that work together to execute animation and other interactions. While HTML5 buzz holds much promise, the current state of HTML5 for ad creation is still in a stage of exploration and refinement. The transition from visual creative development tools to creative coding skill sets, functionality, and additional technology required, HTML5 ad creative practitioners are discovering the methods and practices that work best for their organizations' operational workflows and looking at effective ways to deliver across devices and systems.

The digital advertising ecosystem has a long way to go before this complex set of tools is absorbed into everyday ad technology. Yet, despite the code and additional technology involved, HTML5 provides some of the following benefits:

- It's based on a well-established format that many authoring tools and developers can produce
- It provides a rapid testing infrastructure that all browsers should be able to process without using rendering libraries or wrappers
- It preserves HTML standard methods of laying out content such as positioning, layering, and opacity

Complex issues will arise and the need for formal guidelines and practices around this technology will present itself as more companies move toward adoption. As an initial step to aiding the digital advertising industry in its transition to HTML5, guidance for the most common display ad types is needed to establish common practices necessary for optimizing, packaging and serving an HTML5 display ads, from which richer creative executions can later be developed.

This document provides general guidance on HTML5 ad optimization for display advertising, driving awareness to ad delivery methods, packaging recommendations, and considerations for fallback methods. It also provides guidance on applying video and animation features within an ad and is followed by a tools section addressing designer and developer audiences. In addition, an HTML5 Resource Wiki was developed as a supplemental resource for tools, features and functionality that will evolve over time, and is open to the industry to submit content.

More complex ad formats, especially those that include expansion features, are out of scope for this document and will be addressed in future iterations or in other HTML5 initiative projects.

Although it is too early to declare a guideline or specification for HTML5 ads, this initial document from the IAB HTML5 Working Group helps to ensure reduced fragmented workflows, encourage clean ad code and packaging, and ultimately positive delivery performance across web pages and devices.

1.1 HTML5 and the IAB Display Advertising Guidelines

As a new format for developing interactive display ads, HTML5 was not ready to be considered for the February 2012 release of the IAB Display Advertising Guidelines. Certain features in HTML5, such as the ability to use a responsive ad size instead of specifying a width and height, complicate defining ad formats that fit into the Display Guidelines.

This document defines only one ad format: the HTML5 display ad. At this time, the HTML5 display ad is simply a broad definition for any ad that does not expand outside of its borders. This basic ad type can be constructed with rich animations and video so long as all features are contained within the boundaries of the ad.

To comply with the IAB Display Advertising Guidelines: The 2012 Portfolio, HTML5 display ads should conform as much as possible to the specific ad formats defined. An update to the 2012 Portfolio is anticipated to include considerations for HTML5 ads, but until these updates are made, use good judgment in developing ads that the industry can readily accept, based on the general guidance presented within this document and conforming as much as possible the 2012 Portfolio for IAB Display Advertising Guidelines.

HTML5 Ad Format Note	The general guidance in this document is only preliminary. More detailed HTML5 Creative Guidelines will be established later. The methods presented in this document are only suggestions for basic display ads. Advanced methods like responsive design or rich media concepts that require complex dependencies on external calls are **out of scope**. Common scenarios where external calls are used are addressed within this document, but further investigation and collaboration is required before more in-depth guidelines can be determined.

1.2 HTML5 Ad Distinction

An HTML5 ad is effectively an HTML document or a web page that follows the W3C/WHATWG standards, meaning that web browsers must be able to render the ad like a web page. Preceding all content should be the HTML tag <!DOCTYPE html>. The document should also contain at least <html> and <body> tags.

1.2.1 Visual Distinction

Since the webpage content and HTML5 ad share the same rendering technology (the browser) and resources, the ad unit will need to be clearly distinguishable from the normal webpage content as per the IAB Display Advertising Guidelines. To protect the aesthetics of the ad and brand message, the border should be applied in the ad creative rather than impose upon the ad server to produce a border.

1.2.2 Technical Distinction

An HTML5 ad is its own independent document that is effectively operating from a different origin than the publisher's web page, and depending on the method in which it is delivered, can impact the publisher content or the effectiveness of its own functionality. To enable an independent ad space within the publisher's page and avoid naming collisions between page and creative elements, ad servers and publishers implement a container for the ad the using an iframe.

Iframes help to ensure security for publishers, but they also limit the use of advanced ad features such as expansion along with lack of metadata sharing and other interactive communication. The IAB released SafeFrame 1.0 in early 2012, which offers a more interactive iframe solution using a cross-domain iframe and an API. See section 2.4.3 for more information about SafeFrame.

In mobile applications, HTML5 ads are segregated from content by using a WebView that contains only the HTML5 ad document. As with iframes, ads served into a WebView lack the ability to interact with the application. The IAB Mobile Rich Media Ad Interface Definition (MRAID) solution enables communication between the ad unit and mobile applications. Please see section 2.4.3 for more information about MRAID.

2 HTML5 Display Ads

Ads created in HTML5 offer benefits for flexibility and freedom to deliver ads across platforms and devices. In addition, the demand for rich media is driving adoption of HTML5. While the race to develop and deliver ads in HTML5 has already begun, a baseline for general HTML5 ads needs to be established before we can take the next step in defining specific formats and richer capabilities such as ad expansion, wallpaper and transparency layers.

This section presents a general definition for HTML5 Display Ads until a more formal definition is established. The HTML5 display ad described here includes essential format traits, compression considerations, and ad delivery methodologies. Sections that follow provide recommendations for any video or animation, respectively, that may be included in an HTML5 display ad.

2.1 HTML5 Display Ad Essentials

Establishing a definition for HTML5 display ads enables better communication and workflows among parties across the ad delivery chain and presents a baseline for richer ad creative developments and executions to be included in a later release.

General HTML5 Display Ads are limited to the following traits:

- Creative that does not expand out of its initial boundaries
- Is not dynamic or does not ingest external feeds
- All assets should be packaged and accounted for in the file size

The nature of HTML5 and its distinct differences from Flash mean that exceptions to the suggested external feeds limitation are necessary but should be communicated and agreed upon between all parties involved in the ad delivery process. These HTML5 display ad restrictions meet requirements for the majority of non-rich media ads viewed online and are sufficient to support richly animated ads for maximum compatibility across placements and the ad servers that support them.

Before developing HTML5 ad creative, be sure to consider the technical requirements of the pertinent vendors and partners for ad delivery (ad server and publishers). Such points to remain top of mind are listed below, but guidance on each is provided further into the document:

- File Size limits
- Packaging
- Coding ClickTags for the Click through URL to enable click tracking
- Backup Image or Fallback Experience

2.2 File Size

HTML5 did not become an official recommendation until December 2012, nearly a year after the IAB Display Advertising Guidelines: The New 2012 Portfolio was released. As such, file size limitations were not taken into account for ads developed using HTML5. Current file size limits for ads developed with Flash are sufficient because Flash files can be compiled, compressed, and packaged to accommodate smaller file sizes.

However, HTML5 doesn't have these compression and packaging capabilities, and with high-density displays permeating the market, larger creative assets are necessary in HTML5 to produce crisp visuals. Until IAB Display

Advertising Guidelines is renewed to accommodate HTML5 and high-density displays, the following preliminary guidelines are set forth for HTML5 ads but should not be considered final creative guidelines. Since HTML5 ad development can be optimized in various ways to accommodate file size and performance challenges, you should expect file sizes to be larger than what has been defined for Flash based creatives. Future working group efforts will focus on the appropriate size limits, but for now we recommend you consult your partners (publishers and vendors) on their organizations acceptable file size limits.

HTML5 Ad File Size	HTML5 ad file size should be expected to be larger than what has been defined for traditional creative and should be discussed with other parties involved in delivering and placing the ad.

File Size Measurement Definitions for HTML5 Ads

- File size measured after compressing the ad (all code and assets) to a .zip file
- The .zip file must include all referenced code such as Javascript libraries
- Once the .zip file is uncompressed, the ad (an .html file) must be viewable without a network connection (all code and assets used in the ad is contained in the .zip file)

File Size Limits

In order to establish a common practice, all assets and code for an HTML5 ad should be zipped and delivered together as one file to be unpacked and processed by the publisher. Some files such as Javascript libraries and Web fonts can be called from another location, but the file size of any external files should be considered as part of the initial overall file size because they contribute to ad load performance. Exceptions exist, such as for user-initiated videos, which may be excluded from the initial ad load file size. The maximum file size for HTML5 ads should be negotiated with the publisher.

Well-accepted Javascript libraries, such as jQuery, must be included into the total file size. While this requirement may prove challenging, the limitation is defined to encourage improved ad load performance despite the larger file size. Improvements in library innovation and library version mismatching should help reduce library file size so that more of the maximum file size can be devoted to creative imagery.

The total file weight, including the Javascript libraries, must be known to the ad server in order for it to guarantee a certain level of ad drop-off rate to both advertisers and publishers as well as ensure that the file weight stays under the publisher threshold.

2.3 Optimization of Ad Creative Packaging

Flash automatically provides optimizations for code and assets; it compiles everything into a single file, which is already compact with minimal number of files. For HTML5 ads, IDE or other tools that provide similar optimizations are not yet available. To further complicate the matter, HTML5 is an open web technology, which means a large degree of freedom is exercised in building and delivering content.

Special attention to how an HML5 ad creative file is packaged and delivered is required for quick ad load time, for establishing some common expectations, and to minimize negative impacts to the publisher and consumers. This section highlights some of the important optimizations that are crucial in adopting HTML5 for digital advertising.

2.3.1 Code and Asset Compression

Code Compression

HTML, CSS and Javascript code are typically written in legible text that is highly compressible. Code minifiers and compilers help reduce file size by removing characters like white spaces and line breaks that make the code legible to humans but are not necessary for computers to execute. Minifying code reduces the input required for a coded ad to be compressed, resulting in a smaller ad file for web servers to unpackage.

Asset Compression

Creative development tools for graphics, video, and audio offer a wide array of options for balancing creative quality with file size thresholds. Asset compression is increasingly important with high-density displays. To compress assets, use PNG "crushers" and other forms of image compression, and leverage formats such as scalable vector graphics (SVG) whenever applicable. SVGs can scale indefinitely to high-resolution displays without increasing file size.

Text and Fonts

Certain logos and brand names must be provided as images. However, ads with a lot of text or dynamic text content can benefit from using text directly in HTML documents. HTML text should be used whenever possible while incorporating fonts that are either stored as a part of ad assets, or delivered dynamically as web fonts. Using text and scalable fonts rather than attempting to bake the text into an image offers significant advantages that improve text readability and legibility, reduce image sizes, and improve the accessibility of the HTML5 ad content.

When using fonts or web fonts, consider applying font sub setting. Font sub-setting reduces supported character sets to glyphs that are actually present in the text of an HTML5 ad. Eliminating unused glyphs from an original font prior to adding it to the ad assets offers an easy way to reduce font file sizes and web font download times.

When web fonts must be accessed externally rather than downloaded as part of the ad .zip file, the external font file size must be calculated as part of the total file weight even though it's not included within the zipped asset file.

2.3.2 Minimize File Count

Ads with smaller file size reduce both load time and negative impacts to publishers and the consumer viewing experience. To help in reducing the overall weight of an ad, the number of individual files included for download should be minimized. For every file delivered, the browser must open a new socket connection, which can reduce browser performance as it waits for each new socket to be opened. Some ISPs or mobile carriers limit the number of sockets a device can open in parallel, causing a linear lineup of file downloads. While processing performance isn't reduced, the added wait time is *perceived* as reduced performance.

To reduce the number of individual files for an ad, consider the following:

- Sprite sheets for ads that include several small assets.
- Code compilers that can take all code (Javascript, CSS, HTML, etc.,) and produce a single file.
- Base64 encoding of smaller images into the HTML file to reduce the number of files. However, Base64 adds weight to the image and causes it to be un-cacheable, which is not compatible with IE 6 or 7.

2.3.3 Central Processing Unit (CPU) Resources

Javascript is single-threaded, meaning that for most browsers; there is a single thread that renders both the publisher's page and the HTML5 ad, even if the ad is in an iFrame on a separate domain. In this model, the publisher page may be processed on a different CPU than the ad. Other assets rendered from other sources in this single thread of code may also be processed on other CPUs. Running an ad within applications, especially on mobile devices, using ad SDKs further enhances the complexity of processing logistics.

To improve processing power and simplify logistics, code the ad with CPU utilization in mind and leverage the graphics processing unit (GPU) that exists in most devices whenever possible. Creative developers should maximize code efficiency as much as possible so that it never uses the CPU to its capacity. Also, using CSS styling and transitions for animations divert processing power to the GPU instead of using the CPU.

Considering gzip compression:

Most ad servers will send the packet for the ad as a .gz (gzipped) file to ensure that files are compressed before they are sent over the Internet. Ad designers are not required to use gzip compression, but should consider gzip as a useful tool since nothing is lost in the compression, but results is a smaller file size. Efficient code and assets with optimum compression is important for the best performance and consumer experience.

2.4 Ad Server Compatibility

To bring scale to the ecosystem, an ad designer using creative tools or hand-coding must be confident that ad servers can accept the HTML5 ad produced. Critical to enabling proper rendering and metric counts, the ad creative should be coded consistently so that ad servers can appropriately associate view and click tracking functions to the defined assets.

This section defines the general recommendations for maximizing compatibility with the ad server. Before beginning HTML5 ad development, communication with ad server technology providers regarding specific technical requirements is vital to developing a successful HTML5 ad experience.

2.4.1 Click Tag

The ability to accurately measure when a consumer clicks the ad (clickthrough) is a critical feature of any ad server. Ad servers must be able to identify the click destination of the ad and swap it out with something it can control, most commonly a redirect URL that, when initiated, logs the click. In order to record clickthroughs, the ad server, the publisher and any other parties involved must be able to recognize the code identifying the click and its destination through a standard format.

The following examples are methods for applying the clickTag variable within the ad creative:

var clickTag = "www.example.com";

Click destination methods

Ads must use the `clickTag` variable as the destination of the click event, whether handled by anchor tags `(<a>), window.location, window.open`, or any other method of navigating the user.

When using this variable upon a click, if you use an anchor tag, we recommend setting the target attribute to "_blank". This ensures that the click destination opens in a new page or a tab.

```
 <a href=(the URL of the clickTag) target="_blank">  </a>
```

A typical way to utilize the clickTag variable is to give the anchor tag a unique id and assign the href dynamically after page load via Javascript:

HTML	`<a id=”clickArea” target="_blank"></a>`

Javascript	`var clickArea = document.getElementById(“clickArea”);`
	 `clickArea.href = clickTag;`

To leverage Javascript's window.open method within the HTML:

```
<a href="javascript:window.open(window.clickTag, '_blank')"</a>
```

Multiple Clickthrough URLS:

In instances of multiple clickthrough urls within the same ad, enumerate click tags as follows: `clickTag0, clickTag1, clickTag2`

Click tags should be placed in the .html file without minification or obfuscation. This helps the ad server find the variable easily so that it can substitute the correct value. When multiple .html files are included, the variable must be present in the first .html file that loads.

2.4.2 Ad Unit Size (Dimensions)

Ad dimensions need to match placement dimensions where the ad will display on the publisher page. For example, when a 300x250 ad is served to a placement for a 300x600 placement, the ad is distorted to fit the placement. To prevent distortion, ad servers are built to detect ad dimensions for uploaded creative files. Detecting dimensions for standard images and .swf files is well established for ad units, but HTML doesn’t have a defined unit size, so defining creative dimensions is optional.

One of the powerful characteristics of HTML5 is how easy it is to generate ads that adjust to the container size dynamically, similar to a web-page viewed on browsers of different window sizes. If the width and/or height are left undefined, the publisher or other ad display technology may either disregard the ad unit dimension requirements, or consider the creative to be responsive in size and react according to its policy on responsive ad sizes. IAB recommendations for responsive ads are not addressed in this document. Check with the ad server or publisher on their policy for ad unit dimensions.

Defining Ad Size Dimensions for HTML5 Ads (Optional)

When choosing to define creative dimensions for HTML5 ads, optional functions such as `<meta>` tags or viewports can be applied to the creative code, and are not defined by the ad server.

Size <meta> tag

`<meta>` tags are optional elements that are read by the browser for helping systems identify the size of a creative for a placement. But some methods cannot utilize this option, such as content modules because there is no place to apply the tag. Additionally, `<meta>` tags can be applied either within the `<head>` tag, or the `<body>` tag depending on your organization’s need or practice. The following protocol provides a standard way to define creative dimensions using a `<meta>` tag:

Within the `<head>` tag, use the `<meta>` tag with the name "ad.size" to mark the ad dimensions. For example, if the ad was built as a 300x250, define it as follows:

```
<meta name="ad.size" content="width=300,height=250">
```

2.4.3 Ad Serving Methodologies

To enable an independent ad space within the publisher's page and avoid naming collisions between page and creative elements, ad servers and publishers serve HTML5 ads into an iframe. Iframes help ensure security for publishers, but they limit the use of advanced ad features such as ad expansion and other interaction that requires cross-domain communication.

IAB SafeFrame Solution
The IAB SafeFrame solution is based on iframe technology that uses an API to enable cross-origin communication between the publisher page and any creative served into the SafeFrame. This solution enables rich creative with expansion features to be rendered in an iframe, as well as metadata sharing and other interactions not previously possible from a standard iframe. SafeFrame is also a key element in enabling viewability measurement.

Nested iframes
Whether served into a standard iframe or an IAB SafeFrame, both the publisher and ad server may create an iframe, which results in a situation where the ad server iframe is nested within the publisher iframe. Other technology vendors in the supply chain compound the nested iframes situation by defining their own iframes for served content. Though done to contain content and protect one party from the other, the end result is a chain of nested iframes that prevent any of the parties from interacting properly. Nested iframes create barriers in communication between parties, resulting in a lack of transparency, inability to share metadata, and limits interaction metrics.

The benefits of the SafeFrame solution are negated with nested iframes. SafeFrame was intentionally designed to only communicate with the top-level iframe that is part of the SafeFrame implementation. While traditionally frame-bounded content can still function in a nested iframe within the SafeFrame, proper viewability measurements and metadata sharing are disabled on the serving side of the nested iframe relationship.

Although creative developers don't define the iframes to which ads are served, to take advantage of solutions like SafeFrame, all parties should communicate ad serving methodologies and logistics between the ad server and the publisher. Communication will ensure that when creative is served to a SafeFrame, a Javascript tag is used instead of an iframe to serve the ad content.

For more information on SafeFrame, please visit http://www.iab.net/safeframe.

Web Applications and IAB MRAID
In the case of serving ads into ad software development kits (SDK) for mobile applications, a WebView that contains only the ad achieves: independence, segregation and the ability to render a full HTML document. For ad expansion and other rich interactions, the IAB Mobile Rich media Ad Interface Definition (MRAID) offers an API for interacting with mobile applications. Although originally defined for mobile, this interface should be applicable to ad serving SDKs and WebViews regardless of the device. Even with SDK and WebView applications, an ad server may still serve the ad into an iframe, but iframe or not, check with your ad server to ensure the ad creative is given the space it needs to display as designed.

For more information on MRAID, please visit http://www.iab.net/mraid.

2.5 Backup Image Experience

Flash players are easily detected in today's Web browsers. Thus, the JavaScript checks for Flash before downloading a .swf file. If Flash is not supported, a static image is downloaded instead. This backup image is a separate creative that ad servers accept from ad designers in addition to the .swf file. In most cases, the backup image file is required.

In HTML5, the ad and its features need to be compatible with the browser to which it is served in order to display (render) as intended. Compatibility depends on which APIs the HTML5 ad uses and whether the browser supports each of those APIs. Since HTML5 is a collection of APIs that browser manufacturers can adopt to be compliant, browsers can choose to implement all or partially, without any particular order or priority. In short, whether a browser supports "HTML5" or not is undefined because HTML5 is a collection of API's rather than a stand-alone format. As of the release of this document, ad servers are not designed to detect HTML5 compatibility in browsers.

HTML5 Fallback Recommendations:

Considering the lack of HTML5 identification, the client and the creative developer should discuss various features of the ad and acceptable browser scenarios. Fallbacks should be designed as part of the ad creative. Fallback preparation should include testing ad features across browsers to ensure acceptable rendering as well as static images when required by publishers. Additional recommendations to help ensure HTML5 ads work as designed include:

- **Ad designers must be aware of HTML5 features they use**. Depending on the API usage, an ad can be compatible with all existing browsers or only a certain subset. There are several tools available to identify which browsers support which features, for more information, see the IAB Wiki: HTML5_for_Digital_Advertising_Resources.

- **"Graceful degradation" is highly recommended**. Just because one feature is not available on a given browser where the ad is rendering does not mean that the ad is incompatible. For example, if getting the geo location is not available in the browser, the ad can fall back to the user typing in the zip code or city name.

- **No mandatory backup image** is necessary, given that the ad server cannot control when to show it. Instead, the ad code must detect if certain features or APIs are failing and, if so, the ad should degrade gracefully. Libraries such as Modernizr help with browser feature detection at run-time.

- **`<noscript>` Tag** should be applied, this provides a path to an alternate image when users have scripts disabled in their browser, or don't support client-side scripting. Check with your ad server to see if they use a no script tag to deliver a static image to users that have Javascript disabled in the browser. If this is the case, you may need to provide static a image.

Ad designers may be challenged as they design fallbacks for each HTML5 feature used, but the effort involved defines the quality of the consumer experience and therefore the effectiveness of the ad. As HTML5 adoption increases across the industry, creative tools will soon evolve to aid designers in HTML5 ad development. Also, browsers will eventually improve their support of the HTML5 standard, which will further reduce compatibility fragmentation.

2.5.1 Zip File Contents

The .zip file that is passed to the ad server should follow the guidelines below:

✓	There must be at least one .html file (the starting point of the ad) in the .zip. If multiple .html files exist, the ad server should prompt the uploader for the appropriate .html file to use as the starting point.
✓	Structure files as needed. No specific rules are outlined for the folder structure of the .zip file. Files may be organized in subfolders or may be present solely within the root folder.
✓	All code and assets should be relatively referred to by the .html file
✓	Minimize the number of files included within the .zip file. For performance reasons, some ad servers may limit the number of individual files that may be included with the ad.
✓	All code and assets needed to run the ad should be contained in the .zip file. The ad needs to be self-contained so that rendering the ad is not dependent on a network connection. Exceptions include files such as Javascript libraries or web fonts, but the file size of these external files should still be considered part of the overall file size if they are loaded upon the initial file load.

HTML5 Display Ad Note	These guidelines do not address rich media ad executions. For more advanced ads or rich media executions, ad servers and publishers may have separate requirements, limits or guidelines specific to the ad type.

3 In-Banner Video

The following sections establish a set of guidelines for HTML5 creative that make use of the video tag so that creative designers can produce ads with video that function correctly across multiple browsers.

In-Banner Video vs. In-Stream Video

HTML5 in-banner video is defined as an ad with video that is served to a Web browser. Conversely, an ad can be served to a video player, which is defined as in-stream video. At this time, this document only covers in-banner video ads that are served to browsers.

3.1 HTML Video Tag

Before HTML5 video, creative designers would create video playback from within the Flash (SWF) creative. Because the Flash Player was ubiquitous across all desktop screens, this created a video solution that would run flawlessly across multiple browsers. However, with the emerging phone and tablet market, marketers and creative developers are faced with building a "browser only" solution using HTML5's video element.

Syntax for Implementing an HTML5 Video Element

Define the video within the HTML5 code using the following tags:

```
<video></video>
```

Defining the video includes providing the source filename and the width and height. You can also indicate default video player controls, style your own in the CSS file, or leave them out altogether. Attributes can be included to customize the video experience for your particular use case.

Video without controls:

This video element example instructs the browser to render an HTML5 video element to playback the file, "yourVideoFile.mp4" at the dimensions 640x360.

```
<video src='yourVideoFile.mp4' height='640' width='360'></video>
```

Video with controls:
The added `controls` attribute render's default video player controls. If you wish to create your own controls for the video player, you will need to remove the `controls` attribute from the video tag and define the look and functionality by using your own CSS and Javascript. The following example uses the `controls` attribute (indicated in red) used to invoke the default player controls:

```
<video controls src='yourVideoFile.mp4' height='640' width='360'></video>
```

HTML5 Video in Mobile Phones	Many mobile devices only support full-screen video playback, and in cases where video is rendered in the native video player at full screen, custom controls do not apply.
Video Full Screen	To provide the option to play back video in full screen mode, leverage the W3C Fullscreen API. As of the publish date for this document, the API was in a working draft state. To ensure proper function, the HTML5 ad must detect whether the feature is supported in the targeted browser. Since this attribute is not supported by all browsers, some vendors require implementing a prefix for proper support.
Transparency	Alpha transparency is not supported in any codec or wrapper format using the HTML5 video element—as of the publishing of this document. Until this feature is supported, Flash must still be used to achieve an immersive ad experience using transparent video. For mobile, use the fallback to a clickable link to view the video.

3.1.1 Browser Support

While the HTML5 video element is widely supported among the latest major browser vendors, the video asset itself needs to be transcoded into a few different formats to ensure cross-browser functionality. Resources to check the most updated browser support are provided in the IAB Wiki, HTML5_for_Digital_Advertising_Resources.

As of this publication, the most common formats that are supported are H.264/MP4 and VP8/WebM. At a minimum, HTML5 video should be transcoded into these two formats. Multiple video formats can be provided within the `<video>` element by identifying multiple <source> tags, as shown in the example below:

```
<video controls height='640' width='360'>
    <source src='yourVideo.mp4' type='video/mp4' />
    <source src='yourVideo.webm' type='video/webm' />
</video>
```

To ensure proper functionality across both HTML5-supported browsers and older ones that lack necessary support, incorporate code designed to detect browser and device details. When HTML5 is not supported as needed to play the ad's video, design the ad to "degrade gracefully," or fallback on an alternate method for displaying the video, such as with a backup Flash file.

The following code sample provides an example of how to embed a Flash video to fall back on should the HTML5 video not be supported:

```
<video controls height='640' width='360'>
      <source src='yourVideo.mp4' type='video/mp4' />
      <source src='yourVideo.webm' type='video/webm' />
      <! -- Flash Code Here -->
      <embed src='yourVideo.flv' width='640' height='360' quality='high'
type='application/x-shockwave-flash'></embed>
</video>
```

By including the Flash failover code within the `<video>` element, the browser will omit the video tag if it does not recognize it and use the Flash code instead. Conversely, browsers that support the video element will not render the portion within the video tag, in this case our Flash embed code.

3.2 Video Events

A crucial part of ad measurement understands time spent and interactions within a video ad experience. Through the HTML5 Video Javascript API, we can listen and handle for such events as: video start, play, pause, and progress percentage.

Official IAB metrics have not been developed for in-banner video, but metrics developed for digital in-stream video may serve as a useful reference for establishing common metrics within in-banner video. Please review the IAB Digital Video In-Stream Ad Metrics Definitions as a frame of reference.

To detect events using Javascript, reference the `<video>` element within your ad creative and add an event listener for the specific event you want to handle. The following example demonstrates how to detect events by listening for a "play" event.

```
var theVideo = document.querySelector('#theVideo');

theVideo.addEventListener("play", videoStartHandler, false);
function videoStartHandler (event) {
      console.log("Video Start "+ event);
}
```

For a comprehensive resource on the HTML5 Video Javascript API, please visit the W3C page: HTML5 Video Events and API.

3.3 Forced Video

In some cases creative designers can leverage video playback to replace animation that was too large for specified file size limits, or for other reasons, have the video play as soon as the ad loads. To have an HTML5 video play upon being loaded, declare the `Autoplay` attribute:

```
<video autoplay src='yourVideoFile.mp4' height='640' width='360'></video>
```

Autoplay on mobile devices is not allowed, as per the Mobile Phone Creative Guidelines. In most cases mobile devices ignore this attribute, but if the mobile device is not set to autoplay, the video may still be forced on users by other means. Forcing autoplay on mobile devices while tied to cell carrier networks (2G, 3G, 4G), may result in unexpected overage fees for the device owner.

Forced Video with Audio
Audio must be user-initiated to play, and while video may play automatically in non-mobile devices, audio must be set to a muted state until the user initiates it, per the IAB Display Creative Guidelines.

3.4 Delivery and File Size

One of the biggest concerns with HTML5 video is around delivery. Traditionally, Flash ads pulled video from a streaming server to avoid incurring any of the video's file size in the ad's overall k-weight. However, most HTML5 video delivered over desktop and mobile default to progressive download (video file is downloaded just as an image would be) instead of using a streaming format where the video is processed in real-time sending small packets of video information from its originating location over HTTP to the end user.

While progressive download is widely supported across all browsers, the video file is included as part of the ad’s file size limit. The bigger the file, the more the consumer experience is negatively affected with longer wait times and a perception that the publisher page is slow. The problem is amplified when video bitrates increase for larger dimensions and higher quality video assets (HD).

Video File Size Limit
Whenever delivering video, the file weight must be kept small, but this recommendation is especially applicable when targeting mobile devices that may be connected to limited data plans. IAB Display Advertising Guidelines recommend unlimited, user-initiated streaming video. However, these guidelines were defined for display on non-mobile devices. Until guidelines are defined for mobile video ads, a video file size of 2.5 MB or smaller is recommended.

For HTML in-banner video ads that are specifically targeted to mobile devices, some delivery tools are available to help improve video file delivery to mobile devices. For more information, please visit IAB’s site for HTML5 tools at IAB Wiki: HTML5_for_Digital_Advertising_Resources.

4 Animation

HTML5 animation can be incorporated into display ads in a number of ways and may not be processed the same way as traditional methods. Certain considerations for performance, execution, and publisher/ad server requirements must be taken into account. The following sections offer recommendations for navigating HTML5 animation options.

4.1 Animation Performance

Creative designers have various animation methods to choose from, but the choice of certain animation elements can overwhelm the CPU and affect ad load performance. CPU usage can be affected by various animation elements including: the number of objects being moved simultaneously, the efficiency of the animation code, excessive animated images with transparency, multiple simultaneous CSS3 effects, and complexity of the document’s layout. All of these items impact and determine the amount of work required for browser to render each frame. To avoid animation affects that are excessively CPU intensive, consider proper use of Javascript time controls like `setInterval, setTimeout, requestAnimationFrame`; animation loops, or DOM element manipulation. Then optimized accordingly.

4.1.1 CSS Animations Vs. jQuery/Javascript Animations

Animation can be coded through the use of CSS or jQuery/JS but it is important to be aware of their functions and capabilities before determining which to apply.

jQuery/JS animation offers more control over CSS methods, and when targeting audience's browsers, the resultis fewer issues with Internet Explorer 8 and other older browser versions. jQuery may be necessary when key animation must be seen on desktop browsers although using jQuery creates extra overhead upon loading Javascript in conjunction with the ad's code, which results in a delay in the animation play.

CSS animation is encouraged for improved performance because it is native code that can execute quickly and involves less coding, which eases the use of the processor. In the case of non-supported browsers, CSS is your best option over jQuery, if you are comfortable with gracefully degrading the animation. Support for CSS3 methods should be verified through feature detection prior to serving.

Unique CSS and Javascript Closures & Namespaces	To reduce potential conflicts with publisher's sites, we encourage the use of wrapping any Javascript and CSS in closures or namespaces. Containing Javascript or CSS in a unique namespace or wrapping in a closure prevents this issue in overwriting variables or code on the publisher's site while also making the ad code more portable.

Examples of Animation Coded in CSS

Example 1: Simple Spin Animation

The following CSS-coded animation defines an element named "diagonal-slide" that moves from (0,0) to (100,100) within the duration of 5 seconds and repeats a total of 10 times:

```
div {
  animation-name: diagonal-slide;
  animation-duration: 5s;
  animation-iteration-count: 10;
}

@keyframes diagonal-slide {

  from {
      left: 0;
      top: 0;
  }

  to {
      left: 100px;
      top: 100px;
  }

}
```

Example 2: Expanding Element with CSS Transition Properties

The following CSS-coded animation defines a "div" that expands from 100x100 to 300x100 when hovered over (this example does not work with Internet Explorer). Included in this example is the use of a stack for multiple browsers dealing with a transition. Vendor prefixes are necessary when needing to handle all browsers (non-mobile); this is used through the application of a stack in your code:

-webkit-transition: CSSPROPERTY time ease;
-moz-transition: CSSPROPERTY time ease;
-o-transition: CSSPROPERTY time ease;
transition: CSSPROPERTY time ease;

```
div
{
	width:100px;
	height:100px;
	background:red;
	transition:width 2s;
	-moz-transition:width 2s; /* Firefox 4 */
	-webkit-transition:width 2s; /* Safari and Chrome */
	-o-transition:width 2s; /* Opera */
}

div:hover
{
	width:300px;
}
```

For more examples of CSS animation, refer to the W3C site (http://www.w3.org). The W3 organization defined HTML5 and provides plenty of resources to help encourage adoption.

Animation in Javascript
In general, developers using Javascript should adhere to industry-established and widely accepted Javascript best practices. This document refrains from defining Javascript best practices, but provides the following suggestions to consider:

- Optimize loops and timers
- Minimize DOM access
- Ensure responsiveness and performance
- Feature-detect rather than browser-detect, to more easily and efficiently check if functionality exists and can be utilized within the page
- Avoid the use of the `eval()` function, to avoid potential security risks
- Avoid global variables that may result in naming conflicts and cause code to break
- Account for syntax differences and requirements between browsers (particularly in IE)

4.1.2 Animation Frame Rate

Unlike Flash there is no true FPS setting within an HTML5 document. Establishing frame rate for HTML5 remains an open issue for IAB guidance. We recommend referring to publisher specs or discussing the specs with publisher partners to determine what they can allow. Frames per second for HTML5 doesn't impact browser performance in the same way it does in Flash, and there are ways to define Frames per second using a Javascript variable, but it is not required.

Animation methods like `setInterval` and `requestAnimationFrame` run animation at the allowable pace of the browser and are described as follows:

Applying Animation Method

After the animation time is determined apply one of the two animation methods, `setInterval` or `requestAnimationFrame`.

setInterval

Used as a fallback mechanism for older browsers (IE9 and earlier), setInterval calls an animation function repeatedly on each interval, which is defined within the `animationTime` variable, to simulate animation:

```
setInterval(animationFunction, animationTime);
```

requestAnimationFrame

This animation method should be used whenever possible as an alternative to `setInterval`. The `requestAnimationFrame` method doesn't allow the developer to specify an interval rate. Instead the method requests that the browser draw the animation at the next available opportunity rather than at a predetermined rate, allowing for animations to run at the highest FPS possible. Using this method, frame rate is based on the screen refresh rate, which is usually 60Hz.

requestAnimationFrame Feature Check

The creative should contain code that checks browser support of the `requestAnimationFrame` feature in order to safely use it. The following code example performs this essential check and if the feature is not supported, the ad will fall back to using `setTimeout`. Code for handling the cancellation of a `requestAnimationFrame` loop is included:

```
if (!window.requestAnimationFrame) {
      window.requestAnimationFrame = (window.webkitRequestAnimationFrame ||
      window.mozRequestAnimationFrame ||
      window.msRequestAnimationFrame ||
      window.oRequestAnimationFrame ||
      function (callback) {
            return window.setTimeout(callback, 60);
      });
};

if (!window.cancelRequestAnimationFrame) {
      window.cancelRequestAnimationFrame = (window.cancelAnimationFrame ||
      window.webkitCancelRequestAnimationFrame ||
      window.mozCancelRequestAnimationFrame ||
      window.msCancelRequestAnimationFrame ||
      window.oCancelRequestAnimationFrame ||
      window.clearTimeout);
};
```

4.1.3 Executing Animation

Outside of using CSS3-driven animation, animation can be executed in two specific ways within the browser. The first method, once popularly referred to as DHMTL, requires using Javascript to move and manipulate DOM elements. However, this method is not actually part of the HTML5 spec. The second method is included as part of the HTML5 spec and requires using Javascript, but instead of moving and manipulating DOM elements, it animates by moving and manipulating pixels within an HTML5 Canvas element.

Pros & Cons of Canvas and DOM Elements:
Using Javascript to execute animation takes advantage of Canvas and can improve performance. Canvas focuses on the high performance of 2D graphics, so its peak performance is generally higher than that of the DOM, which is more focused on content and interaction. Canvas also provides a more consistent API than the DOM. In some cases, Canvas and DOM elements can be combined to take advantage of the text handling abilities of the DOM with the animation and graphical capabilities of Canvas.

Both Canvas and DOM elements can be hardware accelerated and achieve incredibly smooth animations while simultaneously having to perform numerous and complex calculations. Tasks that would have otherwise been calculated by the main CPU can be offloaded to the GPU in the user's computer graphics adapter, yielding tremendous performance gains and also reducing resource consumption on mobile devices and tablets.

GPU Execution on DOM and Canvas Elements

The following technique can be used to force the GPU to optimize and execute 2D animations on both the DOM and Canvas elements; however, simply applying this transformation doesn't guarantee any increase in performance and results can be inconsistent. This technique is particularly useful for iOS, but may not work on future versions so we recommend testing for visible performance gains before applying. Results of the technique may only increase CPU processing and battery usage without increasing performance.

Apply the following CSS to your animated DOM element (in this case a div) or your Canvas element:

```
canvas {
      -webkit-transform : translateZ(0);
      -o-transform : translateZ(0);
      -moz-transform : translateZ(0);
      transform : translateZ(0);
}

div {
      -webkit-transform : translateZ(0);
      -o-transform : translateZ(0);
      -moz-transform : translateZ(0);
      transform : translateZ(0);
}
```

HTML5 Canvas Feature Support Check

In order to safely use HTML5 Canvas the creative should contain code that checks the browser for support of this feature. The following code sample is a way to check for Canvas support, and if the feature is not supported, then it will fall back to whatever content is contained within the Canvas tag, typically a static JPEG version of the ad

```
function canvasSupport(){
      return !!document.createElement('canvas').getContext;
};

if(!canvasSupport()){
      var fallback = document.getElementById('fallback');
      fallback.src = imagePath + 'backup.jpg';
      return;
};
```

Feature Detection

The general practice for HTML5 ads is to include code that detects whether features built into the ad are supported in the browser where it is served, and if not supported, offer an alternate experience for the ad to replace the experience that cannot execute because of the unsupported feature. Check the IAB Wiki: HTML5_for_Digital_Advertising_Resources for tools that can help you check for feature support in select browsers.

5 HTML5 Tools

There are a number of tools, utilities, and companies that can help you with HTML5 creation and adhering to the suggestions in these guidelines. A place to find a collection of tools is IAB Wiki: HTML5_for_Digital_Advertising_Resources.

In general, the toolsets for designers and developers will differ. Designers will need tools that focus on producing the creative assets and developers will need tools for the CSS, Javascript and HTML5 files.

5.1 For Designers

HTML5 Mobile Rich Media

Some of the most powerful tools are available from Rich Media ad vendors. These often provide a complete environment for ad creation using templates or components to build your ad and include ad-friendly features like video, geo-location, and mobile calls to action.

As digital advertising specialists, these companies already adhere to many of the best practices outlined in this document. Look for tools that:

- Generate universal tags for cross-platform delivery
- Support animations and transformations
- Import animations and custom HTML from other tools
- Provide components to build ads without hand-coding
- Support web standards and IAB MRAID & SafeFrame API's
- Deliver ads that are lightweight and optimized for various platforms

Image Creation Tools

In general, the tools you use today to create images will still be appropriate when developing for HTML5. The most important aspect is generating the output. Creatives should be saved as PNG, JPG, or GIF files with attention to file size and clarity. Expect to use other tools for image compression, optimization, and sprite creation. Also note that animated GIF images are not supported on many mobile devices.

HTML5 Animation Tools

When choosing an animation tool, give special consideration to the generated output. Some tools require additional Javascript libraries which makes the output too heavy.

- Can be used on your platform (Mac, PC, online)
- Generates small files without external Javascript
- Provides cross-platform support, especially for mobile
- Follows current standards for HTML5 and CSS3

- Follows advertising standards like the IAB's MRAID for advertising in-app

Flash Converters

There are tools that attempt to convert Flash files to HTML5. These have limitations, not all features can be converted and optimization may be limited. Look at these tools as a possibility to convert simple Flash files or to "jump-start" a larger conversion effort.

HTML5 Website Tools

Many website building tools are incorporating HTML5 features and provide a designer-friendly layout for drag-and-drop creation.

Because these are general-purpose tools, the generated HTML5 lacks some best practices for ad creation. For these tools, you should expect to do some hand-coding after the initial designs.

- Supports responsive layouts
- Provides mobile device previews
- Includes web fonts
- Has a code-view

5.2 For Developers

Browser Reference

As you consider cross-platform and cross-browser support, having a reference is essential. HTML5 is an emerging standard and support for CSS3 is not complete for all vendors. Consider a site like http://caniuse.com/ that is updated often to provide up-to-date information.

Javascript Libraries

Using Javascript libraries is a common practice in web-site building, but is not always the right approach for ad building. Look for libraries that do not conflict with publisher sites and can be optimized to include only the methods you use.

Code Compression

To serve quickly, look for code compression tools that can reduce file size. Better tools also provide obfuscation and can be incorporated with development or production tools to run automatically.

Image compression

Image optimization is often the most effective approach to reducing file size. Look for tools that provide lossless optimization, can work on a batch of files, and allow you to choose a compressed image from a preview.

Sprite Generators

These tools create a single image from several smaller images which is a great technique for limiting the number of connections and improving load times. Look for a tool that not only combines images to a single sprite, but also provides the CSS to display the portion of sprite needed in your ads.

Web Fonts Technology

Font foundries also provide fonts for the web and tools to include them. Look for plug-in support that make it easy to include web fonts in your other creative tools, and support for font sub setting where only the letters needed for your creative are downloaded to the device.

6 Terminology

Animation: A programmatically generated display of sequential frames or transitioning images, creating the illusion that objects in the image are moving. Not digital video, as it relates to this document.

Application Programming Interface (API): Application Programming Interface is a set of commands in a common language that programmers or developers use to communicate with a specific piece of software or hardware. Mobile ads delivered in apps use an API to communicate with the SDK that is built into the app.

Base64 Encoding: A method for converting image information into text (radix-64 representation) for pasting into a document instead of making additional server request to load the image.

Bitrates: a measure of bandwidth that indicates how much data is traveling from one place to another on a computer network. Bitrate is usually expressed in kilobits per second (kbps) or megabits per second (Mbps).

Canvas: an HTML5 element that is a resolution-dependent bitmap container used for rendering graphics, interactivity and animation dynamically through Javascript directly within the browser and without the need for any 3rd party plug-ins. Canvas provides a set of functions ("the canvas API") for drawing shapes, defining paths, creating gradients, applying transformations and more.

Code Minification: the practice of removing unnecessary characters from code to reduce its size, removing unnecessary spacing, and optimizing the CSS code; thus improving load times.

Central Processing Unit (CPU): the key component of a computer system, which contains the circuitry necessary to interpret and execute program instructions.

Cascading Style Sheets (CSS): language used for describing the presentation semantics (the look and formatting) of a document written in a markup language.

Degrade Gracefully: coding an ad unit so that when it's rendered in browser where select functionality is unsupported, the ad provides alternative features that allow the ad to display in a way that is still functional but without the need for unsupported features. Also referred to as, Graceful Degradation.

Document Object Model (DOM): The DOM is a W3C standard for accessing documents like XML and HTML. The HTML DOM defines the objects and properties of all HTML elements, and the methods interface to access them. Javascript can be used to move and manipulate DOM elements to create animation.

Frame Rate: The rate at which video frames or animated images display as the video or animated file executes, measured as the number of frames per second (fps).

Frames Per Second (FPS): metric used to indicate the frame rate of animated or video creative content.

Graphics Processing Unit (GPU): GPU handles graphical processing, decreasing the processing burden handled by the CPU.

Gzip: Automatic compression of creative assets for an ad when delivering from an ad server to a web page or application.

High Resolution Displays: screen displays that uses more pixels to cover the same physical area, also known as retina display.

Javascript Libraries: a library of pre-written Javascript that typically include functions for common tasks like animations, DOM manipulation, and Ajax handling. (Often called Javascript frameworks)

Kilobyte (KB): A multiple of the unit 'byte' for digital information, used to quantify computer memory or storage capacity equal to a 1,000 bytes (or technically, 2^10 = 1,024 bytes).

K-Weight: weight of a file measured in kilobytes.

Megabyte (MB): A multiple of the unit 'byte' for digital information, used to quantify computer memory or storage capacity equal to 1,000 kilobytes (or technically, 2^20 = 1,048,576 bytes).

Mobile Rich Media Ad Interface Definitions (MRAID): standardized set of commands, designed to work with HTML5 and Javascript, that developers creating rich media ads will use to communicate what those ads do (expand, resize, get access to device functionalities such as the accelerometer, etc) with the apps they are being served into. http://www.iab.net/mraid

Progressive Enhancement: when a creative developer uses features that are widely supported across browsers, but also develops an enhanced version using the newest HTML5 features for browsers that are compatible.

SafeFrame: SafeFrame 1.0 offers a solution that prevents external HTML content from accessing the hosting website and its sensitive data by framing and rendering the content from within a secondary domain.
An API enables communication between the webpage and the external content to allow for any rich interactions. http://www.iab.net/safeframe

Software Developer's Kit (SDK): a pre-packaged piece of code that developers can incorporate into their application to avoid having to develop it from scratch. For example SDKs from rich media vendors and networks are often implemented into the publisher's mobile app to handle advertising.

Sprite Sheets: a large image filled with smaller images. Putting all image assets for a creative into one sprite sheet reduces the server calls needed to load the images. Another use of sprite sheets is to create animation by displaying each of the smaller images in the correct order.

Scalable Vector Graphics (SVG): Defines graphics in XML format and can scale indefinitely to high-resolution displays without increasing file size.

Transcoded: the direct digital-to-digital data conversion of one encoding to another, such as for movie data files or audio files. Files such as video assets may need to be transcoded into a few different formats to ensure cross-browser functionality.

Video Suite (VSuite): a set of technical specifications and protocols for in-stream video ad formats that allow compliant ads to seamlessly play across multiple compliant publisher sites. This included IAB's Video Ad-Serving Template (VAST), Video Player-Ad Interface Definition (VPAID) and Video Multiple Ad Playlist (VMAP). http://www.iab.net/vsuite

WebViews: A container with a rendering engine that displays web content within an app environment. This is sometimes referred to as a "micro-browser".

7 Update Release Notes

2013.17.7 Update, Document Version 1.0.1

Section 2.4.1 ClickTag required an update to the code example for implementing a clickTag into the creative file. The original version stated, item highlighted in red indicates where the error occurred:
To format clickthrough URLs, use the "clickTag" variable for storing the click destination as shown in the following example:

```
var clickTag = "www.example.com" target= "_blank";
```

Changes have been applied to the new section, to view the revised section click: 2.4.1 ClickTag