Video Player-Ad Interface Definition (VPAID)

Released April 2009

This document has been developed by the IAB Digital Video Committee.

Document Version	1.1
Primary Author	Ryan Cunningham – Yahoo!
Contributing Author	Prabhakar Goyal – Yahoo!
Contributing Author	Ashish Gupta - Microsoft
Date	4/20/2009

About the IAB's Digital Video Committee:
The Digital Video Committee of the IAB is comprised of over 180 member companies actively engaged in the creation and execution of digital video advertising. One of the goals of the committee is to implement a comprehensive set of guidelines, measurement, and creative options for interactive video advertising. The committee works to educate marketers and agencies on the strength of digital video as a marketing vehicle. A full list of Committee member companies can be found at: www.iab.net/digital_video_committee

This document can be found on the IAB website at: http://www.iab.net/vpaid

IAB Contact Information:
Ryan Walker
Manager of Industry Services, IAB
212-380-4731
ryan@iab.net

Jeremy Fain
Vice President of Industry Services, IAB
212-380-4724
jeremy@iab.net

Table of Contents

1 Executive Summary

This document establishes a Video Player-Ad API definition (VPAID) that standardizes the communication between video players and in-stream video advertising as designed by the Digital Video Committee of the Interactive Advertising Bureau (IAB). This standard intends to meet the needs of emerging in-stream ad formats such as:

- Non-linear video ads
- Interactive video ads

The goal of the VPAID standard is to address known interoperability issues between publisher's video players and different ad servers. Today, video ads created using a specific ad platform can only run on publisher video players that are pre-integrated with that same platform. This issue is further exacerbated when the video ad is expressed in innovative formats (such as non-linear and interactive ads) that require a high level of communication and interaction between the ad and the video player.

This standard does not sit alone. Publishers and ad networks who wish to participate in the more liquid video ad environment should also adopt the following standards:

- *Digital Video Measurement Guidelines*
- *Digital Video Ad Format Guidelines and Best Practices*
- *Digital Video In-Stream Ad Metrics Definitions*
- *Digital Video Ad Serving Template (VAST)*

A publisher that adopts VPAID and the above standards will be able to render any type of video advertising trafficked from any video ad server that also adheres to the same standards.

While this document's purpose is to promote interoperability in the most common areas of the digital video landscape, the IAB continues to encourage creativity and innovation in video ad formats. As with all IAB guidelines, this document will be updated as the dynamic digital video advertising landscape progresses and new ad formats become more widely adopted.

2 Motivation

The lack of technology standards in video advertising is a growing concern for the IAB. Because of the lack of common technology frameworks, video ad implementation is often technology–specific and results in a lack of liquidity in the video ad marketplace. Advertisers who wish to run the latest and greatest ad formats are often frustrated that their creative spends for a campaign often only work on one video player and ad server combination. This hampers the industry's overall ability to adopt innovative ad formats that both provide better ROI to advertisers and enable a less intrusive experience to video content viewers. The end result is advertisers retreating to simple, traditional formats (such as pre-roll) that may be outperformed by more advanced formats. Publishers are also reluctant to accept 3rd party ad

serving when the publisher knows each integration project is platform-specific and not scalable.

With this API standard, the IAB hopes to address the following market inefficiencies for publishers, advertisers, and ad networks:

- Lack of common video ad supply technology, preventing video publishers from adopting 3rd party advertisements
- Lack of common technology standards for advertisers to develop against, thereby increasing cost of asset production
- Lack of video ad supply liquidity, increasing cost of integration with each publisher

3 Introduction

In-stream video advertisements, unlike other forms of display or banner ads, execute in the player environment. The player is responsible for making the ad call, and interpreting the ad response from the ad server. The player also provides the run-time environment for the ad and controls its lifecycle.

With the advent of advanced video advertisements, the relationship between the player and the ad has become more complex. Advanced ad formats provide an ad experience by interspersing ad parts with the content in interesting ways and require careful orchestration of content and ad play. The ad itself can run scripts and has variability based on the user interactivity and other run time context. The interplay of the video player, video content, and the ad require certain capabilities via an API from the player run-time.

To allow interoperability of advanced video ads across different player environments requires, at minimum, standardization at the following two levels:

- The ad server response should be understood by compliant players – this is covered by the VAST standard
- The advanced ad requires a standard player run-time expressed as a standard API – covered by this document and called VPAID

The following diagram depicts the basic video ad flow and the interface point where VPAID is applicable.

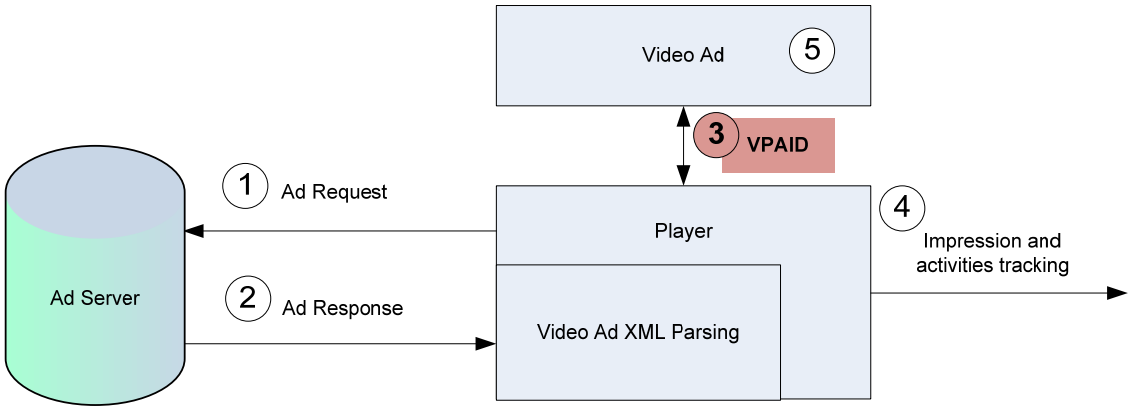

Figure 1: Basic Video Ad Flow

The video ad flow requires the following steps. (Note that there are many other details and variations, but this description has intentionally been kept simple):

1. The player makes an ad call to the ad server. IAB has not published a standard around the ad call request format (Though the VAST standard has an appendix which recommends parameter support needed for video advertising).
2. The ad server responds with an ad XML. The IAB VAST standard covers the format of video ad server response.
3. The player retrieves the ad media referred to in the ad XML and renders it. The advanced ad media typically requires a set of parameters from the player, and may interact with the player using an API. This standard covers the player and ad interaction.
4. The player or ad fires the impression and activity beacons. The IAB has published *Digital Video In-Stream Ad Metrics Definitions* and *Digital Video Ad Measurement Guidelines* that cover the measurement aspect of the video advertisement.
5. Ad rendering, behavior, and format specification is covered by the IAB's *Digital Video Ad Format Guidelines and Best Practices*

This standard defines a uniform run-time environment for advanced video advertisements, so that a compliant player can accept any compliant advertisement from any other party, thus enabling advanced video ad's interoperability across publishers. The following diagram depicts the level of video ad interoperability across publishers with different level of standards compliance.

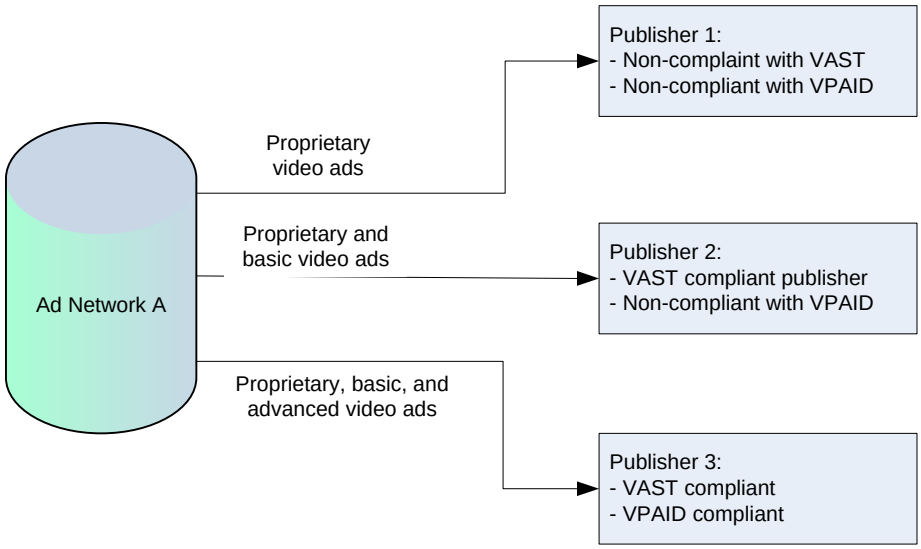

Figure 2: Video ad interoperability across publishers with different level of standards compliance
(all 3 Publishers shown are pre-integrated with Ad Network A's video ad rendering technology)

Publishers that are compliant with both the VAST and VPAID standards will be able to accept larger varieties of video ads from 3rd party ad servers and ad networks that are also compliant.

VPAID is most useful (and indeed necessary) for ads that are displayed concurrently with the video content, such as overlay ads, or for ads that must interrupt the playback of the video content as a result of user interactions with the ad, such as with interactive ads. While basic pre-roll or post-roll video ads need not use an API, having the framework in place allows new ad formats to be developed. In addition, pre-roll and post-roll ad formats can be extended to have a section that is present during video content playback without changing the Player-Ad framework.

No API Required	API Required
Pre-roll video	Pre-roll that runs script
Pre-roll video with companion banner	Click-to interactive overlay
Image or simple creative overlay	

3.1 Scope and Limitations

This standard only covers the interaction between the player run-time and a video ad that runs script, such as one build using the Adobe Flash technology. Furthermore, this standard only covers the interoperability between the ads and the player written using the same technology stack. For example, a compliant player written using Flash ActionScript 3 technology will be able to accept any compliant ad written in Flash ActionScript 3, but may not be able to accept compliant ad written using Silverlight technology and vice versa.

The mechanism to match technology of a compliant ad to technology of a compliant player, operationally or during ad serve time, is out of scope of this document.

This document is designed for any on-demand video player. It may be possible to use this specification for delivering ads in applications other than on-demand viewing such as live video streaming, downloadable video players, set-top boxes, etc, but those applications are explicitly beyond the scope of the current effort.

Though the specification allows for preloading of the ad content by separating out the ad loading from ad rendering, the actual logic to preload the ad and dealing with its ramification on impression or inventory counting is out of scope of this document.

The timing and method for counting the revenue impression and other advertisement related measurement specification is out of scope of this document. It is assumed that the Video Players will attempt to align impression tracking with the IAB *Digital Video Ad Measurement Guidelines*.

The specification provides provisions for time sequencing multiple parts of a single ad. But any business logic to sequence multiple advertisements, either from the same campaign or different campaigns, is explicitly out of scope of this specification.

Management of ad rules, including any business rules, which decide when to make an ad call and what type of ads to request is out of scope of this specification.

It should be noted that while this document uses the term "player" throughout, VPAID is not limited to an interface specifically between a video player and an ad – there can be other application interface layers so long as when combined the VPAID specification and interface are respected.

4 API Reference

4.1 Guiding Principles

VPAID was designed using the following guiding principles:

1. Generalized – the API needs to be generalized enough to be applicable to the next generation ad formats (for the foreseeable future)
    a. Allow player to position ads in time and space, when sufficient data is available
2. Simple – the API needs to be as simple as possible while still satisfying 1.
3. Shift the API burden on to the player – For example, simple ad formats do not require any API implementation.
4. Player in control – the API is designed in a way that the ad could request certain services from the player, but the player ultimately remains in control of the overall run-time environment. For example, the player has the ability to unload a misbehaving ad.

5. Keep the ad portable – API specification makes the attempt to keep the ad independent of the ad request parameters, XML schema used to traffic the ad, reporting beacon formats, etc. This is to allow the ad to be portable across the multiple ad platforms.
6. Consistency across technology implementations – Best effort to specify a single API across all technology implementations (AS2, AS3, Silverlight, JS bridges, etc.).

4.2 Notation

name : Type – Specifies a variable, property, or method name followed by data type or return data type.

{} – Specifies an object with name : Type properties.

Array<Type> – Specifies an array of a particular data type.

[] – Specifies optional parameters or data structure properties.

| – Specifies "or" for data structure properties.

* – Specifies repeat 0 or more times.

+ – Specifies repeat 1 or more times.

4.3 Player Requirements

The player must implement ad loading, check for presence of <VPAID>, and if present implement the correct VPAID version. The player must enforce recovery mechanisms should the ad fail to implement VPAID correctly, e.g. if AdStopped event is not received within some grace time period following a call to stopAd. The player should never allow uncaught exceptions to be thrown while handling VPAID events from the ad. See Implementation sections for more specific requirements.

4.4 Ad Requirements

If the ad implements VPAID, it must indicate the correct API version. The ad must implement all methods and properties listed, but can choose to do nothing or return values indicating that the method or property is not implemented. The ad should never allow uncaught exceptions to be thrown during calls into VPAID from the player. See Implementation sections for more specific requirements.

4.5 Ad Dependency on XML Format

The API specification is designed to isolate the ad, to the extent possible, from the schema of the XML used to deliver the ad. This allows a VPAID compliant ad to be reused without any modification across variety of publishers and networks, even if different XML formats are used to convey the ad. This isolation diminished to recode the ad as versions of the XML schema (e.g., *VAST [4]*) evolve. For example, the API specification explicitly avoids requiring passing

the whole XML received from the ad server to the ad, which would have required XML schema specific parsing logic in each ad.

This approach, at the same time, does not preclude externalizing some of the ad parameters into the ad XML, if the ad designer so chooses. There are several common usage examples of externalizing ad parameter in the XML for the advanced ads, including:

- Externalizing video URLs from the advanced ad SWF. This allows video of the ad to be encoded/updated independently of the ad development
- Parameterizing certain parts of the ads such as call-to-action text. Parameterization could be used to allow advanced functionality such as personalization and internationalization/localization of the ad based on serving context.

The specification allows a designated section of the XML to be passed to the ad. The *VAST [4]* XML specification already contains such an ad data section. It is assumed that the XML parsing is done by the player (or some component on the player), which has the responsibility of identifying the ad data section and extracting and passing that to the ad as specified in the *initAd()* API below.

4.6 Ad and Reporting Beacons

In this specification, we have kept the firing of the reporting beacons outside the ad. This is to keep the ad portable across as many ad platforms as possible:

- This approach keeps the ad independent of how the various beacons are represented within the XML schema.
- Different ad platforms differ in the way they handle beacons. This approach keeps the ad independent of these differences in the ad format.
- This also allows complex beacon handling (e.g., buffering and batching of non-essential events) to be coded at one place within the player, instead of repeating the same logic within each ad.

The API allows the ad to fire custom events, carrying the event id and additional data. The player can catch these events and map to the reporting beacon in the XML. The most common use case would be to map the event id from the ad to the id of all the beacons in the ad XML, and firing of all the matching beacons.

This approach does not preclude an ad from baking in the "fully resolved beacons" and firing them by itself. Fully resolved beacons are beacons which are known at the time of ad development and are not externalized within the ad XML.

The example of using events for reporting beacons is described in the AdImpression, etc. events section below.

4.7 Methods

All methods are called by the player, on the ad's VPAID member property object.

4.7.1 handshakeVersion

handshakeVersion(playerVPAIDVersion : String) : String

The player calls handshakeVersion immediately after loading the ad to indicate to the ad that VPAID will be used. The player passes in its latest VPAID version string. The ad returns a version string minimally set to "1.0", and of the form "major.minor.patch". The player must verify that it supports the particular version of VPAID or cancel the ad. All VPAID versions are backwards compatible within the same major version number (but not forward compatible). So if the player supports "2.1.05" and the ad indicates "2.0.23", the player can run the ad, but not in the reverse situation. Static interface definition implementations may require an external agreement for version matching. Dynamic implementations may use the handshakeVersion method call to determine if an ad supports VPAID. For dynamic languages, the ad or the player can adapt to match the other's version if necessary. A good practice is to always call handshakeVersion even if the version has been coordinated externally, in case the ad supports multiple versions and uses handshakeVersion to decide which to act on at runtime.

4.7.2 initAd

initAd(width : Number, height : Number, viewMode : String, desiredBitrate : Number[, creativeData : String][, environmentVars : String]) : void

After the ad is loaded and the player calls handshakeVersion, the player calls initAd to initialize the ad experience. The player may pre-load the ad and delay calling initAd until nearing the ad playback time, however, the ad does not load its assets until initAd is called. The ad sends the AdLoaded event to notify the player that its assets are loaded and it is ready for display. The player passes a width and height to indicate the display area for the ad. viewMode can be one of "normal", "thumbnail", or "fullscreen" to indicate the players current viewing mode, as defined by the player and ad publisher and may not be applicable to all ads. The player also passes a desired Bitrate in kbps (kilobits per second) that the ad may use when selecting the bitrate for any streaming content. creativeData is an optional parameter that can be used for passing in additional ad initialization data; for example, the extensions node of a *VAST [4]* response. enviromentVars is an optional parameter that can be used for passing implementation-specific runtime variables, URL-encoded name=value pairs separated by '&'. (see resizeAd below for more information on sizing)

4.7.3 resizeAd

resizeAd(width : Number, height : Number, viewMode : String) : void

Following a resize of the ad UI container, the player calls resizeAd to allow the ad to scale or reposition itself within its display area. The width and height always matches the maximum display area allotted for the ad, and resizeAd is only called when the

player changes its video content container sizing. For ads that expand or go into linear mode, the entire video content display area is given in the width height as these ads may take up that entire area when in linear or expanded modes. Also, the player should avoid using the built-in scaling and sizing properties or methods for the particular implementation technology. viewMode can be one of "normal", "thumbnail", or "fullscreen" to indicate the players current viewing mode, as defined by the player and ad publisher and may not be applicable to all ads. The player should never set the width, height, scaleX, or scaleY properties of the ad, but should mask the ad to the provided width and height. However, the player may set the x and y properties of the ad to position. As well, the ad may choose to mask itself to the width and height to ensure UI placed off screen is never visible.

4.7.4 startAd

startAd() : void

startAd is called by the player and is called when the player wants the ad to start displaying. The ad responds by sending an AdStarted event notifying the player the ad is now playing. An ad may not be restarted by the player by calling startAd + stopAd multiple times.

4.7.5 stopAd

stopAd() : void

stopAd is called by the player when it will no longer display the ad. stopAd is also called if the player needs to cancel an ad. However, the ad may take some time to close and clean up resources before sending an AdStopped event to the player.

4.7.6 pauseAd

pauseAd() : void

pauseAd is called to pause ad playback. The ad sends an AdPaused event when the ad has been paused. The ad must turn off all audio and suspend any animation or video. The player may use pause in order to then hide the ad by settings its display container's visibility. It the discretion of the ad whether to remove UI elements or just stop their animation and perhaps dim their brightness.

4.7.7 resumeAd

resumeAd() : void

resumeAd is called to continue ad playback following a call to pauseAd. The ad sends an AdPlaying event when the ad has resumed playing.

4.7.8 expandAd

expandAd() : void

expandAd is called by the player to request that the ad switch to its larger UI size. For example, the player may implement an open button that calls expandAd when clicked. (see adExpanded property below) The player may use the value of the adExpanded

property as well as an AdExpandedChange event to determine when to display an open or a close button, if required. expandAd may not be applicable to all ads.

4.7.9 collapseAd

collapseAd() : void

collapseAd is called by the player to request that the ad return to its smallest UI size. For example, the player may implement a close button that calls collapseAd when clicked and is displayed only when the ad is in an expanded state (see adExpanded property below). collapseAd may not be applicable to all ads.

4.8 Properties

All properties are accessed by the player on the ad's VPAID member property object. If the property is a get property, the ad writes to the property and the player reads from the property. If the property is set property, the player writes to the property and the ad reads from the property.

4.8.1 adLinear

get adLinear : Boolean

The adLinear Boolean indicates the ad's current linear vs. non-linear mode of operation. adLinear when true indicates the ad is in a linear playback mode, false nonlinear. The player checks adLinear initially as well as each time an AdLinearChange event is received and updates its state according to the particulars of the ad placement. While the ad is in linear mode, the player has the content video paused. If adLinear is set to true initially and the ad is designated as a pre-roll (defined externally), the player may choose to delay loading the content video until near the end of the ad playback.

4.8.2 adExpanded

get adExpanded : Boolean

The adExpanded Boolean value indicates whether the ad is in a state where it occupies more UI area than its smallest area. If the ad has multiple expanded states, all expanded states show adExpanded being true. An AdExpandedChange event indicates the value has changed, but the player may check the property at any time. If ad is statically sized adExpanded is set to false.

4.8.3 adRemainingTime

get adRemainingTime : Number

The player may use the adRemainingTime property to update player UI during ad playback. The adRemainingTime property is in seconds and is relative to the time the property is accessed. The player may periodically poll the adRemainingTime property, but should always check it when receiving an AdRemainingTimeChange event. If not implemented, returns -1. If unknown, returns -2. The player may use the

adRemainingTime property value to display a countdown timer or other ad duration indicator.

4.8.4 adVolume

get adVolume : Number

set adVolume : Number

The player uses the adVolume property to attempt to set or get the ad volume. The adVolume value is between 0 and 1 and is linear. The player is responsible for maintaining mute state and setting the ad volume accordingly. If not implemented the get always returns -1. If set is not implemented, does nothing.

4.9 Dispatched Events

All events are dispatched from the ad to the player. It should be noted that several events are raised as a result of method calls into the ad. Event handlers in the player may in turn call ad methods. Players must be written in a way to prevent a stack overflow caused by calling back into methods as a result of an event.

4.9.1 AdLoaded

The AdLoaded event is sent by the ad to notify the player that the ad has finished any loading of assets and is ready for display. The ad does not attempt to load assets until the player calls the init method.

4.9.2 AdStarted

The AdStarted event is sent by the ad to notify the player that the ad is displaying.

4.9.3 AdStopped

The AdStopped event is sent by the ad to notify the player that the ad has stopped displaying, and all ad resources have been cleaned up.

4.9.4 AdLinearChange

The AdLinearChange event is sent by the ad to notify the player that the ad has changed playback mode. The player must get the adLinear property and update its UI accordingly. See the adLinear property for more information.

4.9.5 AdExpandedChange

The AdExpandedChange event is sent by the ad to notify the player that the ad's expanded state has changed. The player may get the adExpanded property and update its UI accordingly. (see expandAd method above)

4.9.6 AdRemainingTimeChange

The AdRemainingTimeChange event is sent by the ad to notify the player that the ad's remaining playback time has changed. The player may get the adRemainingTime property and update its UI accordingly. The ad only sends this event when the adRemainingTime changes relative to the current time, meaning if the ad extends or

contracts its overall display time the event will be sent, but if the overall display time is not changed no event will be sent. The adRemainingTime property can be read once and the player set a timer based on it that it need not update unless receiving an AdRemainingTimeChange event.

4.9.7 AdVolumeChange

The AdVolumeChange event is sent by the ad to notify the player that the ad has changed its volume, if the ad supports volume. The player may get the adVolume property and update its UI accordingly.

4.9.8 AdImpression

The AdImpression event is used to notify the player that the user-visible phase of the ad has begun. The AdImpression event may be sent using different criteria depending on the type of ad format the ad is implementing. For a linear mid-roll the impression should coincide with the AdStart event. However, for a non-linear overlay ad, the impression will occur when the invitation banner is displayed, which is normally before the ad video is shown. This event matches that of the same name in *Digital Video In-Stream Ad Metrics Definitions [2]*, and must be implemented to be IAB compliant.

4.9.9 AdVideoStart, AdVideoFirstQuartile, AdVideoMidpoint, AdVideoThirdQuartile, AdVideoComplete

These five events are sent by the ad to notify the player of ad's video progress. They match the *VAST [4]* events of the same names, as well as the "Percent complete" events in *Digital Video In-Stream Ad Metrics Definitions [2]*, and must be implemented to be IAB compliant. These strictly apply to the video portion of the ad experience, if any.

4.9.10 AdClickThru

The AdClickThru event is sent by the ad when a click thru occurs. Parameters can be included to give the player the option for handling the event. Three parameters are included with the event, String url, String id and Boolean playerHandles. If playerHandles is true, the player must handle the event by opening a new browser window to the url, false, the ad will handle the event. The url can be used for the ad to specify the click thru url, and the id can be used for tracking purposes. This event matches that of the same name in *Digital Video In-Stream Ad Metrics Definitions [2]*, and must be implemented to be IAB compliant.

4.9.11 AdUserAcceptInvitation, AdUserMinimize, AdUserClose

The AdUserAcceptInvitation, AdUserMinimize and AdUserClose events are sent by the ad when it meets the requirement of the same names as set in *Digital Video In-Stream Ad Metrics Definitions [2]*. Each of these is a reporting by the ad to the player of a user-initiated action. The player may use these to report externally, but does not take any other action. The ad must implement the UI actions but also fire these events to notify the player.

4.9.12 AdPaused, AdPlaying

The AdPaused and AdPlaying events are sent in response to method calls to pauseAd and resumeAd, respectively, to indicate the action has taken effect. See pauseAd and resumeAd method descriptions for more detail.

4.9.13 AdLog

The AdLog event is optionally sent by the ad to the player to relay debugging information, in a parameter String message. It is not required that the ad provide any AdLog events, but may be convenient for player engineers to help debug particular ads.

4.9.14 AdError

The AdError event is sent when the ad has experienced a fatal error. Before the ad sends AdError it must clean up all resources and cancel any pending ad playback. The player must remove any ad UI, and recover to its regular content playback state. The parameter String message is included for more specific information to be passed to the player.

4.10 Sequence Diagram

4.11 Error handling and timeouts

Any fatal errors detected by the ad initiate an AdError event that cancels ad playback and returns the player to content playback state. As well, the player implements a timeout mechanism for the following:

1. Request of ad file to ad file successfully loaded
    a. Recovery action: cancel ad load, skip ad
2. Calling initAd and receiving AdLoaded event
    a. Recovery action: remove ad from UI, skip ad
3. Calling startAd to receiving AdStarted event
    a. Recovery action: call stopAd, remove ad from UI, skip ad
4. Receiving AdLinearChange event with adLinear set to true to receiving AdLinearChange event with adLinear set to false (excluding ad pause/resume time)
    a. Recovery action: call stopAd, remove ad from UI
5. Calling stopAd and receiving AdStopped event
    a. Recovery action: remove ad from UI

5 Security

VPAID was designed to allow for unidirectional scripting, where possible. All VPAID methods and properties are on the VPAID member of the ad object, allowing player to ad scripting. However, only events can be sent from ad to player. For client platforms such as Flash, this allows unidirectional scripting from player to ad but prevents the ad from scripting into the player or using a JavaScript bridge to attack the web page. See Implementation sections for more detail for each client platform.

6 Example Ads

6.1 Clickable Pre-roll

Clickable ads are a video pre-roll format, with an additional interactive overlay. The overlay can either be text or an image, and when the user clicks on the ad, they will be taken to a page the advertiser specifies. This page will open in a separate window.

With VPAID:

1) player calls handshakeVersion("1.0") -> "1.0"
2) player gets adLinear = true, delays content video loading
3) player calls initAd(400, 300, "normal", 500)
4) as sends AdLoaded
5) player calls startAd
6) ad sends AdStarted
7) player gets adRemainingTime*
8) [set adVolume | pauseAd...resumeAd | resizeAd]*
9) player buffers content video, player determines timing
10) ad sends AdStopped because it has completed
11) ...player plays content video

Without VPAID:

Use a linear pre-roll with pre-defined duration.
-no interactivity that could extend the ad duration
-potential scaling problems
-no volume control, position info, etc. if packaged as an advanced ad

6.2 Companion Banner

This format is a video ad, with the interactive section external to the video player. Because the video area is not clickable, the companion banner is necessary to determine if the user has interacted with the ad.

VPAID is not required for this use case

Without VPAID:

Use a linear video ad plus a companion banner.
-no interactivity that could extend the ad duration
-potential scaling problems
-no volume control etc. if packaged as an advanced ad

6.3 Overlay Banner

The overlay format displays the sponsor images over the video during the video playback. There may be rules added related to the banner display timing as well as the position.

With VPAID:

1) player calls handshakeVersion("1.0") -> "1.0"
2) player gets adLinear, = false, starts loading content video
3) player calls initAd(80, 300, "normal", 500)
4) ad sends AdLoaded
5) ... player plays content video, may be before 4
6) player calls startAd()
7) ad sends AdStarted
8) player calls stopAd() when ad display time has elapsed
9) ad sends AdStopped
10) ... content video continues to play

Without VPAID:

Use a non-linear ad with defined duration. Player must define ad position.
-no interactivity that could extend the ad duration
-potential scaling problems
-no volume control, etc. if packaged as an advanced ad

6.4 Overlay Banner with click-to-linear video ad

The overlay format displays the sponsor images over the video during the video playback. If the user clicks on the banner, the ad shows a linear video ad followed by the ad disappearing. If the user does not click, the overlay banner displays until its display time has elapsed.

With VPAID:

1) player calls handshakeVersion("1.0") -> "1.0"
2) player gets adLinear, = false, starts loading content video
3) player calls initAd(400, 300, "normal", 500)
4) ad sends AdLoaded
5) ... player plays content video, may be before 4
6) player calls startAd()
7) ad sends AdStarted
8) if user clicks
    a. ad sends AdLinearChange, adLinear = true
    b. player pauses content video
    c. video ad plays, completes
    d. ad sends AdLinearChange, adLinear = false
    e. player continues content video playback
9) else
    a. player calls stopAd() when ad display time has elapsed
10) ad sends AdStopped
11) ... content video continues to play

Without VPAID:

Not possible

7 ActionScript 3 Implementation

7.1 API Specifics

The loaded ad swf should be loaded into its own ApplicationDomain and therefore be treated as a * data type. Both the ad and the player need a VPAID interface (IVPAID) definition as they are running in separate ApplicationDomains so the interface cannot be shared. The player accesses VPAID by calling getVPAID on the ad object. For calling safety the player may choose to wrap the returned * object with a class that implements VPAID explicitly, as shown below. The VPAID class must extend EventDispatcher. The player should call addEventHandler on the VPAID object using events with String names as listed in the events section above, e.g. "AdStarted". See Custom events section below for more explanation on events. Stage.frameRate should be passed to initAd in the environmentVars parameter, e.g. "frameRate=15".

```
package
{
 public interface IVPAID
 {
  // Properties
  function get adLinear() : Boolean;
  function get adExpanded() : Boolean;
  function get adRemainingTime() : Number;
  function get adVolume() : Number;
  function set adVolume(value : Number) : void;
  // Methods
  function handshakeVersion(playerVPAIDVersion : String) : String;
  function initAd(width : Number, height : Number, viewMode : String, desiredBitrate : Number, creativeData :
String, environmentVars : String) : void;
  function resizeAd(width : Number, height : Number, viewMode : String) : void;
  function startAd() : void;
  function stopAd() : void;
  function pauseAd() : void;
  function resumeAd() : void;
  function expandAd() : void;
  function collapseAd() : void;
 }
}

package
{
 // Player wrapper for untyped loaded swf
 public class VPAIDWrapper extends EventDispatcher implements IVPAID
 {
  private var _ad:*;

  public function VPAIDWrapper(ad:*) {
   _ad = ad;
  }

  // Properties
```

```
    public function get adLinear():Boolean {
      return _ad.adLinear;
    }
    public function get adExpanded():Boolean {
      return _ad.adExpanded;
    }
    public function get adRemainingTime():Number {
      return _ad.adRemainingTime;
    }
    public function get adVolume():Number {
      return _ad.adVolume;
    }
    public function set adVolume(value:Number):void {
      _ad.adVolume = value;
    }

    // Methods
    public function handshakeVersion(playerVPAIDVersion : String):String {
      return _ad.handshakeVersion(playerVPAIDVersion);
    }
    public function initAd(width:Number, height:Number, viewMode:String, desiredBitrate:Number,
creativeData:String, environmentVars : String):void {
      _ad.initAd(width, height, viewMode, desiredBitrate, creativeData, environmentVars);
    }
    public function resizeAd(width:Number, height:Number, viewMode:String):void {
      _ad.resizeAd(width, height, viewMode);
    }
    public function startAd():void {
      _ad.startAd();
    }
    public function stopAd():void {
      _ad.stopAd();
    }
    public function pauseAd():void {
      _ad.pauseAd();
    }
    public function resumeAd():void {
      _ad.resumeAd();
    }
    public function expandAd():void {
      _ad.expandAd();
    }
    public function collapseAd():void {
      _ad.collapseAd();
    }

    // EventDispatcher overrides
    override public function addEventListener(type:String, listener:Function, useCapture:Boolean=false,
priority:int=0, useWeakReference:Boolean=false):void {
      _ad.addEventListener(type, listener, useCapture, priority, useWeakReference);
    }
    override public function removeEventListener(type:String, listener:Function,
useCapture:Boolean=false):void {
      _ad.removeEventListener(type, listener, useCapture);
```

```
    }
    override public function dispatchEvent(event:Event):Boolean {
     return _ad.dispatchEvent(event);
    }
    override public function hasEventListener(type:String):Boolean {
     return _ad.hasEventListener(type);
    }
    override public function willTrigger(type:String):Boolean {
     return _ad.willTrigger(type);
    }
  }
}
```

7.2 Custom events

As with the VPAID interface itself, event class definitions are not shared between the player and the ad. The below class definition can be defined in both the player and the ad, but since they are in separate ApplicationDomains it cannot be shared. The ad must create a flash.events.Event derived class that implements a data property getter as shown below.

```
package
{
 import flash.events.Event;

 public class VPAIDEvent extends Event
 {
   public static const AdLoaded : String       = "AdLoaded";
   public static const AdStarted : String      = "AdStarted";
   public static const AdStopped : String      = "AdStopped";
   public static const AdLinearChange : String  = "AdLinearChange";
   public static const AdExpandedChange : String = "AdExpandedChange";
   public static const AdRemainingTimeChange : String= "AdRemainingTimeChange";
   public static const AdVolumeChange : String  = "AdVolumeChange";
   public static const AdImpression : String    = "AdImpression";
   public static const AdVideoStart : String    = "AdVideoStart";
   public static const AdVideoFirstQuartile : String= "AdVideoFirstQuartile";
   public static const AdVideoMidpoint : String  = "AdVideoMidpoint";
   public static const AdVideoThirdQuartile : String= "AdVideoThirdQuartile";
   public static const AdVideoComplete : String  = "AdVideoComplete";
   public static const AdClickThru : String     = "AdClickThru";
   public static const AdUserAcceptInvitation : String= "AdUserAcceptInvitation";
   public static const AdUserMinimize : String   = "AdUserMinimize";
   public static const AdUserClose : String     = "AdUserClose";
   public static const AdPaused : String       = "AdPaused";
   public static const AdPlaying : String      = "AdPlaying";
   public static const AdLog : String          = "AdLog";
   public static const AdError : String        = "AdError";

   private var _data:Object;

   public function VPAIDEvent(type:String, data:Object=null, bubbles:Boolean=false,
cancelable:Boolean=false) {
```

```
    super(type, bubbles, cancelable);
    _data = data;
   }
   public function get data():Object {
    return _data;
   }
  }
}
// sample ad dispatch call from a function within ad's VPAID class
dispatchEvent(new VPAIDEvent(VPAIDEvent.AdStarted));
dispatchEvent(new VPAIDEvent(VPAIDEvent.AdClickThru,
 {url:myurl,id:myid,playerHandles:true}));
```

The player uses addEventListener with a handler function that that receives a * typed parameter that will be the custom event. To continue the above example:

```
public function onAdClickThru(event:*) : void
{
 trace("Ad url is: " + event.data.url);
}
_VPAID.addEventListener(VPAIDEvent.AdClickThru, onAdClickThru);
```

7.3 Security

To implement unidirectional scripting in ActionScript 3, use Security.allowDomain("<playerdomain or *>"). The ad swf must also be served from a domain where /crossdomain.xml allows the ad swf to be loaded by the player domain or *. The player should load the ad swf into a separate security and application domain.

8 ActionScript 2 Implementation

8.1 API Specifics

The loaded ad swf should be treated as an un-typed data type, its getVPAID method called to return the VPAID object as un-typed data type. VPAID methods and properties are accessed directly on the returned VPAID object. For calling safety the player may choose to wrap the un-typed VPAID object with a class that implements VPAID explicitly. The VPAID class should use the ActionScript 2 class EventDispatcher. The player should call addEventHandler on the VPAID property using events with String names as listed in the events section above, e.g. "STARTED". Note, it is especially important that the player never set the width, height, scaleX, or scaleY properties of the ad directly or risk having unpredictable scaling effect on the ad. The player should call resize instead.

8.2 Custom Events

Custom events in AS2 are implemented similar as in AS3 using the EventDispatcher class, although the event can be simply an Object with the correct properties, e.g.:


```
// sample ad dispatch call from a function within VPAID class
dispatchEvent({target:this, type:"AdClickThru", data:{url:myurl, id:myid, playerHandles:true}});
```

8.3 Security

To implement unidirectional scripting in ActionScript 2, use Security.allowDomain("<playerdomain or *>"). The ad swf must also be served from a domain where /crossdomain.xml allows the ad swf to be loaded by the player domain or *. The player should load the ad swf into a separate security and application domain.

9 Silverlight Implementation

9.1 API Specifics

Silverlight ad will expose the above APIs to support the communication from player. As it's not very easy to access getVPAID method on ad object, the ad object will implement "IVPAID" interface. The player can determine if the ad implements "IVPAID" interface and if it does, will understand that the ad is VPAID compliant ad. Player/Ad combination may decide to use this VPAID interface as unsafe but the recommended way would be to use a type safe interface which defines the above APIs. This type safety can be achieved by using the binary IVPAID interface which the ad can implement. Player will also use this binary interface to check if the ad implements the interface and thus will be sure of VPAID compliant ad. More information about this interface can be found athttp://www.iab.net/iab_products_and_industry_services/508676/508950/vpaid).

Following is the interface which VPAID-compliant Silverlight ad will implement (based on above explanation of VPAID.

```
public interface IVpaid {

        #region VPAID Methods

        string HandshakeVersion(string version);
        void InitAd(double width, double height, string viewMode, int desiredBitrate, string
creativeData, string environmentVariables);
        void StartAd();
        void StopAd();
        void ResizeAd(double width, double height, string viewMode);
        void PauseAd();
        void ResumeAd();
        void ExpandAd();
        void CollapseAd();

        #endregion

        #region VPAID Properties

        bool AdLinear { get; }
```



```
        bool AdExpanded { get; }
        TimeSpan AdRemainingTime { get; }
        double AdVolume { get; set; }

        #endregion

        #region VPAID Events

        event EventHandler AdLoaded;
        event EventHandler AdStarted;
        event EventHandler AdStopped;
        event EventHandler AdPaused;
        event EventHandler AdResumed;
        event EventHandler AdExpandedChanged;
        event EventHandler AdLinearChanged;
        event EventHandler AdVolumeChanged;
        event EventHandler AdVideoStart;
        event EventHandler AdVideoFirstQuartile;
        event EventHandler AdVideoMidpoint;
        event EventHandler AdVideoThirdQuartile;
        event EventHandler AdVideoComplete;
        event EventHandler AdUserAcceptInvitation;
        event EventHandler AdUserClose;
        event EventHandler AdUserMinimize;
        event EventHandler<ClickThroughEventArgs> AdClickThru;
        event EventHandler<VpaidMessageEventArgs> AdError;
        event EventHandler<VpaidMessageEventArgs> AdLog;
        event EventHandler AdRemainingTimeChange;
        event EventHandler AdImpression;

        #endregion
}
```

9.2 Custom Events

```
#region Event Argument classes

public abstract class VpaidMessageEventArgs : EventArgs {
        public abstract string Message { get; }
}

public abstract class ClickThroughEventArgs : EventArgs {
        public abstract string Url { get; }
        public abstract string Id { get; }
        public abstract bool PlayerHandles { get; }
}

#endregion
```

9.3 Security

The Silverlight ad XAP must be served from a domain where /crossdomain.xml allows the ad XAP to be loaded by the player domain or *.

10 JavaScript Bridge Implementation

10.1 API Specifics

VPAID may be exposed to JavaScript by creating an ad wrapper using the same implementation technology as the ad. The wrapper may expose VPAID to JavaScript by forwarding JavaScript method calls and property access on the wrapper object to the ad via VPAID, as well as dispatching events received in the wrapper from the ad via VPAID out to JavaScript as function calls. Object serialization must be dealt with using the implantation technology's capabilities. As a convention, property access should be implemented using functions "getVPAIDProperty(propertyName)" and "setVPAIDProperty(propertyName, value)" and events from ad to player be functions named like "VPAIDAdLoaded".

10.2 Security

Security for use of VPAID with JavaScript is dictated by the rules of the particular implementation technology and its JavaScript bridge.

11 References

1. Digital Video Ad Format Guidelines and Best Practices
2. Digital Video In-Stream Ad Metrics Definitions
3. Digital Video Ad Measurement Guidelines
4. IAB video ad serving template (VAST)

12 Appendix A: Glossary

Video Ad – Advertisement which is displayed within the context of video content play. Note that the key part of the definition is that the content is video. The advertisement itself may or may not be a video.

Advanced Video Ad – Video advertisement with programming logic that enables user interactivity, interaction with the video content play, or other advanced features.

Companion Ad – Commonly text, display ad, rich media, or skin that wrap around the video experience. These ads come in a number of sizes and shapes, and typically run alongside or surrounding the video player.

Linear Video Ad – The ad is presented before, in the middle of, or after the video content is consumed by the user, in very much the same way a TV commercial can play before, during or after the chosen program.

Non-linear Video Ad – The ad runs concurrently with the video content so the users see the ad while viewing the content. Non-linear video ads can be delivered as text, graphical ads, or as video overlays.

Post-roll – a Linear Video ad spot that appears after the video content completes.

Pre-roll – a Linear Video ad spot that appears before the video content plays.

VAST (Video Ad Serving Template) – IAB-defined XML document format describing an ad to be displayed in, over, or around a Video Player.

Video Player – Environment in which in-stream video content is played. The Video Player may be built by the publisher or provided by a vendor.

13 Appendix C: Future Considerations

1. Add AdInteraction event: The AdInteraction event is used by the ad to report user interactivity information to the player. This event allows publisher players to collect and report user interaction metrics associated with the advanced video advertisement. It is the publisher player's responsibility to encapsulate information carried by this event into the beacon format supported by the publisher reporting system. The event carries following generalized data;
    a. eventType : String
    b. eventValue : String
    c. [Time : Number] (time since the ad load, in ms, optional)
    d. [Seq : Number] (sequence number of the event, optional, 0 based).
2. Create and include links to sample ads in both Silverlight and Flash.

