DIGITAL AUDIO AD SERVING TEMPLATE (DAAST)

VERSION 1.0

Release Date: October 30, 2014

This document has been developed for the IAB Audio Committee

The Digital Audio Ad Serving Template (DAAST 1.0) specification was created by a working group of volunteers from 28 IAB member companies.

The Digital Audio Ad Serving Template (DAAST) Working Group was led by:

- Chris Doe, VINDICO
- Benjamin Masse, Triton Digital

The following IAB member companies contributed to this document:

Abacast
Adswizz
Aha Radio
Apple Inc. (iAd)
CBS Interactive
Clear Channel Media and Entertainment
comScore
Dial Global
Ernst & Young
Goodway Group
Google
Katz 360
Microsoft Advertising
Mirror Image Internet
National Public Media
NRG Media
Pandora
PROXi Digital
SET
Smartclip
Spotify
TargetSpot
Triton Digital
TuneIn
Univision Communications Inc,
VINDICO
Westwood One
XappMedia, Inc.
ZEDO

The IAB leads on this initiative were Jessica Anderson and Katie Stroud.

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

ABOUT THE IAB'S AUDIO COMMITTEE

The Audio Committee's Mission is to establish industry guidance by creating standards and best practices to help bring clarity to the Audio marketplace. The Committee will strive to educate marketers and agencies on the value of Audio as a powerful and effective advertising medium.

http://www.iab.net/audio_committee

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

Table of Contents

Executive Summary

The Digital Audio Ad Serving Template (DAAST) version 1.0 specification is the IAB's first technical solution to addressing a fragmented audio advertising marketplace. With widespread adoption of DAAST 1.0, the audio advertising community can begin to serve and track audio ads in a digital audio stream in much the same way that ads are tracked in digital video.

DAAST is rooted in the same digital ad serving technology used in digital video. The IAB Digital Video Ad Serving Template (VAST) was used as a model for developing DAAST. Although actual adoption numbers are not known, VAST is widely implemented in nearly all video players that monetize their content with digital advertising. The adoption of VAST has paved the way for exponential growth in digital video. With DAAST, we hope to achieve the same kind of growth in digital audio.

Today, advertising in digital audio is operationally challenged by trying to integrate audio advertising partners that all use different proprietary code for receiving and executing audio ads. In addition to a variety of audio advertising platforms, the range of audio players and devices is extensive. Implementing the common specifications available in DAAST, digital audio advertisers and partners can more efficiently exchange audio ads across platforms.

But the vision for DAAST extends far beyond efficiencies in digital audio.

As the digital marketplace strives to achieve an omni-channel approach to reaching their audiences, DAAST becomes a step toward making this possible. With the similarities between digital video and digital audio, some companies were already attempting to use VAST to serve and execute digital audio ads. However, digital audio faces certain challenges that are more prevalent in audio distribution than in video.

One such challenge is the proliferation of so-called "thin clients"— devices that can only execute streaming content and have no way to identify when an ad plays, much less to track it. Digital video is increasingly experiencing the same challenge. Developing a specification that addresses this and other challenges in audio brings the industry closer to developing a specification that other media channels can use. A single media ad serving template can easily extend from the templates used in video and audio once the nuances for each are worked out.

However, before moving ahead, digital audio needs a solution today. As streaming audio platforms adopt the DAAST guideline and specification, the digital audio marketplace can grow and the industry can begin to determine where adjustments need to be made in order to develop a solution that extends to video and other forms of digital media.

Intended Audience

Anyone interested in serving and executing ads dynamically within audio content can benefit from being familiar with these guidelines, but implementation details are targeted toward audio player developers and device manufacturers, organizations that monetize audio content with advertising, and any technology companies that develop and serve ads designed for digital audio placement. Specifically, audio player and device engineers and product managers should use this document as a guide when implementing technology designed to support an IAB-compliant digital audio ad response.

1 General Overview

In general, digital advertising depends on browser technology to display ads along with other content on webpages. Browsers such as Internet Explorer, Google Chrome, Firefox, and Safari use hypertext markup language (HTML) to communicate with servers and other systems in the ad supply chain that store, serve, display, and track digital ads.

Browser capability is an important component of digital advertising. While the proliferation of devices continues to grow, HTML remains the common language for communication within the ecosystem. However, audio content, much like video content, is framed within code that makes up a player. The code used to build these players is unique to the content publisher. While one publisher may use JavaScript to build its player, another may use Adobe Flash™. Even if two different publishers use the same coding language to build their players, they each use their own instance of the language with similar functions and variables that are likely named differently and function slightly differently from player to player.

Without a common interface and guidelines for making use of HTML, marketers have to develop multiple ads, each designed uniquely for each player it might play in. In this kind of fragmented marketplace, publishers suffer from a lack of demand and limited options for monetizing their content.

The IAB addressed this problem in digital video with an interface and guideline known as the IAB Video Ad Serving Template (VAST) and intends to address digital audio advertising using a similar approach with the Digital Audio Ad Serving Template (DAAST) specified in this document.

1.1 Rooting Digital Audio in Technology for Video

The IAB Video Ad Serving Template (VAST) revolutionized advertising in digital video and continues to contribute to growth and innovation in digital video. Taking away the visual component of video, today's digital audio marketplace looks a lot like video did before VAST was introduced.

Before VAST, video struggled in a fragmented market in which ads had to be specially developed for each player it used in any digital video campaign. The introduction of VAST enabled ad developers to define ad creative that could be easily delivered to any VAST-enabled player.

Today, audio suffers a similar dilemma; players are all built differently, contributing to a fragmented market for ad delivery that stagnates digital audio growth. Vendors for audio ad delivery and some audio players have already turned to VAST to help reduce friction in audio advertising.

Building a template on something that is already working helps to establish a foundation for technology that the industry can readily adopt. Audio is different than video, however; and certain provisions are needed in an audio ad serving template that allows for the unique technology of audio.

The Digital Audio Ad Serving Template (DAAST) is a step forward for audio, but also a step forward for ad serving in general. If adjustments made for DAAST can accommodate audio in a video ad serving medium, then a future version of an ad serving template can accommodate ad serving in mediums that include video, audio, functional applications, games, and perhaps even connected TVs.

1.2 Audio Players and Devices

Audio players may be a package of code that can be installed on select devices or operate in a browser. However, in many cases the audio player is the device such as in the case of a car stereo or portable audio device.

Audio player software and devices along with their capabilities play a big role in how ads are served and tracked in digital audio. Players can range from full-featured, such as a browser-based player or device-native application, complete with visual components, to a limited, or "thin," client such as a car stereo with only the ability to play an audio stream.

The following list describes some different players and their capabilities:

Browser-based audio players
There are two types of browser-based audio player: the single-feed station and the multiple-feed station.

The single-feed radio station represents a single station that plays its own licensed or syndicated content. Examples of single feed radio stations include stations like Pandora, news stations, sport stations, music stations of a specific genre, and other branded stations. These radio stations may have the ability to display image ads within the player, banner ads on the webpages where they're implemented, and track ads using tracking pixels. These players represent the full-featured players that are nearly as functional as a browser-based video player.

The multiple-feed radio station, often known as "aggregators," is a browser-based audio player that hosts multiple radio stations. The player itself has no visual component, but the webpage on which it's implemented has the ability to display banner ads.

In most cases, the audio content publisher sends content to the aggregator without any control over the ad experience. And while the aggregator has the ability to display banners, the audio content publisher that sent content to the aggregator has no way of tracking how the ad experience is processed within the aggregator. If the audio publisher wants the aggregator to display a companion ad, the ad may have to be sent ahead of time and embedded within the aggregator's server.

Reducing friction in a multiple-feed radio station involves a solution that enables audio content publishers to control and track the ad experience within the content sent to aggregators. One such solution may include adapting the IAB Video Ad Multi-Playlist (VMAP) specification to audio. VMAP enables video content distributors to indicate ad breaks within their content. The video aggregator then processes ad breaks as indicated and sends tracking information to applicable parties. For more information about VMAP, please visit http://iab.net/vmap.

Examples of multiple-feed radio stations include TuneIn and iHeartRadio.

Desktop Applications
Some audio players can be downloaded to your desktop computer. These players may or may not have a visual component, and while they connect to the Internet, they don't necessarily maintain a persistent connection and can't always track ads using traditional methods.

Two common examples of desktop radio applications are iTunes and Spotify. iTunes is a player with no visual component, but Spotify does include a visual component capable of displaying visual image ads within the player.

Mobile Applications
Like desktop applications, mobile applications are downloaded onto the mobile device and operate within the device's operating system instead of within a browser. Both single feed and multiple feed radio stations often offer mobile applications.

Some mobile audio applications may include a visual component capable of displaying banner ads and may even have the ability to send tracking information when the device is connected to the Internet. Other applications may be more simplistic and offer only audio without any visual capabilities. In either case, mobile devices lack a persistent connection to the Internet. While most commercial audio content providers have circumvented this limitation, some providers may find that providing real-time tracking information is challenging.

In-Car Audio Players
In-Car Audio Players possibly represent the least functionality available for digital audio players. The car audio player is both the device and program in which audio signals or files are executed. We call this type of player a "thin" device. This type of thin device is hardcoded for operation, often lacking a "soft" operating system that can be updated. Without the ability to be updated to recognize something like an XML file for receiving ads, these thin devices cannot send tracking signals for tracking ad delivery and performance on their own.

However, an in-car audio player may function in one of two ways: either directly executing signals from the device itself or from an auxiliary device such as a mobile device that is connected to the in-car audio player. In the second case, the mobile device is tethered to the in-car player and is considered the player that executes audio signals. The in-car player simply functions as speakers for the mobile device.

Smart Radios, Connected TVs and Set-top Boxes
Modern entertainment devices such as radios, TVs, and auxiliary devices for TVs are being developed with the ability to connect to the Internet and take advantage of modern advances in technology. However, in many cases, updating device software is limited to the device's hardware design. For example, if buttons are built in to the device (play, rewind, pause, mute, etc.) updating software to include additional buttons simply can't be done.

Despite their limitations, these modern devices fall somewhere between full-featured devices and thin client devices. Many support browser functionality and include a visual component that could be used to display image-based ads.

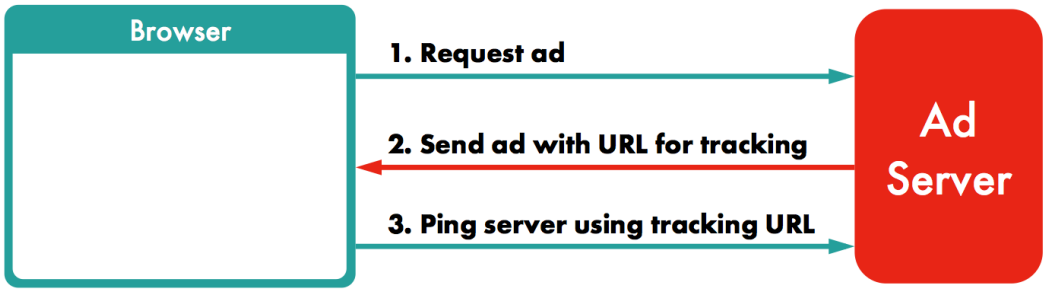
Game Console Audio Players
Digital game consoles are full of rich functionality that includes the ability to incorporate audio player applications that game-players can use to listen to their favorite music while playing video games. Such audio applications are capable of full browser functionality and should be able to receive digital audio ads as well as track audio ad performance. Visual components for game console audio player applications may also be available depending on the application design.

1.3 How Ad Serving Works in Display Advertising

Display advertising involves serving images to a webpage. Webpages are built using HTML, and serving an image to a page is done in much the same way that a Web developer would insert images and other media from his or her own files. It's all based on browser technology that uses HTML as a framework for requesting, sending, receiving, and acknowledging files from one server to another.

HTML enables dynamic insertion of ads into content based on what is known about the audience and context into which it is served such as content classification and geographical location. Details about a served ad, such as time, date, and context can be tracked using browser-based technology.

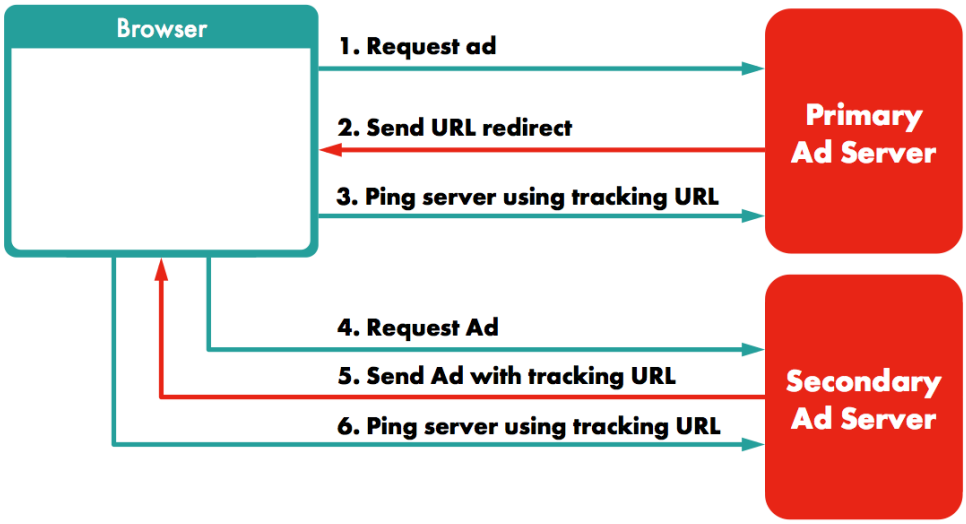
The following diagram models a simplified ad serving process using browser-based technology:



1. **Request ad:** The browser requests an ad using a line of HTML code that was placed on the page ahead of time. This line of HTML code is executed when a site visitor opens the page.
2. **Send ad with URL for tracking:** The ad server responds with html code that includes the ad file and a tracking URL.
3. **Ping server using tracking URL:** When a browser "pings" a server, it's making a request to a server using the URL the server sent. In the case of tracking, the URL identifies a server location where it can retrieve a file, typically an invisible 1x1 pixel-sized image that the browser loads on the page. When this file is requested, the server knows that the ad sent in the previous step was received and logs tracking information, such as time and date, about the ad served.

The above model represents a direct exchange between the browser and an ad server. The ad server may belong to the site owner who publishes content using the browser, or the ad server can belong to any number of technology platforms that either manage ad service for the publisher, for the advertiser, or for a variety of vendor services in between. In most cases, several ad-serving systems are involved in providing ads to the browser.

The following diagram models an ad serving process in which the primary ad server redirects the browser to a secondary ad server:



1. **Request ad:** The browser requests an ad using a line of HTML code that was placed on the page ahead of time. This line of HTML code is executed when a site visitor opens the page.
2. **Send URL redirect:** The ad server responds with html code that redirects the browser to a secondary ad server.
3. **Ping server using tracking URL:** The ad request is logged.
4. **Request ad:** The browser requests the ad from the secondary server using the HTML code received from the primary server.
5. **Send ad with tracking URL:** The secondary server responds with HTML code that includes the ad file and multiple tracking URLs—one for each server used in serving the ad. Sometimes additional URLs are included for pinging systems not involved in serving the ad but instead collect information on the ad served.
6. **Ping server using tracking URL:** The ad request is logged.

In both of the simplified ad serving examples above, HTML is used between systems to serve, render, and track ads. Browser technology provides a stable protocol used to communicate between systems. However, audio systems, like video, are built independently of HTML. In order for an audio system and a server to exchange information, they have to use a common language. DAAST uses XML to establish this common language. The following section explains how audio ad serving works using an XML template.

1.3.1 Audio Ad Serving Using an XML Template

Since audio players are not browsers, ad serving works a bit differently in digital audio than in display advertising. In digital audio, player code must be able to recognize a response to its request for an ad. Since players are all coded differently, an ad response must be coded specifically to match the player's code. The operational overhead for the parties involved to achieve this limits the ability for growth.

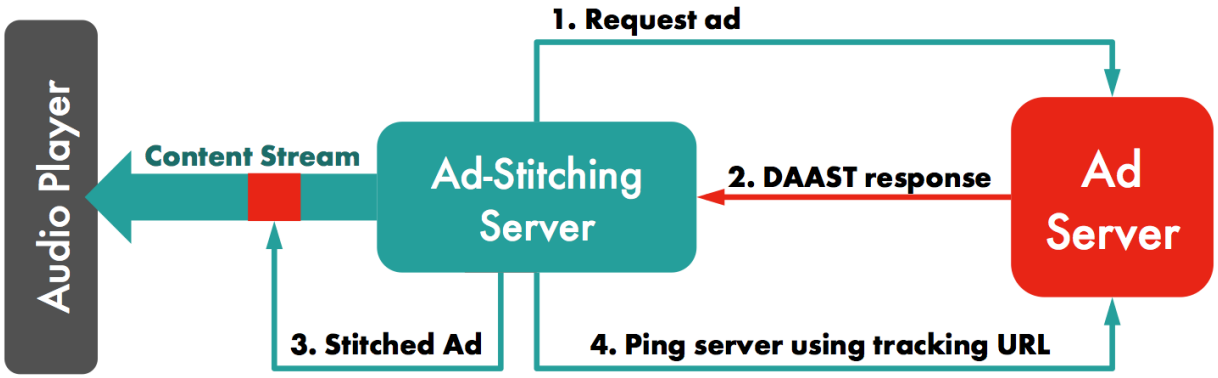
However, using a template based on standardized Extensible Markup Language (XML), audio player code can be adjusted to identify ad file information that a server sends using the template. XML is a way of packaging information that can be structured independently of the

rendering so that other systems can find the information if they know what they're looking for. Using an XML template, digital audio ad servers can send files to any audio player that is programmed to support the template and parse its contents for execution.

In general, the ad-serving process supported by an XML template involves the audio player requesting an ad, executing the ad response, and sending tracking information for ad impressions and other events back to the server. For audio players that use browser technology, this process looks a lot like the browser-based display advertising process. The only difference is that the systems involved are using XML instead of HTML. However, within the XML template, URLs are included to locate and track ad details.

But in many other digital audio scenarios, audio players may only use simple code for executing a stream of content without the browser-based ability to send and receive XML-nested HTML code for requesting and tracking ads. The audio player in this case is often called a "thin client." In general, thin clients are devices, such as a car radio, that can only execute a stream of content. In other cases, the thin client may be implemented within a device as an application, such as the native audio application in a smart phone. In these cases an intermediary server may be used to "inject" or "stitch" ads into the audio stream.

The following diagram illustrates how ad serving works when an ad-stitching server is used to serve and track ads:



1. **Request ad:** The ad-stitching server makes a call to the ad server.
2. **DAAST Response:** DAAST is the XML-formatted response the server uses to deliver the ad and all relevant information regarding how to execute it and track it.
3. **Stitch ad:** The ad-stitching server stitches the ad into a stream of content that it serves to the audio player. The audio player only recognizes a single stream of content, but the ad-stitching server can detect, to some extent, when the ad was played as well as a few other select events.
4. **Ping server using tracking URL:** The ad-stitching server pings the ad server to indicate that the ad was played.

In the scenario just described, two servers are involved: one is the ad server and one is the ad-stitching server, but the ad-stitching server is placing and tracking ads on the audio player's behalf. The audio player is the client, which is the system that plays the ad for the listener.

In interactive advertising, tracking directly from the client has been determined to be more accurate and produces fewer count discrepancies. The following sections outline the implications of using the ad-stitching server to track ads when the client cannot.

1.3.2 Digital Ad Tracking: Client-Side Vs. Server-side

Digital advertising began in a browser-based system and uses protocols designed to use the browser to execute and track ad service and performance. The process involves a client and a server.

The client is the party that hosts content and executes ads that it receives. The server is the party that sends the ad. The server can track when it sends the ad, and the client can track when the ad is received. However, the server has no way of knowing whether the ad was received unless the client can send confirmation to the server. Notifying the server that an ad was received is called client-side tracking.

Client-side tracking begins when a page begins to load within a browser. Within the page content is a line of code that tells the browser to fetch an ad from the server. This ad call to the server prompts the server to return a line of code that tells the browser where to fetch the ad creative using a URL. Included in that line of code is another URL to a 1x1 transparent pixel image or other file. When the browser makes the request for the secondary URL, the server is notified that the ad was received. Because of the use of the pixel, this process is often called "pixel tracking."

Client-side tracking is a better method of counting how many ads are served rather than relying only on server-side counts because both parties are involved in the process and counts on both sides can be more closely aligned. Client-side counts are also closer to the ad being displayed.

In server-side tracking, both parties track the ad independently and discrepancies between the two counts can vary a great deal.

1.3.3 Tracking With an Ad-Stitching Server

The ad-serving process described in section 1.3.1 uses an ad-stitching server on behalf of an audio player client that can't execute and track ads independent of the content stream they play. This method falls into a gray area. The ad-stitching server is not the client that plays the ad, but it tracks ads on behalf of the client. This type of tracking is done neither server-side nor client side.

However, because a lot of audio players are clients that can't track ads, ad-stitching server tracking is a common practice in digital audio advertising.

1.3.4 Moving Toward Client-Side Tracking in Digital Audio

Technology in audio has not yet caught up to the capabilities of digital display advertising and limitations exist for client-side ad tracking in audio ad serving. The biggest limitation in audio is the abundance of devices used to play audio, many of which are not browser-based and therefore cannot use browser-based technology to track ads. The benefit for implementing DAAST, however, is that audio players can track ads with or without browser support.

While there is work to be done, the IAB along with the Media Ratings Council (MRC) are moving the industry toward the more accurate client-side tracking methods where possible, but work is being done to consider the best way to count impressions for devices that cannot provide tracking details.

Audio players that can be updated as DAAST matures should build support for DAAST into their players as soon as possible. Audio players that lack browser support can use DAAST to support client-side tracking. Until DAAST is implemented for players and devices that lack client-side counting abilities, using an ad-stitching server to tracks ads on the audio player's behalf is acceptable because it's closer to client-side tracking than relying solely on server counts.

As digital audio's support for client-side tracking grows, the more marketers will find digital audio as a viable option to use in their campaigns.

1.3.5 Caching and Asset Transcoding with Ad-stitching

An ad-stitching server acts on behalf of many different types of clients (e.g. browsers, phones, TVs, stereos, etc.). In order to guarantee appropriate asset availability across all of the device types it supports, the ad-stitching server may assume responsibility for transcoding any assets it receives. In such cases, the ad-stitching server may be required to cache DAAST documents from 3rd-party servers in order to accommodate its transcoding needs.

Ad-stitching document caching changes the one-to-one correspondence between the client and the ad server. Impressions and tracking events are counted correctly and assets are up-to-date within reasonable time frames using an ad-stitching server, but in this model, the DAAST response document and its enclosed assets may be cached and used for many clients, and therefore, listeners.

Until the industry has standardized some best practices for the ad-stitching server business model, parties to a digital audio transaction should make note of this practice and discuss its implications in their audio campaigns.

1.4 Ad Types in Digital Audio

Streaming audio is an auditory experience for listeners. In most cases, advertising in digital audio means interrupting the stream of content with an audio ad. We call this a linear ad because the ad is played linearly within the same stream that plays the content. A linear audio ad is provided in audio format only, unlike a linear video ad that includes both audio and visual components.

Many audio devices don't support visual components because these devices are typically designed to be heard while the listener is engaged in other activities like driving, jogging, or riding a bike. However, some audio players, such as those built using browser technology or designed to play on the desktop or a mobile device, may support the ability to display a visual component to the ad.

DAAST supports linear audio-only ads with options to display a visual component when visual components are supported.

The following sections provide more detail on the type of digital audio ads that can be served using DAAST. Despite DAAST support for these ads, audio players can only process ad types

based on their technical capabilities. See section 1.2 for information on different audio player and devices. See section 2 for information on compliance recommendations.

1.4.1 Linear Ads

One of the key characteristics of a linear audio ad is that it demands the full audible attention from the listener because the ad takes over the audio experience entirely for a bounded duration of time. In general, linear audio ads may play for anywhere from between 15 seconds and a minute. Ad agencies should check with publishers about guidelines for audio creative, including ad duration.

Linear audio ads play along the timeline of the content media. Placement along the timeline may play a role in determining the value of the placement and classifies the ad as one of the following:

- **Pre-Roll Audio Ad:** delivered before the audio content stream begins to play. (Sometimes referred to as a Gateway Audio Ad)
- **Mid-Roll Audio Ad:** delivered between audio content segments.
- **Post-Roll Audio Ad**: delivered after or upon completion of the audio content. This type of unit is only available for audio content with a set length of time.

While most audio ads are a simple audio file, technology is challenging static audio content providers with innovations that enable the listener to interact with the ad using an audible response. DAAST supports the ability to track such interactions, but execution and tracking can only be done with audio players that can process interactions and supply tracking details.

1.4.2 Companion Ads

Companion ads are a concept inherited from the world of display. A companion ad is a visual ad that is served with the audio file and displayed in the webpage surrounding the player or in some other visual component the player supports.

The purpose of a companion ad is to invite interaction from the audience viewer, or to leave behind a reminder (leave-behind) after the audio ad has completed. Another use for companion ads is to display a logo tile for the ad in place of the artist tile that typically displays while the audio content plays. A companion ad may also serve as a skin that wraps the audio player.

One or more companion ads may be served with the originating linear portion of the ad, or what is called the Master Ad. The player may disregard companion ads when no visual component is available.

Although displayed only in audio players with a visual component to do so, companion ads in an audio context are displayed while the listener may have the content and ads running in the background and therefore will not even be seen when the ad is in-view. As of the release of this document, companion ad impressions are governed by the display measurement guidelines where viewable metrics are aggressively pushed as a measurement standard. While no plans exist to consider provisions for counting companion ads in an audio context, buyers and sellers of companions in audio should discuss the expectations for processing companion ads in audio inventory.

1.4.3 Ad Pods

Ad Pods support the delivery of a set of sequential linear ads. Ad Pods can play before, during a break in, or after the content audio plays and function like a commercial break in a radio program with multiple ad spots.

Placement of an ad pod is outside the scope of this specification but may be positioned by audio player programming. Additionally, while not specified for audio, the IAB Video Multi-Ad Playlist (VMAP) specification may have applications in audio for placing ad breaks in a stream of audio content. See http://www.iab.net/vmap to learn more about VMAP.

1.4.4 Skippable Ads

Recent technology has enabled skippable ads in audio. Implementations in digital video have shown that providing skip controls for the viewer helps advertisers learn more about the viewers' interests, and may reduce viewer drop off rates within the publisher's audience.

At the publication of these specifications a few audio publishers had begun to experiment with skippable ads in digital audio. While the technology is new, DAAST supports the ability to implement and track skip controls but audio publishers may choose whether they wish to implement and support skippable ads.

1.5 What This Template Specification is not

This document describes functional schema format requirements and expectations for compliance with this specification. The following details are out of scope for this document.

Creative Specifications
This document provides technical details on using the XML schema that transfers data from one system to another. Audio ad creative specs are not included as part of those technical details. Marketers should communicate with audio content publishers regarding creative specifications for advertising in their players.

Ad Metric Specification
While this template covers some details about sending URIs that track ad delivery, this document does not provide details on audio ad-serving metrics or the best practices for tracking audio ads. The IAB Digital Audio Committee is working on audio ad metric definitions and guidelines.

Player Execution of a Digital Audio Ad Serving Template response
XML does not "do" anything; it's simply a mechanism for transferring data from one system to another. Audio player technology must be able to parse a DAAST ad server response and use the data in accordance with the guidelines in this document. This document provides detailed requirements for the ads in a DAAST response, but does not provide a concrete technical implementation. Audio player engineers can use the information in this document to design and build a compliant audio player, using whatever technology the engineer prefers to use.

Implementation
This specification provides details about key elements used for serving and tracking audio ads and how to support them, but implementation details are left to the businesses that choose to support them. To the extent possible, minimum compliance recommendations should be implemented, but where technology is lacking, the audio player should disclose which DAAST elements are supported and to what extent.

Mechanism for Disclosure
DAAST 1.0 offers some options for audio player implementations on what they must support in order to be DAAST-compliant. The party who owns the DAAST-compliant audio player must disclose what features are supported, but the timing and process for doing so is currently left to the parties involved. Some audio ad agencies and partners need disclosure in real-time, but some parties may find that disclosure prior to campaign implementation is sufficient. The IAB is moving toward putting guidelines and certification practices in place to improve efficiency in IAB-compliant systems.

2 Compliance Recommendations

The IAB Digital Audio Ad Serving Template (DAAST) is a template for providing digital audio ad information to audio players using Extensible Markup Language (XML). This specification outlines the structure for providing ad details such as where to find ad creative files, the advertiser represented in the ad, tracking information, and other information necessary for the audio player to execute as well as for providing execution details valuable to marketers. As noted in section 1.5, audio ad creative specifications are not included as part of these technical details. To be interoperable within systems of the digital audio ecosystem, both ad servers and audio players should adopt these guidelines.

This section provides general details about how audio players and ad servers should use and respond to DAAST ads so that compliant systems contribute to increased demand and expanded options for monetizing audio content. As discussed further in sections 2.3 and 2.4 of this document, where technology is lacking, the audio player should disclose which DAAST elements are supported and to what extent.

2.1 Audio Ad Server Expectations

In order for an audio ad to function in a DAAST-compliant audio player or audio streaming server the ad server must respond to the player's request for an ad with an XML response formatted according to the specifications outlined in this document. Ideally, audio ad servers should also be able to receive, process, and respond to any player requests for tracking and error details in the DAAST response that the ad server sent.

As a best practice, ad vendors should consider volume normalization. A sudden spike in volume can sometimes damage a listener's device. Some publishers and their ad-stitching server vendors won't allow an ad to be played until they've ensured that the ad's volume has been normalized.

2.2 Audio Player Expectations

DAAST enables audio publishers to leverage their audio players' unique designs, but adjustments to the code are needed in order to read and process a DAAST ad response. In general, the audio player should be able to recognize a DAAST response, parse it, and execute the provided files depending on what the player can support.

Audio players should accept ads that they can support and that comply with the publisher's specifications. Ads that fall short of publisher requirements may be ignored. For example, if a publisher requires ads that are 28 seconds in duration, then any ads returned that don't meet that requirement may be ignored.

A DAAST-compliant audio player is responsible for:

- Playing the audio ad as specified and within the limits of audio player declared supports
- Respecting ad server instructions in a DAAST response, including those of any subsequent ad servers called in a chain of wrapper responses, providing the responses comply with the specifications outlined in this document and within limits of audio player declared supports
- Supporting XML conventions such as standard comment syntax (e.g. acknowledge comments in the standard XML syntax: `<!--comment-->`)

Details for proper audio ad play and DAAST support are defined throughout this document.

2.2.1 Requesting DAAST Ad Types

DAAST offers four different types for compliance. (See section 2.3 for details.) Publishers must declare which type(s) they support for compliance. Audio players should also indicate supported type(s) during the ad request.

Players may communicate directly with the audio ad server or with the use of an ad-stitching server that could be either the publisher's or a vendor's. When using an ad-stitching server, publishers need to pass as much information as possible about the listener and the player capabilities for DAAST requests, which may be relevant in the ad selection process. Such data may include:

- User's IP address
- Client's user-agent
- Ad types supported by the player
- If supporting companion ads
    - Details on the rendering capabilities, such as Flash or JavaScript
    - Ability to handle click-through links

The mechanism to do this is outside the scope of DAAST but should be considered.

2.2.2 Imposing Template Structure

Publishers are encouraged to set requirements (i.e. file size, audio format, duration, etc.) for what they will accept and play in their audio players. Advertisers should always consider publisher requirements when developing creative for an audio ad campaign.

However, when publishers also require that DAAST ad responses include ONLY details for whatever meets their requirements, then the spirit of the cross-platform audio ad delivery that DAAST offers is lost.

For example, DAAST offers a means for ad servers to include multiple media files: perhaps one for each audio type that meets requirements for a variety of audio players. Each audio player can then parse the DAAST response for the media file that meets individual publisher requirements. Such a response can be served across multiple DAAST-compliant audio players without modification.

If, however, a publisher stipulates that a DAAST ad contain ONLY the elements that are relevant to its requirements (e.g. only one media file that meets its requirements), then the DAAST response can only be served to that publisher (and any others that happen to have the exact same requirements).

Limiting an ad server's ability to create dynamic DAAST responses with details that meet a wide range of requirements greatly limits the benefits that DAAST was designed to offer. While DAAST guidelines don't restrict publishers from imposing specific structure, this practice is not recommended. Instead, publishers should accept any DAAST response that conforms to the specification as defined in the DAAST XSD. Publishers may ignore any details not supported and may decide not to play those ads that do not meet the publishers' requirements.

2.3 Ad Type Compliance

DAAST supports four ad types: linear ads, companion ads, ad pods, and skippable ads. These ad types and specific details for including them in a DAAST ad response are covered in section 3. At a minimum, compliant audio publishers should support linear audio ads. In addition, publishers may choose to support variations on linear audio ads with provisions for companion display ads and/or audio ad pods. However, supporting additional ad types is not required for IAB DAAST compliance.

The DAAST compliance ad formats are as follows:

1. Linear Audio Ads
2. Companion Display Ads
3. Audio Ad Pods
4. Skippable Ads

Companies wishing to display IAB's seal for DAAST compliance must declare which of the above ad types they support.

2.4 Minimal Compliance Recommendations

In addition to declaring which audio ad types are supported in compliance with DAAST, audio players should support some key elements used for serving and tracking audio ads. While this section provides recommendations for minimal compliance, any DAAST-compliant audio player with the ability to support recommended features should make every attempt to do so. Players that lack the ability to support any of the features listed should be transparent about what is supported and what is not. Additionally, players lacking such technology should work toward implementing technology or services that meet DAAST minimal compliance recommendations.

DAAST-compliant audio players should support the following DAAST features:

1. Inline and wrapper ads (to support multiple ad-serving vendors)
2. Tracking events
3. Error reporting
4. Industry icons (only when visual components are supported)*

* With privacy concerns among consumers and special interest groups, any attempts to support industry icons, such as the AdChoices icon used to offer consumers information and choice on the ads they see, further serves to improve self-regulation in the industry. The Digital Advertising Alliance (DAA) leads the Internet-Based Advertising (IBA) Self-Regulatory Program responsible for the AdChoices icon and other initiatives designed to offer consumers some control over their ad experience and any data collected on them. To learn more about the DAA's IBA program, visit http://www.iab.net/public_policy/behavioral-advertisingprinciples.

This specification provides details about the features listed above and how to support them, but implementation details are left to the businesses that choose to support them. For example, supporting tracking events involves client-side tracking, meaning that the audio player (client) makes a call to the ad server to indicate that an event was executed on the client-side. Some audio devices may not have the technology available to support client-side tracking. In such cases, services such as ad-stitching servers should be considered to help bring ad tracking closer to a client-side tracking solution. See section 1.3.3 for more information on tracking ads with an ad-stitching server.

Digital advertising in general has promoted and supported client-side tracking to the extent possible as technology allows. Client-side tracking offers the most accurate measurement for both the ad server and the audio player by helping to reduce discrepancies between what is tracked on both sides of the transaction. However, digital audio measurement and tracking may struggle to achieve client-side tracking because of the proliferation of devices used that lack the technology to do so. However, as the technology matures, DAAST can help remove much of the friction present in today's market just as VAST did the same for digital video. See section 1.3.2 for more information on client-side tracking.

As discussed in section 1.5, this document does not provide details on audio ad-serving metrics or the best practices for tracking audio ads. The IAB Digital Audio Committee is working on audio ad metric definitions and guidelines.

These compliance recommendations are flexible for audio players that may lack the technology necessary to comply with certain features in DAAST. In these cases, the audio publisher should disclose to its customers and partners what DAAST features are supported and to what extent.

2.5 Browser Security for Flash™ and JavaScript™

Modern browsers restrict Adobe Flash and JavaScript runtime environments from retrieving data from other servers. For audio players that receive DAAST responses using a browser, measures must be taken to accept any Flash or JavaScript ad files.

2.5.1 crossdomain.xml for Flash

To enable Flash audio players to accept a DAAST response, ad servers must provide a crossdomain.xml file at their root HTTP domain. For example, adserver.com should provide the file as follows:

`http://adserver.com/crossdomain.xml`
Flash audio players know to check the root domain of any server that sends it data.

The xml is a simple file that includes the following:

```
<cross-domain-policy>
      <allow-access-from domain=”*”>
<cross-domain-policy>
```

The asterisks (*) value for the domain attribute enables any Flash audio player to accept data from the server providing the crossdomain.xml file.

For more information, visit http://kb2.adobe.com/cps/142/tn_14213.html

2.5.2 Cross Origin Resource Sharing (CORS) for JavaScript

In order for JavaScript audio players to accept a DAAST response, ad servers must include a CORS header in the HTTP file that wraps the DAAST response. The CORS header must be formatted as follows:

```
Access-Control-Allow-Origin: <origin header value>
Access-Control-Allow-Credentials: true
```

These HTTP headers allow an ad player on any origin to read the DAAST response from the ad server origin. The value of `Access-Control-Allow-Origin` should be the value of the `Origin` header sent with the ad request.

Setting the `Access-Control-Allow-Credentials` header to `true` will ensure that cookies will be sent and received properly.

For more information, visit http://www.w3.org/TR/cors

2.6 XML Namespace

Whenever DAAST is used in conjunction with any other XML template, a namespace should be declared for each so that the elements of one are not confused with the elements of another.

For more information, visit: http://www.w3.org/TR/REC-xml-names/

3 DAAST Implementation Details

This section provides detailed requirements for ad servers and audio players that wish to consider themselves compliant with DAAST guidelines, thus ensuring that any two DAAST-compliant systems behave as expected and therefore interoperable. Both the general DAAST concepts and the requirements for different ad formats are provided.

3.1 General DAAST Document Structure

A DAAST-compliant ad response is a well-formed XML document, compliant with XML 1.0 so that standard XML requirements such as character entities and `<!--XML comments-->` should be honored. It must also pass a schema check against the DAAST 1.0 XML Schema Definition (XSD) that is distributed in conjunction with this document. Lastly, it must conform to certain additional dependencies that cannot be expressed in the DAAST 1.0 XSD. These dependencies are outlined in this section and further described throughout this document.

3.1.1 Declaring the DAAST response

All DAAST responses share the same general structure. Each DAAST response is declared with `<DAAST>` as its topmost element along with the `version` attributes indicating the official version with which the response is compliant. For example, a DAAST 1.0 response is declared as follows:

```
<DAAST version="1.0">
```

As with all XML documents, each element must be closed after details nested within the element are provided. The following example is a DAAST response with one nested `<Ad>` element.

```
<DAAST version="1.0">
      <Ad>
            <!--ad details go here-->
      </Ad>
</DAAST>
```

3.1.2 The Root <Ad> Element

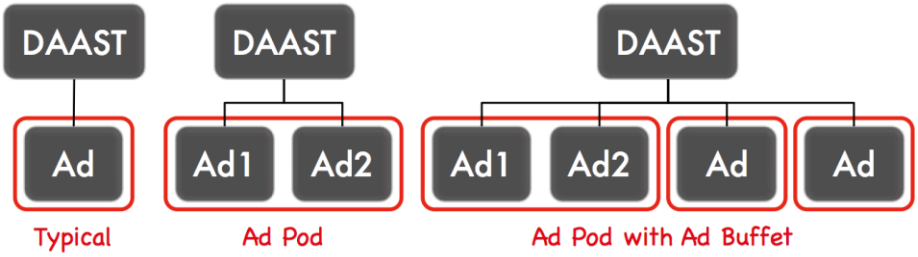
Within the `<DAAST>` element are one or more `<Ad>` elements. Advertisers and audio content publishers may associate an `<Ad>` element with a line item audio ad defined in contract documentation, usually an insertion order. These line item ads typically specify the creative to play, price, delivery schedule, targeting, and so on. In DAAST, an `<Ad>` element contains all of the information necessary for the audio player to play and track the Ad creative, and often maps back to the line item for the audio ad in a contract between parties.

While the association of the DAAST `<Ad>` element with a line item in a contract is common, it's not always the case. Regardless of how contracts define an ad, DAAST defines ads in a very structural way as described in the following sections.

A DAAST response may offer multiple ads. A single `<Ad>` element is most common, and represents the case where only a single ad is to be played by the audio player. Other responses may include an ad pod or "ad buffet." The difference between an ad pod and ad buffet is that

ads in an ad pod are sequenced and played in a specified order. An ad buffet is a collection of ads that can be played in any order. Which ads or how many ads are played in an ad buffet is optional, but in an ad pod, all ads must be played and in the order specified.

The following diagram illustrates how the <Ad> element may be represented in a DAAST response.



When multiple ads, whether part of a pod or a collection of stand-alone ads, are included in a DAAST response, the audio player is only required to support multiple ad playbacks if it has declared that it supports multiple ads or ad pods. If the audio player cannot execute an ad response with multiple ads, it can decline from loading the ad resources and send an error code. See section 3.2.3 for details on Ad Pods.

Audio Player Implementation Note	If multiple <Ad> elements are provided with sequence attributes, they must be played as a Pod when the Ad Pod format is supported. If not supported or the Pod cannot be played, the audio player should report an error using the error-tracking URI provided.

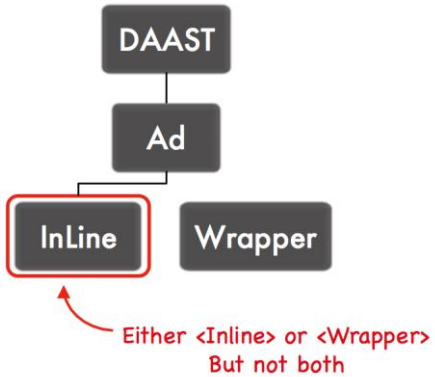
3.1.2.1 Ad Attributes

Two attributes are available for the <Ad> element:

- **id:** an ad server-defined identifier string for the ad
- **sequence:** a number greater than zero (0) that identifies the sequence in which an ad should play; all <Ad> elements with sequence values are part of a pod and are intended to be played in sequence

3.1.2.2 Ad Structure

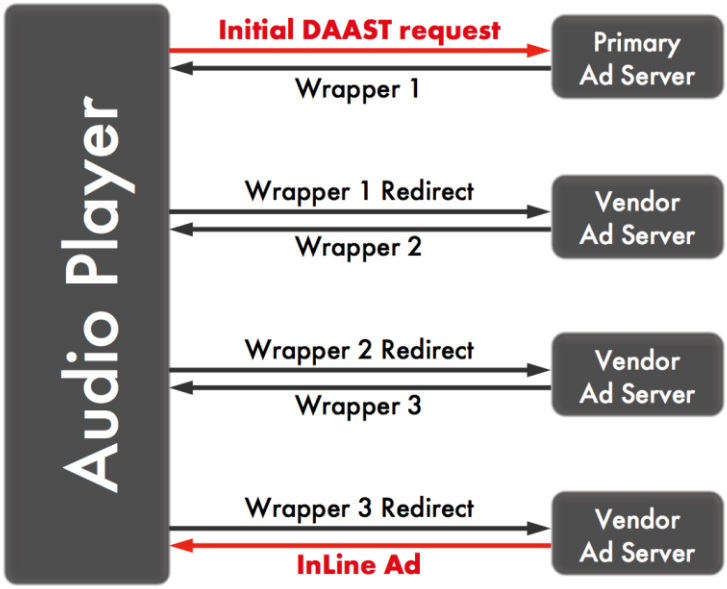
Each <Ad> contains a single <InLine> element or <Wrapper> element (but never both) as illustrated in the following diagram.



3.1.3 The <Wrapper> Element

The `<Wrapper>` element contains a URI reference to another ad server for an ad. When the first ad server, or primary ad server, responds to a DAAST request, it may not provide an inline ad. Instead, it will respond with a wrapper that points to a secondary ad server, or vendor ad server (often called a third party ad server). The secondary ad server may provide the `<InLine>` ad or may provide a secondary `<Wrapper>` response that points to yet another ad server. Eventually, the last ad server in the ad supply chain must respond with an `<InLine>` ad that provides all required elements to serve the ad.

The following diagram illustrates how a chain of wrappers leads to an inline ad:



See section 3.3.1 for more details on Wrapper ads.

3.1.4 The <InLine> Element

The last ad server in the ad supply chain serves an `<InLine>` element. Within the nested elements of an `<InLine>` element are all the files and URIs necessary to display the ad.

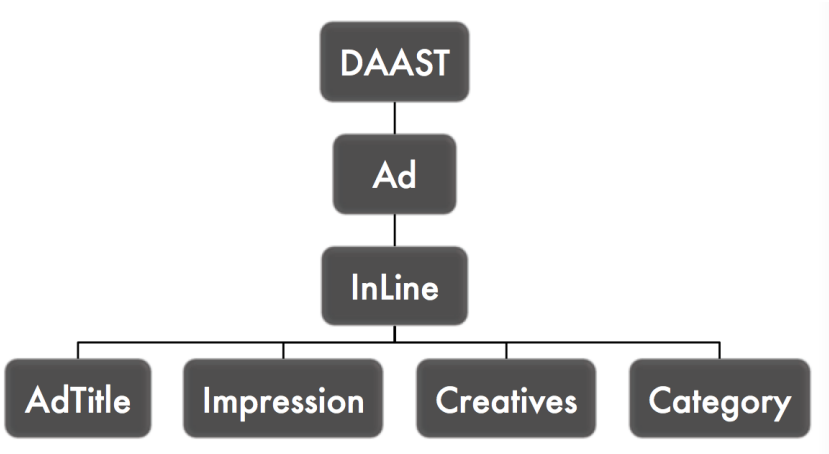
3.1.4.1 Required <InLine> Elements

Contained directly within the `<InLine>` element are the following required elements:

- **<AdTitle>**: the common name of the ad
- **<Impression>**: a URI that directs the audio player to a tracking resource file that the audio player should request when the ad begins to play
- **<Creatives>**: the container for one or more `<Creative>` elements
- **<Category>**: for use to identify the ad category code as defined in the DAAST Ad Categories (see, http://www.iab.net/DAAST)

DAAST Ad Categories Implementation Note	The DAAST Ad Categories are based off of the IAB Quality Assurance Guidelines (QAG) Contextual Taxonomy and their codes from IAB OpenRTB specification. DAAST ad content should be coded at the first and second tiers of the QAG contextual taxonomy. • Quality Assurance Guidelines Contextual Taxonomy • OpenRTB 2.0 Specification, section 6.1 Content Categories

Thus far, the DAAST response structure can be represented as follows:



3.1.4.2 Optional <InLine> Elements

The following may also be contained within the `<InLine>` element, but these elements are optional:

- **<AdSystem>**: the name of the ad server that returned the ad
- **<Description>**: a string value that provides a longer description of the ad.
- **<Advertiser>**: the name of the advertiser as defined by the ad serving party. This element can be used to prevent playing ads with advertiser competitors. Ad serving parties and publishers should identify how to interpret values provided within this element. As with any optional elements, the audio player is not required to support it.
- **<Expires>**: the number of seconds in which the ad is valid for execution. In cases where the ad is requested ahead of time, this timing indicates how many seconds after the request that the ad expires and cannot be played. This element is useful for preventing an ad from playing after a timeout has occurred.
- **<Survey>**: a URI to a survey vendor that could be the survey, a tracking pixel, or anything to do with the survey. Multiple survey elements can be provided. A type attribute is available to specify the MIME type being served. For example, the attribute might be set to `type="text/javascript"`. Surveys can be dynamically inserted into the DAAST response as long as cross-domain issues are avoided.
- **<Error>**: a URI representing an error-tracking pixel; this element can occur multiple times. Errors are defined in section 3.3.2.3.
- **<Pricing>**: provides a value that represents a price that can be used by real-time bidding (RTB) systems. DAAST is not designed to handle RTB since other methods exist, but this element is offered for custom solutions if needed. If used, the following two attributes must be identified:
  - **model:** identifies the pricing model as one of `"CPM"`, `"CPC"`, `"CPE"`, `"CPV"`, or `"CPO"`.

- **currency:** the 3-letter ISO-4217 currency symbol that identifies the currency of the value provided (i.e. USD, GBP, etc.…)

If the value provided is to be obfuscated/encoded, publishers and advertisers must negotiate the appropriate mechanism to do so. When included as part of a DAAST Wrapper in a chain of Wrappers, only the value offered in the first Wrapper need be considered.

- **<Extensions>**: XML node for custom extensions, as defined by the ad server. When used, a custom element should be nested under `<Extensions>` to help separate custom XML elements from DAAST elements. The following example includes a custom xml element within the Extensions element.
  `<Extensions> <CustomXML>…</CustomXML></Extensions>`

3.1.5 DAAST Tracking

Tracking an ad served in DAAST format is done using a collection of DAAST tracking elements at different levels in the DAAST response. These tracking elements each contain a URI to a resource file or location on the ad server that sent the DAAST response. The resource file is usually (but not always) a 1x1 transparent pixel image (i.e. tracking pixel) that when called, records an event that is specific to that tracking pixel. The tracking pixel need not be rendered; just requested.

Audio Player Implementation Note	The audio player is responsible for requesting tracking pixels at appropriate times during the execution of a DAAST ad response. In the case where the audio player is not aware of the ads it plays and instead uses an ad-stitching server to receive ads, then the ad-stitching server is responsible for requesting tracking pixels at the appropriate times.

Most tracking elements are optional for the ad server to include, but if included, the audio player or a party acting on the audio player's behalf (such as an ad-stitching server) is required to request the resource file from the URI provided at the appropriate times. Advertisers and publishers depend on accurate tracking records for billing, campaign effectiveness, market analysis, and other important business intelligence and accounting. Good tracking practices throughout the industry are important to the success and growth of digital audio advertising.

General Implementation Note	The audio player sends requests to the URIs provided in tracking elements; however, the audio player is not required to do anything with the response that is returned. The response is only to acknowledge an event and to comply with the HTTP protocol. This response is typically a 200 with a 1x1 pixel image in the response body (although the response could be of any other type).

3.1.5.1 Summary of DAAST Tracking Elements

The following list of DAAST tracking elements summarizes tracking options offered at each level in DAAST.

<DAAST> Tracking Elements

- **<Error>** used when a "no ad" response is received

<InLine> and <Wrapper> Tracking Elements

- **<Error>** used to track when the InLine ad could not be served and why

- **<Impression>** used to track when the ad "impression" metric should be counted. Please note that as of the release of this document an industry standard definition for an audio impression had not yet been defined. DAAST implementers should be transparent about how they define audio impressions until an industry standard definition is formally defined.

General Implementation Note	As of the release of this document, a standard for measuring audio impression had not yet been developed. DAAST implementers should be transparent about how they define audio impressions until an industry standard definition is formalized.

<Linear> Tracking Elements

- **<TrackingEvents>** used to contain tracking events using the `<Tracking>` element for IAB-defined tracking events. While tracking events have not been defined for streaming audio, any of the events defined for digital video can be used here. See section 3.2.1.7 for details on available tracking events.
- **<AudioInteractions>** is used to track listener engagement with the ad. Two types of engagement are tracked with a `<ClickTracking>` element to track clicks in wrappers:
  - **<ClickThrough>** tracks an event that takes the listener away from the ad-sponsored content.
  - **<CustomClick>** tracks any type of interaction with the ad that is not included in the `TrackingEvents/Tracking` element.
  - **<ClickTracking>** used in a Wrapper response to track the ClickThrough for the resulting InLine ad or used in the InLine ad to track the ClickThrough when the `<ClickThrough>` URI lacks a tracking component or because the creative handles the click.
- **<IconClicks>** a container in the `Icons/Icon` element for elements of the following types:
  - **<IconClickThrough>** redirects the listener to the supplied URI and may track the click.
  - **<IconClickTracking>** used to track the icon click when the click cannot be tracked using `<IconClickThrough>`. Also used in a wrapper to track the clickthrough for creative that is not included in the wrapper.
- **<IconViewTracking>** tracks that the icon was displayed.

<Companion> Tracking Elements (See Section 3.2.2.7 for more information)

- **<CompanionClickThrough>** redirects the listener to the supplied URI and may track the click.
- **<CompanionClickTracking>** used to track the clickthrough click when it cannot be tracked using `<CompanionClickThrough>`. Also used in a wrapper to track the clickthrough for creative that is not included in the wrapper.

All tracking elements are available in both the InLine and Wrapper formats EXCEPT for the `<Error>` element at the `<DAAST>` level since it is only used when an InLine response is not returned.

3.1.5.2 Companion Click Tracking

Companion creative that are of a `<StaticResource>` type, such as an image, need a way to provide a clickthrough URI that directs users to the advertiser's webpage when they click the ad. The DAAST element `<CompanionClickThrough>` enables advertisers to include a clickthrough

URI for static image creative. In most cases, this clickthrough also provides tracking information that notifies appropriate parties that the ad was clicked.

However, when the resource type is one of `IframeResource` or `HTMLResource`, the creative can provide its own clickthrough. In this case the `<CompanionClickTracking>` element may be used to track the click.

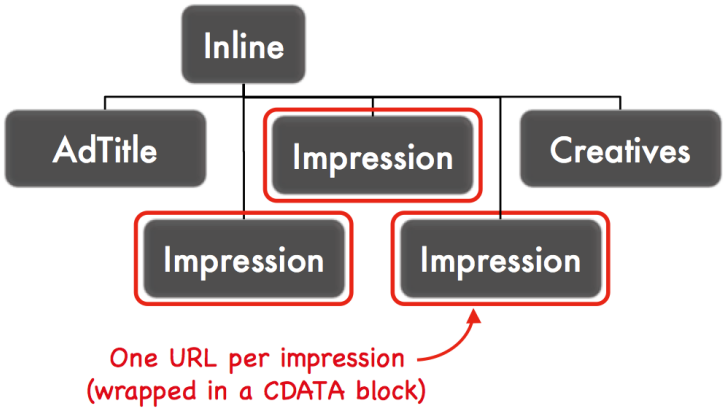
The table below describes which element to use for select static resource creative in DAAST InLine and Wrapper responses.

StaticResource Creative Type	InLine Creative	Wrapper Creative
Image	`<CompanionClickThrough>`	`<CompanionClickTracking>*`
Flash with no apiFramework	`<CompanionClickThrough>`	`<CompanionClickTracking>*`
Flash with apiFramework = clickTAG	`<CompanionClickThrough>`	`<CompanionClickTracking>*`
Any static resource where the ad unit handles the clickthrough	`<CompanionClickTracking>`	`<CompanionClickTracking>`

*When tracking clicks for Companion creative in a Wrapper that also include the creative resource files, then Companion creative should be treated as InLine creative and the <CcompanionClickThrough> element should be used.

3.1.5.3 The <Impression> Element

When an impression occurs in a DAAST audio ad, multiple parties can be notified using multiple `<Impression>` elements. The `<InLine>` element in a DAAST response may contain one or more `<Impression>` elements. Each `<Impression>` element contains exactly one child CDATA-wrapped URI. The included URI is used to notify a single party that an impression has occurred. Additional `<Impression>` elements are used to notify each additional party.



DAAST URIs and any other free text fields that might contain potentially dangerous characters should be wrapped in a CDATA block as demonstrated in the following example:

```
<Impression id=”myserver”>
```

```
        <![CDATA[
        http://ad.server.com/impression/dot.gif
        ]]>
</Impression>
```

Ad Server Implementation Note	All URIs or any other free text fields containing potentially dangerous characters contained in the DAAST document should be wrapped in CDATA blocks.

3.1.5.4 Impression vs. "Start" Event

Impression tracking URIs should be used to track the ad play. However, an ad may be made up of multiple creative. If the advertiser wants to track when individual ad creative are started in addition to tracking the audio ad impression, the DAAST response should include a "`start`" event under the `<TrackingEvents>` element for the creative to be tracked. See the tracking notes in sections were relevant.

3.1.5.5 Multiple Impressions

The use of multiple impression URIs allows the ad server to share impression-tracking information with other ad serving systems, such as a vendor ad server employed by the advertiser. When multiple impression elements are included in a DAAST response, the audio player should request all impressions at the same time. Any significant delay between impression requests may result in count discrepancies between ad serving systems.

Video Player Implementation Note	If multiple `<Impression>` elements are provided, they should be requested at the same moment in time or as close in time as possible. If any of the requests are delayed significantly, discrepancies may result in the participating ad serving system counts.

3.1.5.6 Tracking Records for Multiple Parties

Multiple parties involved in a digital advertising campaign may all want their own tracking records for an audio ad served in a DAAST format. There are different ways to do this, but DAAST enables the use of multiple tracking elements—each of which can provide a URI to the server of any party requesting notification of tracking information on the ad.

Tracking elements for each of the two creative elements (Linear and Companion) all include an `id` attribute. As with multiple Impressions described in the previous section, whenever multiple tracking elements of the same id are provided, the tracking resource for each should all be requested at the same time. Any significant delay in tracking resource requests can result in discrepancies in the participating parties' systems.

3.1.6 The <Creatives> Element

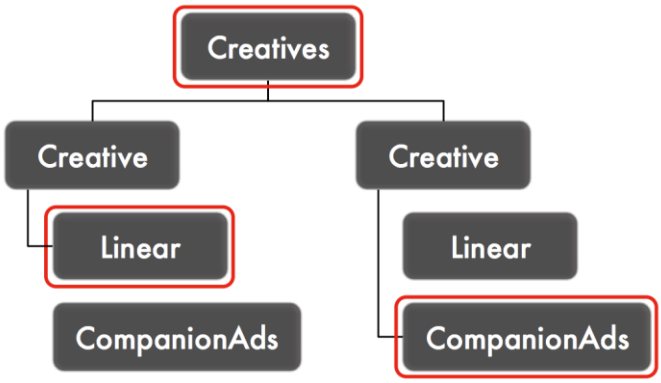
An ad in DAAST may be made up of several creative files. Despite how many or what type of creative are included as part of the Ad, all creative files should generally represent the same creative concept.

Within the `<InLine>` element is one `<Creatives>` element. The `<Creatives>` element provides details about the files for each creative to be included as part of the ad experience. Multiple `<Creative>` (singular) may be nested within the `<Creatives>` (plural) element. Note the plural

spelling of the primary element <Creatives> and the singular spelling of the nested element <Creative>.

Each nested <Creative> element contains one of: <Linear> or <CompanionAds>.

The following diagram represents a <Creatives> element that contains a Linear Ad with complimentary Companion ads.



The <Creative> element may contain a sequence attribute that identifies the numerical order in which each creative should play. Different than an ad pod, the creative sequence attribute identifies the order in which creative for a single ad should play. A multi-part ad in which two creative are played in sequence represent the single ad. Sequential display of creative in the absence of sequence values is at the video player's discretion.

General Implementation Note	The <Creative> sequence attribute should not be confused with the <Ad> sequence attribute. Creative sequence identifies the sequence of multiple creative within a single Ad and does NOT define an ad pod. Conversely, the <Ad> sequence identifies the sequence of multiple Ads and defines an Ad Pod. See section 3.2.3 for details about Ad Pods.

3.1.6.1 Creative Attributes

The following attributes are available for the <Creative> element:

- **id:** an ad server-defined identifier for the creative
- **sequence:** the numerical order in which each sequenced creative should display (not to be confused with the <Ad> sequence attribute used to define Ad Pods)
- **adId:** identifies the ad with which the creative is served
- **apiFramework:** the technology used for any included API

All creative attributes are optional.

3.1.6.2 DAAST Example: Linear with Companions

The following example demonstrates the basic structure of a DAAST response in XML format. This response represents a Linear Ad with Companions. Ellipsis (…) represent missing information and are used in examples throughout this document in order to highlight only the DAAST sections being discussed.

```
<DAAST version=”1.0”>
      <Ad>
            <InLine>
                  <AdSystem>My Ad Server</AdSystem>
                  <AdTitle>Car Company</AdTitle>
                  <Category>IAB2-20</Category>
                  <Impression>...</Impression>
                  <Creatives>
                        <Creative>
                              <Linear>...</Linear>
                        </Creative>
                        <Creative>
                              <CcompanionAds>...</CcompanionAds>
                        </Creative>
                  </Creatives>
            </InLine>
      </Ad>
</DAAST>
```

3.1.6.3 Creative Extensions

When an API framework is needed to execute creative, a `<CreativeExtensions>` element can be added under the `<Creative>`. This extension can be used to load an executable creative with or without using a media file.

A `<CreativeExtension>` (singular) element is nested under the `<CreativeExtensions>` (plural) element so that any xml extensions are separated from DAAST xml. Additionally, any xml used in this extension should identify an xml name space (xmlns) to avoid confusing any of the xml element names with those of DAAST.

The nested `<CreativeExtension>` includes an attribute for `type`, which specifies the MIME type needed to execute the extension.

The creative attribute, `apiFramework`, identifies the API needed to execute the creative. If the `apiFramework` attribute is not specified, the audio player may disregard creative. If the Ad cannot be fully executed without an API (identified or not) that the player does not support, then the audio player may disregard the Ad and use the `<Error>` element (under the `<Ad>` element) to notify the ad server that the Ad could not be displayed.

3.2 DAAST Requirements by Compliance Format

In DAAST 1.0, an audio player may choose to support different audio ad formats while maintaining DAAST-compliant status. Audio players may opt to support one or more of three DAAST ad formats:

- Linear Ads
- Companion Ads*
- Ad Pods*
- Skippable Ads*

*Support for Companion Ads, Ad Pods, and Skippable Ads is optional.

Providing optional compliance formats enable audio players to be compliant with DAAST guidelines while only supporting the formats that best suit their audio publishing model. Allowing the option of complying with select ad formats helps to increase adoption across the industry, making it easier for audio ad servers to increase reach across more publisher platforms.

Identifying Ad Formats in a DAAST Response

The following table summarizes the properties for elements found in a DAAST response that represents one of the three DAAST ad formats. A check ✓ under one of the ad format columns indicates that the DAAST element in that row should be found in the DAAST response for the listed format.

Engineers can use the following table to program audio players that quickly identify the DAAST Ad formats included in the DAAST response.

DAAST Ad Properties	Linear Ads	Companion Ads	Ad Pods
`<Ad>` (no sequence)	✓		
`<Ad sequence="n">`			✓
`<Linear>`	✓		
`<CompanionAds>`		✓ *	

*Companion creative must be served with a Linear creative and cannot be served alone.

Signature element properties for other formats may be found but can be ignored if they represent an ad format not supported.

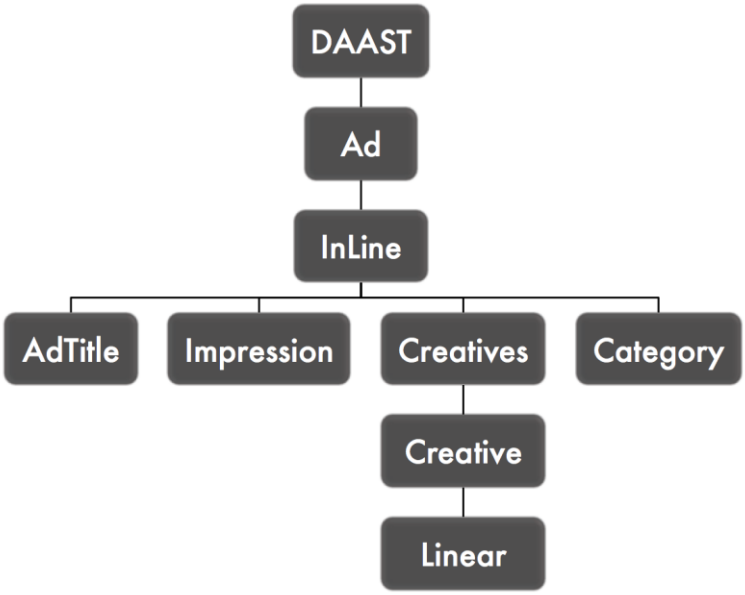
For example, if the audio player supports only Linear Ads but the DAAST response contains `<Ad>` elements with sequence attributes intended to play as an Ad Pod, as long as there's at least one `<Ad>` element with no sequence attribute, the audio player can ignore the additional sequenced `<Ad>` elements. But if the only options for ad formats found in the DAAST response are those of a format the audio player doesn't support, then the audio player can reject the ad and notify the ad server using the `<Error>` element for the `<Ad>`.

Likewise, if an audio player supports multiple formats such as both Linear Ads and Companion Ads, the signature element properties for both formats should be respected.

3.2.1 Linear Ad Format

The most common type of audio advertisement trafficked in the industry is a “linear ad”, which is an ad that plays instead of programmed audio content (e.g. during a commercial break or in between songs). Linear ads can be played before audio content plays (typically called pre-rolls), in between programmed audio content segments (typically called mid-rolls or in-stream ads), or after programmed audio content (typically called post-rolls).

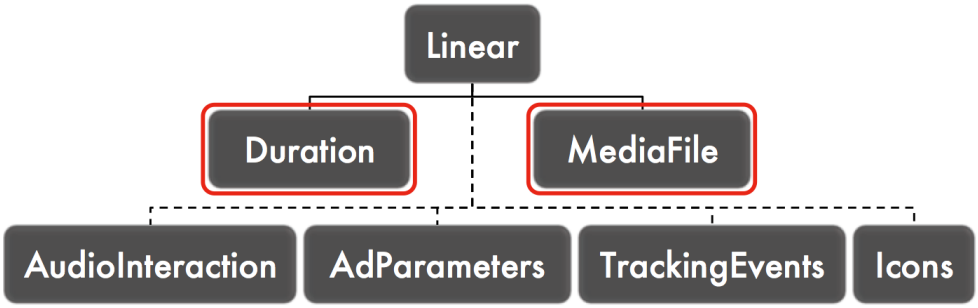
The DAAST response structure that represents a Linear Ad is represented in the diagram below.



3.2.1.1 Linear Elements

A `<Linear>` element has two required child elements, the `<Duration>` and the `<MediaFiles>` element.

The following diagram represents the elements that fall directly under the `<Linear>` element. Elements outlined in red are required.



3.2.1.2 The <Duration> Element

The ad duration of a Linear creative is expressed in the `<Duration>` element. Duration is expressed in the "`HH:MM:SS.mmm`" format (`.mmm` represents milliseconds and is optional). For example, a 30 second audio file is represented as follows:

```
<Duration>00:00:30</Duration>
```

Or alternately:

```
<Duration>00:00:30.000</Duration>
```

The `.mmm` extension for milliseconds should be used whenever possible to avoid stopping the creative prematurely.

A `<MediaFiles>` element may contain multiple `<MediaFile>` elements (described in the next section), each of which must be of the duration defined in the Linear duration element. Minor variations resulting from the transcoding process are acceptable.

3.2.1.3 The <MediaFiles> Element

The `<MediaFiles>` element is a container for one or more `<MediaFile>` elements, each of which contains a CDATA-wrapped URI to the media file to be downloaded or streamed for the Linear creative.

A `<MediaFiles>` element may contain multiple `<MediaFile>` elements, each one best suited to a different technology or device. When an ad may be served to multiple audio platforms, one platform (i.e. device) may need the media file in a different file format than what another platform needs. More specifically, different devices are capable of playing audio files with different encodings and containers, and at different bitrates.

Thus, for ads delivered cross-platform, the DAAST document usually contains multiple alternative `<MediaFile>` elements, each with different container-codec versions and at a few different bitrates. Only the media file best matched to the audio player system should be used. The creative content should be the same for each media file.

Ad Server Implementation Note	For ads to be delivered cross-platform, the ad server should return a DAAST response containing multiple alternative `<MediaFile>` elements, each with different container-codec versions and at a few different bitrates.

The `<MediaFile>` element also has several attributes that the audio player uses to select a media file to play for the user. The audio player must choose only one media file to play, and should choose the one that will play best for the user on his or her device and with the devices existing capabilities (audio decoder, network connection, etc.).

3.2.1.4 Media File Attributes

The following attributes are available for the <MediaFile> element.

Required Attributes:

- **id:** an identifier for the media file
- **delivery:** either "`progressive`" for progressive download protocols (such as HTTP) or "`streaming`" for streaming protocols.
- **type:** MIME type for the file container. Popular MIME types include, but are not limited to "audio/vnd.wav" for .wav files, "audio/mpeg" for MP3, and "audio/mp4" for MP4

Optional Attributes:

- **codec:** the codec used to encode the file which can take values as specified by RFC 4281: http://tools.ietf.org/html/rfc4281
- ***bitrate:** (not used if optional `minBitrate` and `maxBitrate` are used) average bits per second for variable bitrate audio.
- **minBitrate and maxBitrate:** for progressive load audio, the bitrate value specifies the average bitrate for the media file; otherwise the minBitrate and maxBitrate can be used together to specify the minimum and maximum bitrates for streaming audio ads.
- **apiFramework:** identifies the API needed to execute an interactive media file
- **sampleRate:** a whole number that represents the audio sampleRate in Hz
- **tracks:** an integer representing the audio tracks built into the file. This number is used to identify how the audio file is designed for use with the listener's speakers. Represented with a decimal, the following numbers indicate which tracks are used:
    - **"1.**0" mono (one track)
    - "2.0" stereo (left and right tracks)
    - "5.0" surround (front right and left tracks, back right and left tracks, and a center track)
    - The number of tenths represents the number of subwoofers. For example, a value of "5.1" indicates surround sound (with 5 tracks) and 1 subwoofer.

Audio Player Implementation Note	Multiple media files are often included in creative elements in order to ensure that at least one of the files can display most optimally to the user. The audio player is required to choose only one `<MediaFile>` element to use, and should choose the one that will play best to the user on his or her device. Some audio players may only look at the first media file available and disregard the creative when it can't be played, but the first media file provided may not be the most appropriate media file to play to the user. The audio player should poll all media files before choosing one to play.

3.2.1.5 The <AdParameters> Element (Optional)

Some ad serving systems may want to send data to the media file when first initialized. For example, the media file may use ad server data to identify the context used to display the creative, what server to talk to, or even which creative to play. The optional `<AdParameters>` element for the Linear creative enables this data exchange.

The optional attribute `xmlEncoded` is available for the `<AdParameters>` element to identify whether the ad parameters are xml-encoded. If present, the audio player must use xml to decode the data. Audio players may not be able to xml-decode data, so data should only be xml-encoded when being served to audio players capable of xml-decoding the data.

The `<AdParameters>` value in DAAST should be wrapped in a CDATA block.

General Implementation Note	The precise mechanism for passing the AdParameters information to the executable media for other APIs depends on the API framework that is used and whether the API is programmed to look for the `<AdParameters>` element or if some other mechanism is expected.

3.2.1.6 The <AdInteractions> Element (Optional)

While audio ads are not clicked in most cases, today's technology offers opportunities for listeners to interact with the content in other ways. For example, a listener may respond to an ad that uses voice recognition by saying, "LEARN MORE" or "CALL ME." Other methods for responding to advertiser content include device-supported functions such as shaking or tapping.

The `<AdInteractions>` element tracks interactions of two kinds: one that redirects the listener away from the ad-sponsored content and one that includes interactions with the ad while remaining in context of the ad-sponsored content. While audio ads are not typically clicked, the DAAST tracking elements use the word "click" for tracking ad interactions whether clicked or not.

The following three elements in `<AdInteractions>` help track ad interactions:

- **<ClickThrough>** contains a URI that redirects the listener away from the ad-sponsored content. The URI may also track the interaction.
- **<ClickTracking>** used in a Wrapper response, this element contains a URI for tracking the ClickThrough in the resulting Inline response. It may also be used to track the ClickThrough for an Inline response when the `<ClickThrough>` URI doesn't also track the click, such as when an interactive creative handles the click.
- **<CustomClick>** contains a URI to track any kind of interaction with the ad in which completion of the interaction either keeps the listener interacting with the ad or returns the listener to the ad-sponsored content.

The following attributes can be used with the three click elements listed above to describe designed interactions:

- **method:** describes how the listener will interact. (ex: voice, shake-up, tap)
- **actionType:** describes the action that results upon listener interaction. (ex: phone call, opening a webpage, sending an email, etc.)

General Implementation Note	The ad vendor can use the attributes, method and actionType, to describe the ad interaction as designed; they are not a request for the player to define the interaction during ad play.

3.2.1.7 The <TrackingEvents> Element (Optional)

In an effort to establish a baseline for communication on ad metrics in video, the IAB worked with member companies to define metrics for digital video. However, industry-defined metrics have not yet been defined for digital audio. In VAST the `<TrackingEvents>` element is used to track these industry-defined metrics in video. In DAAST, the `<TrackingEvents>` element can be used to track any of the digital video metrics that apply to digital audio ads.

Establishing industry-defined metrics for digital audio is important to the IAB and its members. When completed, those metrics will be provided in a DAAST update. Until then, the `AdInteractions/CustomClicks` element can be used to track any metrics that don't fall under the industry-defined metric events for digital video.

The IAB metrics are defined within the `<TrackingEvents>` element using the `<Tracking>` element. The attribute, `event`, is used to name the metric.

The following values are accepted for the `event` attribute as applicable to audio:

- **creativeView:** not to be confused with an impression, this event indicates that an individual creative portion of the ad was viewed (or played). An impression indicates the first frame of the ad was displayed; however an ad may be composed of multiple creatives or creative that only plays on some platforms and not others. This event enables ad vendors to track which ad creative are viewed, and therefore, which platforms are more common.
- **start:** this event is used to indicate that an individual creative within the ad was loaded and playback began. As with creativeView, this event is another way of tracking creative playback.
- **firstQuartile:** the creative played for at least 25% of the total duration.
- **midpoint:** the creative played for at least 50% of the total duration.
- **thirdQuartile:** the creative played for at least 75% of the duration.
- **complete:** The creative was played to the end at normal speed.
- **mute:** the user activated the mute control and muted the creative.
- **unmute:** the user activated the mute control and unmuted the creative.
- **pause:** the user clicked the pause control and stopped the creative.
- **rewind:** the user activated the rewind control to access a previous point in the creative timeline.
- **resume:** the user activated the resume control after the creative had been stopped or paused.
- **skip:** the user activated a skip control to skip the creative after a duration indicated by the value in the `offset` attribute.
- **progress:** the creative played for a duration at normal speed that is equal to or greater than the value provided in an additional attribute for `offset`. Offset values can be time in the format `HH:MM:SS` or `HH:MM:SS.mmm` or a percentage value in the format `n%`. Multiple progress events with different values can be used to track multiple progress points in the Linear creative timeline.

3.2.1.8 Multiple Tracking Events of the Same Type

The use of multiple tracking events of the same kind enables the ad server to share impression-tracking information with other ad serving systems such as a vendor ad server employed by the advertiser. When multiple tracking events of the same type (i.e. multiple "`start`" events) are provided, the audio player should send tracking info for the same events simultaneously or as close in time as possible. Any significant delay between requests may result in count discrepancies between ad serving systems.

Audio Player Implementation Note	If multiple tracking events of the same type are provided, the player should send tracking info for each at the same moment in time (or as close as possible in time). If any of the requests are delayed significantly, discrepancies may result in the participating ad serving system counts.

3.2.1.9 The <Icons> Element (Optional)

In an effort to support consumer privacy, DAAST offers the `<Icons>` element to provide information about behavioral tracking and other privacy information with each ad. Advertisers can do this by providing an interactive icon, such as the Ad Marker provided by the Digital Advertising Alliance (DAA). Only players with a visual component are required to display any icons.

Details about the structure and features of the `<Icons>` element are provided in section 3.3.3.

3.2.2 Companion Ad Format

Linear ads are often served with Companion ads—ads served outside of the audio player on the publisher page. When these ads are served with Linear ads placed before the audio content, the format is commonly called “pre-roll with companion.” Audio players may choose whether they want to support this format or not. Companion ads are usually not served alone and should be served with a Linear ad.

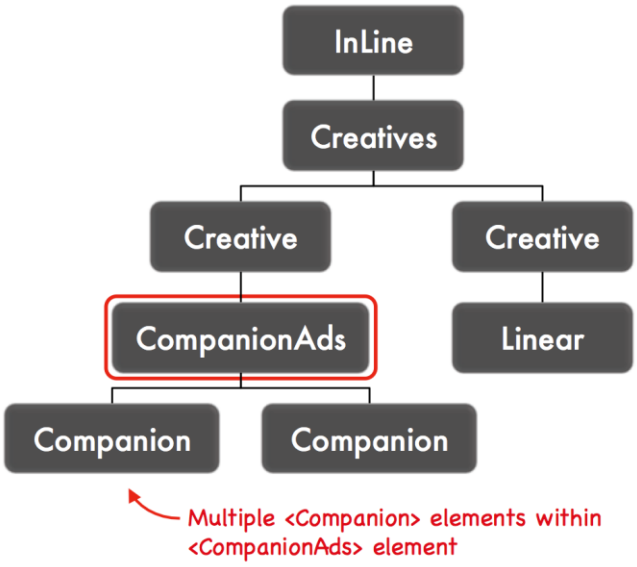
The DAAST response for a Linear Ad with Companions contains both a `<Linear>` creative and a `<CcompanionAds>` creative, as represented in the following example:

```
…
<InLine>
      <Creatives>
            <Creative>
                  <Linear>
                  …
                  </Linear>
            </Creative>
            <Creative>
                  <CompanionAds>
                        <Companion>
                        …
                        </Companion>
                  </CompanionAds>
            </Creative>
      </Creatives>
</InLine>
…
```

3.2.2.1 Companion Ad Structure

Unlike the `<Linear>` element, the `<CompanionAds>` (plural) element may contain one or more Companion ads, each Companion within its own `<Companion>` (singular) element.

The following diagram illustrates the structure of a DAAST InLine Ad containing Linear and CompanionAds creative.



Each `<Companion>` element must specify at least one resource file that may be one of: `<StaticResource>`, `<IFrameResource>` or `<HTMLResource>`. The resource used identifies the format of the creative file and provides a CDATA-wrapped URI to the file.

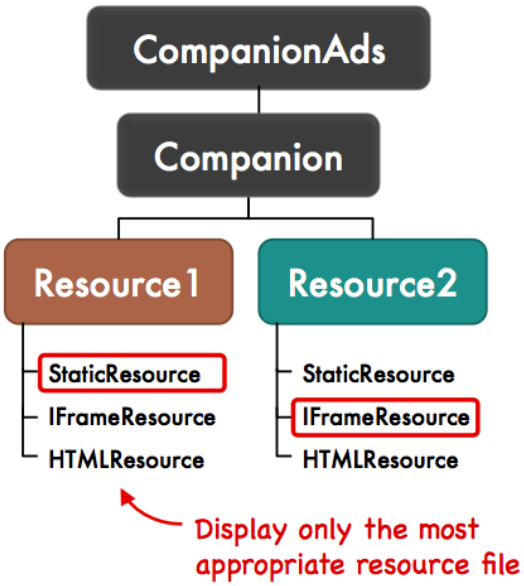
3.2.2.2 Companion Resource Elements

Companion resource types are described below:

- **StaticResource:** Describes non-html creative where an attribute for creativeType is used to identify the creative file format. The audio player uses the creativeType information to determine how to display the resource:
  - **Image/gif, image/jpeg, image/png:** displayed using the HTML tag `<img>` and the resource URI as the src attribute.
  - **Application/x-javascript:** displayed using the HTML tag `<script>` and the resource URI as the src attribute.
  - **application/x-shockwave-flash:** displayed using a Flash™ player
- **IFrameResource:** Describes a resource that is an HTML page that can be displayed within an Iframe on the publisher's page.
- **HTMLResource:** Describes a "snippet" of HTML code to be inserted directly within the publisher's HTML page code.

DAAST 1.0 allows for multiple resource files in one `<Companion>` element. The audio player should poll the resource files in each `<Companion>` element to find the most appropriate file to display.

The following diagram illustrates a <Companion> element with two resource files. The audio player should display the Companion creative using the most appropriate resource file provided.



Audio Player Implementation Note	DAAST 1.0 allows multiple resource elements for one <Companion> element. The audio player should poll <Companion> elements to find the most appropriate creative to display. For example, if the content publisher accepts html resource files, then the audio player should look for the <HTMLResource> element among the available resource files in the <Companion>.

The following DAAST example is a sample of a <Companion> element with multiple resource files:

```
<CompanionAds required=”all”>
      <Companion id=1>
            <StaticResource creativeType=”image/jpg”>
                  <![CDATA[http://AdServer.com/companion1.jpg]>
            </StaticResource>
            <HTMLResource>
                  <!CDATA[http://AdServer.com/companion1.html]>
            </HTMLResource>
      </Companion>
      <Companion id=2>
            <StaticResource creativeType=”image/jpg”>
                  <![CDATA[http://AdServer.com/companion2.jpg]>
            </StaticResource>
            <HTMLResource>
                  <!CDATA[http://AdServer.com/companion2.html]>
            </HTMLResource>
      </Companion>
</CompanionAds>
```

3.2.2.3 Other Companion Elements

The following elements are optional under the <Companion> element:

- **AltText:** used to provide an image description that displays when a user's cursor moves over the Companion creative
- **CompanionClickThrough:** provides a URL to an advertiser-related page when the user clicks the ad; only necessary for static resource files that lack technology to provide a clickthrough
- **CompanionClickTracking:** used to track Companion clickthroughs
- **TrackingEvents:** a container for the <Tracking> element used to track defined metrics defined by the event attribute
- **AdParameters:** used to pass information to the creative unit; includes the attribute xmlEncoded that is a Boolean value for identifying whether the <AdParameters> value is xml encoded.

3.2.2.4 The "required" Attribute for CompanionAds

In DAAST 1.0, the optional required attribute for the <CompanionAds> element provides information about which Companion creative to display when multiple Companions are supplied and whether the Ad can be displayed without its Companion creative. The value for required can be one of three values: all, any, or none.

The expected behavior for displaying Companion ads depends on the following values:

- **all:** the audio player must attempt to display the contents for all <Companion> elements provided; if all Companion creative cannot be displayed, the Ad should be disregarded and the ad server should be notified using the <Error> element
- **any:** the audio player must attempt to display content from at least one of the <Companion> elements provided (i.e. display the one with dimensions that best fit the page); if none of the Companion creative can be displayed, the Ad should be disregarded and the ad server should be notified using the <Error> element
- **none:** the audio player may choose to not display any of the Companion creative, but is not restricted from doing so; the ad server may use this option when the advertiser prefers that the master ad be displayed with or without the Companion creative

If not provided, the audio player can choose to display content from any or none of the <Companion> elements. In all cases when Companions are displayed, the audio player should display Companion creative at the same time as the Linear master creative.

3.2.2.5 Companion Attributes

The following attributes are available for the <Companion> element.

Optional attributes:

- **width:** the pixel width of the placement slot for which the creative is intended
- **height:** the pixel height of the placement slot for which the creative is intended
- **id:** an optional identifier for the creative
- **assetWidth:** the pixel width of the creative
- **assetHeight:** the pixel height of the creative
- **expandedWidth:** the maximum pixel width of the creative in its expanded state

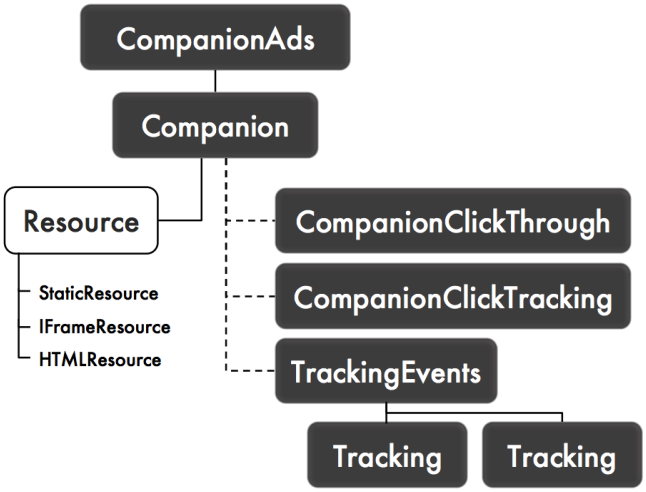
- **expandedHeight:** the maximum pixel height of the creative in its expanded state
- **apiFramework:** the API necessary to communicate with the creative if available
- **adSlotID:** used to identify desired placement on a publisher's page; values to be used should be discussed between publishers and advertisers
- **required:** a value of either "all," "any," or "none" identifying whether (and how many) of the companion creative should be displayed with the ad (see section 3.2.2.4 for details)
- **logoTile:** a Boolean value that identifies the companion as a logo tile. One way to use companion ads in digital audio is to display an ad's logo or other thumbnail-sized image in place of the artist tile for content when the ad is playing. Additional logo- attributes identify the title, artist, and URL listed below.
- **logoTitle:** A display title for the logo file being displayed.
- **logoArtist:** typical nomenclature for display of artist tiles, this field is used to identify the brand for which the logo is being displayed.
- **logoURL:** the URL for directing listeners to brand content if the listener interacts with the tile.

3.2.2.6 The Optional adSlotID Attribute for <Companion>

Advertisers and publishers can use the `adSlotID` attribute to match Companion creative to appropriate placement areas reserved on the Publisher's page. Values for this attribute have yet to be determined, as long as the value returned by the ad server matches one offered by the audio player and the dimensions are compatible, the audio player should respect the ad server's suggestion.

3.2.2.7 Companion Tracking Details

A `<Companion>` element may contain the `<TrackingEvents>`, `<CompanionClickThrough>` or `<CcompanionClickTracking>` elements, as represented in the following diagram.



The `<TrackingEvents>` element may contain one or more `<Tracking>` elements, but the only event available for tracking under each Companion is the `creativeView` event. The `creativeView` event tracks whether the Companion creative was viewed. This view does not count as an impression because impressions are only counted for the Ad and the Companion is only one part of the Ad.

The `<CompanionClickThrough>` element is provided to enable a clickthrough for any static resources that cannot provide clickthroughs of their own.

The `<CompanionClickTracking>` element is used to track clicks in the companion creative when the creative handles the clickthrough using an interactive API such as VPAID.

See section 3.1.5.2 for details about when to use the clickthrough and click-tracking elements.

As with all tracking events that are implemented, the audio player should send a request to the tracking URI provided. When multiple `creativeView` tracking events are provided, the audio player must send requests for each of the events provided simultaneously or as close in time as possible. Any delay in sending requests for all `creativeView` tracking events may result in count discrepancies for the participating ad servers.

Ad Server Implementation Note	A `<CompanionClickThrough>` element is provided to enable a clickthrough for Companion creative that is a `<StaticResource>`. Creative resources that are provided under `<IFrameResource>` and `<HTMLResource>` can provide their own clickthrough.

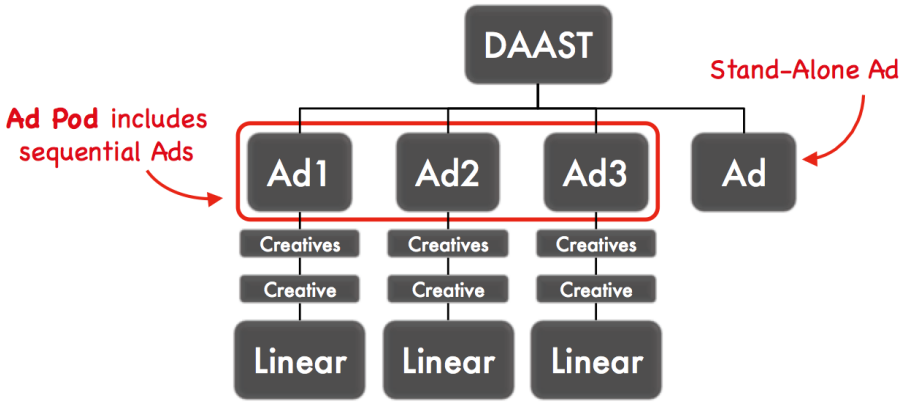
3.2.3 Ad Pods

A pod of ads is a sequence of Linear ads that are played back to back such as commercial breaks during a broadcast radio program. Pods can be used to create a Radio-like ad experience.

3.2.3.1 Ad Pods and Stand-Alone Ads

A pod of ads is described by a single DAAST response with multiple `<Ad>` elements, each with a distinct `sequence` attribute, starting with 1 and numbered sequentially. Only Linear ads can comprise a Pod and all `<Ad>` elements with sequence numbers are part of the Pod. All `<Ad>` elements with no sequence number are not part of the Pod and are considered to be stand-alone ads.

The following diagram represents the structure of a DAAST response with three sequential ads considered part of an Ad Pod and one stand-alone Ad.



A Pod of ads is intended to be played in sequence and in its entirety. In this format, a DAAST response can contain only one Ad Pod. Stand-alone ads can be considered part of an “ad buffet” from which a audio player can choose as many or as few ads as needed in a given circumstance. Stand-alone ads may be provided as a secondary choice when the Ad Pod cannot play or when a particular Ad in the Pod cannot play. When an Ad Pod follows a Wrapper, attributes can be used to manage which ads should be played and are described in section 3.3.1.2.

When the Ad Pod is served as a singular Inline response (without a Wrapper), playing the Pod or selecting one or more stand-alone ads from the buffet is left to the audio player’s discretion and may involve instructions passed to the player by mechanisms outside of the scope of DAAST. The IAB VMAP guideline includes such an option. See section 1.4.3 and the IAB VMAP guideline document for more information.

3.2.3.2 Playing a Pod of Ads

When electing to play a Pod of ads returned by the ad server, the audio player should play the ads in the Pod in the prescribed sequence and should play as many of the ads as possible. The player may elect not to play all of the ads (truncating the Pod from the end) if either: the ads cannot be played because they cannot physically fit into the stream (such as when time is limited in a live stream) or if the entire Pod of ads returned by the ad server violated any limits specified by the calling audio player (i.e. number of ads to return, or maximum Pod duration).

When an Ad Pod is the result of following a DAAST `<Wrapper>` the same impression and tracking URIs in the DAAST `<Wrapper>` are called as each Ad is played in the Pod.

Should an Ad in a Pod fail to play after a “no ad” response from a secondary ad server, the audio player should substitute an un-played stand-alone Ad from the response if provided. See section 3.3.2.4 for details on a no ad response.

3.2.3.3 Ad Pod Example

In the following DAAST example, the first three `<Ad>` elements form the Pod; the last two are stand-alone ads. The audio player must choose between displaying the three-ad Pod, or one or more of the standalone ads. Within the DAAST response, the three elements of the Pod need not appear in the right order or back to back, but the audio player must find and play Pod ads sequentially.

The example below shows an example of an ad pod in which elements are nicely organized (left), and another example in which the elements are out of order (right) but should yield the same result.

```
<DAAST>
      <Ad sequence=1>…</Ad>
<Ad sequence=2>…</Ad>
<Ad sequence=3>…</Ad>
<Ad>…</Ad>
<Ad>…</Ad>
</DAAST>
```

```
<DAAST>
      <Ad sequence=2>…</Ad>
<Ad>…</Ad>
<Ad sequence=1>…</Ad>
<Ad sequence=3>…</Ad>
<Ad>…</Ad>
</DAAST>
```

3.2.4 Skippable Linear Creative

Skippable Linear creative are creative that listeners can choose to skip, typically after a prescribed number of seconds have passed. Skippable creative is an innovative concept in

digital audio, and some companies have begun to experiment with technology such as voice command to enable the kind of interactions that would make skipping ads possible.

In support of audio creative that can be skipped, DAAST includes the following features:

- skipoffset attribute for the `<Linear>` element
- skip event
- progress event

3.2.4.1 Skipoffset Attribute

To specify that a Linear creative can be skipped, the DAAST ad file must include the `skipoffset` attribute in the `<Linear>` element. This `skipoffset` value indicates when the skip control should be provided after the creative begins playing. The value for `skipoffset` is a time value in the format `HH:MM:SS` or `HH:MM:SS.mmm`. The `.mmm` value in the time offset represents milliseconds and is optional. As an alternative, a percentage can be used in the format n%, where n represents the percentage of ad duration that must play before the skip control is made available.

Skipoffset examples:

Time skipoffset: The following example provides a skipoffset of :05 seconds.

```
<Creative>
      <Linear skipoffset="00:00:05">…</Linear>
</Creative>
```

Percentage skipoffset: The following example provides a skipoffset of 10%.

```
<Creative>
      <Linear skipoffset="10%">…</Linear>
</Creative>
```

Audio content publishers and advertisers should negotiate an acceptable `skipoffset` value. The audio player should send an error when a creative includes an unacceptable `skipoffset` value.

Audio players that support ads that the listener can skip must provide a "skip" control in the interface at the time indicated by an acceptable `skipoffset` value. If no `skipoffset` value is provided, then the creative is considered a standard Linear creative and may be handled as such.

The UI design for skip controls is left to the discretion of the audio publisher and can be negotiated with the advertiser.

3.2.4.2 Skip Event

The skip event is provided to support tracking Linear creative that is skipped and is only available for Linear creative. When the listener skips a Skippable creative, the audio player must request the tracking resource from the `skip` event URI provided.

The following example provides a tracking URI for the skip event in a DAAST response:

```
<TrackingEvents>
      <Tracking event="skip">
            <![CDATA[
            http://ad.server.com/skip/dot.gif
            ]]>
      </Tracking>
</TrackingEvents>
```

The `skip` event should not be confused with the `close` event. The `close` event should only be triggered if the user takes action to close the player or the window. The `skip` event, on the other hand, is triggered when a specific skip control is activated.

3.2.4.3 Progress Event

Whether or not an ad is skipped, advertisers and publishers need the flexibility to negotiate when a Skippable Linear creative counts as an impression. For example, some vendors who support skippable ads may count an impression when at least 30 seconds of the creative has played. The progress event includes an `offset` attribute that provides a time value (`HH:MM:SS` or `HH:MM:SS.mmm`) or percentage (`n%`) value that indicates the timing for recording an impression. The `creativeView` event can be used to track a view in this case, but details should be negotiated between the publisher and advertiser.

The following example provides a tracking URI for a `progress` event that is triggered after the Linear creative has played for at least 30 seconds.

```
<TrackingEvents>
      <Tracking event="progress" offset="00:00:30.000">
            <![CDATA[
            http://ad.server.com/view.gif
            ]]>
      </Tracking>
</TrackingEvents>
```

Video players that support Skippable Linear Ads must send a request to the URI provided for the progress event, if one is provided. If `progress offsets` are provided in percentage values when duration is unknown, then the video player can ignore the progress event. Multiple progress events with different `offset` values may be used to track different time points in the Linear creative timeline.

Regardless of the progress event record, publishers and advertisers must negotiate the terms for counting views based on progress events.

General Implementation Note	Video players complying with the Skippable Ad format must support the `progress` event, but actual counts based on `progress` values are dependent on terms negotiated between the publisher and advertiser.

The progress event provides metrics comparable to the quartile tracking events (i.e. first quartile, midpoint, third quartile, complete) when progress offsets are set at 25%, 50%, 75% and 100%. However, progress events are tracked separately so quartile events must still be supported when provided.

3.3 General DAAST Requirements

In DAAST 1.0, certain general areas of DAAST technology must be supported. These areas of technology are:

- **DAAST Wrapper Ads (Ad Server Redirects):** enables cross-platform interoperability when multiple ad serving systems are involved.
- **Error Reporting:** enables improved diagnostics across the industry, reducing errors and improving the overall audio ad experience for user and for the systems involved.
- **Industry Icon Support (for visual players only):** supports delivery, display, and tracking of industry icons such as the AdChoices program offered by the DAA.
- **Macros:** supports metadata collection for servers.

The following sections provide details about supporting these areas of DAAST Technology.

3.3.1 DAAST Wrapper Ads (Ad Server Redirects)

Wrapper ads provide a way for one ad server to redirect an audio player to a secondary ad server in order to retrieve an ad, multiple ads, or yet another DAAST Wrapper.

One ad server may redirect to another for a variety of reasons:

The first ad server has selected a specific advertiser campaign to fill the inventory, where the advertiser or a secondary ad server is hosting the DAAST creative for the campaign. In this case the redirect instructs the secondary ad server to return specific ads from a particular ad campaign.

The first ad server is delegating a specific piece of inventory for either a single ad or an entire Pod of ads to the secondary ad server to fill with any ads that are within an established agreement between the two parties.

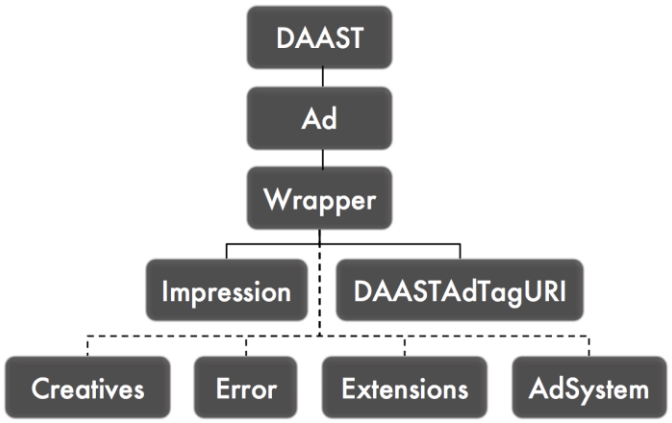
An ad server may have no ad to return and may return a redirect to a backfill provider.

The following sections describe DAAST Wrapper responses in detail.

3.3.1.1 General Wrapper Structure

A DAAST Wrapper is used to redirect the audio player to a secondary location for the Ad's resource files and can also redirect to yet another DAAST response. Using tracking events in the Wrapper, impressions and interactions can be tracked for the Ad that is eventually displayed.

The structure of a Wrapper Ad is illustrated in the diagram below. Dotted lines show optional elements.



Directly under the `<Wrapper>` element are two required elements:

- **<Impression>:** Contains a URI to a tracking resource that is requested when the Inline Ad is displayed
- **<DAASTAdTagURI>:** The redirecting URI to the next DAAST response, which may be either another Wrapper ad or the Inline response

Four optional elements are also available:

- **<AdSystem>:** The name of the system serving the DAAST Wrapper response; the attribute version can be used to identify the DAAST version used by the system
- **<Error>:** A URI to a tracking resource that is requested when an error has occurred either in the Wrapper itself, any subsequent Wrappers, or in the final Inline Ad.
- **<Extensions>:** A custom XML extension; when used, a custom element should be nested under `<Extensions>` to help separate custom XML elements from DAAST elements. The following example includes a custom xml element within the Extensions element.
  `<Extensions> <CustomXML>…</CustomXML></Extensions>`
- **<Creatives>:** Contains a creative element for including a Companion ad or tracking elements to track the subsequently served Inline ad

3.3.1.2 Wrapper Chain and Multiple Ad Management

As a gateway to additional Wrappers in a chain of DAAST responses, the Wrapper is in a position to prevent subsequent Wrappers from being served or to instruct the audio player on what to do with the ads in subsequent Wrappers.

For example, a Wrapper may redirect the audio player to a network for the Ad. If the target network has no ads to offer, it may redirect to yet another network for the Ad. However, the originating Wrapper may want to limit Ad selection to only the target network.

In another example, a Wrapper may represent an Ad in a Pod of ads and may want to allow only one Ad to fill its place in the Pod. Or in the case of a no-ad response, the Wrapper may provide instruction for how to select from additional ads to fill the empty opportunity placement.

Wrapper chain and multiple Ad management is handled with three attributes for the <Wrapper> element in DAAST:

- **followAdditonalWrappers:** a Boolean value that identifies whether subsequent wrappers after a requested DAAST response is allowed. If false, any Wrappers received (i.e. not an Inline DAAST response) should be ignored. Otherwise, DAAST Wrappers received should be accepted.
- **allowMultipleAds:** a Boolean value that identifies whether multiple ads are allowed in the requested DAAST response. If true, both Pods and stand-alone ads are allowed. If false, only the first stand-alone Ad (i.e. no sequence value for the Ad) in the requested DAAST response is allowed.
- **fallbackOnNoAd:** a Boolean value that provides instruction for using an available Ad when the requested DAAST response returns no ads. If true, the audio player should select from any stand-alone ads available. If false and the Wrapper represents an Ad in a Pod, the audio player should move on to the next Ad in a Pod; otherwise, the audio player can follow through at its own discretion where no-ad responses are concerned.

Infinite Loops

When serving an Ad involves a chain of Wrappers, an infinite loop is possible where a chain of Wrappers never results in a final InLine DAAST response. The audio player can be programmed to detect these loops and react accordingly, or may limit the total number or wrapper redirects.

Audio Player Implementation Note	The audio player should be aware of infinite Wrapper loops and be prepared to respond either with an <Error> or other appropriate action.

3.3.1.3 Wrapper Creative

In some cases, the Companion creative for an Ad may be included with resource files in the Wrapper, while redirecting the audio player to another server for the Inline Linear portion of the Ad.

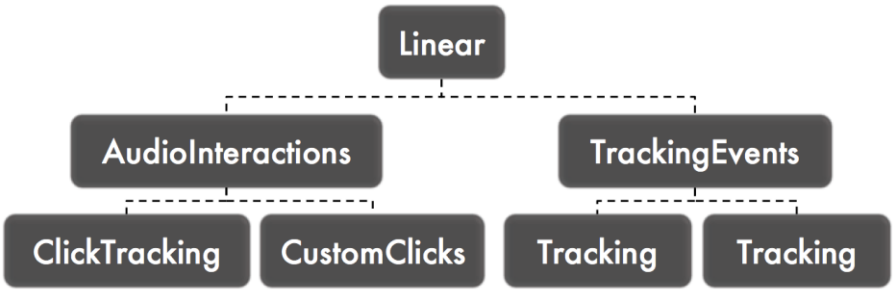
Creative elements in a Wrapper are used to provide files for Companion ads or to collect tracking information on the InLine creative that are served subsequent to the Wrapper. If the <Creatives> element is included in the Wrapper, one or more <Creative> elements may be included (but are not required; an empty <Creatives> element is acceptable). At most, each <Creative> element may contain one <Linear> or <CompanionAds> element.

Wrapper creative differ from InLine creative. The following sections describe each in detail.

3.3.1.4 Linear Creative Format within a Wrapper

The most important difference between a Wrapper Linear creative and an Inline one is that a Wrapper Linear creative is absent of any media files. The only elements allowed in a Wrapper Linear creative are <AudioInteractions> and <TrackingEvents>. These tracking elements enable tracking data to be collected at the Wrapper for any events that occur in the Inline Linear creative that is served following the Wrapper. See section 3.1.5.1 for more information on Linear tracking events.

A Linear creative in a DAAST Wrapper is structured as illustrated in the following diagram:

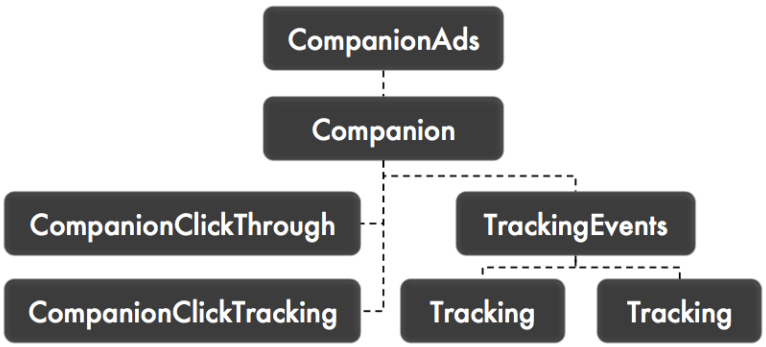


Audio Player Implementation Note	The audio player must send requests to each of the tracking URIs provided in the Wrapper elements under `<TrackingEvents>` and `<AudioInteractions>` whenever the associated tracking event occurs in the InLine Linear Ad that is served.

3.3.1.5 Companion Creative Format in Wrapper Ads

Unlike Linear creative, Companion creative can be served directly within a DAAST Wrapper response but may also be used for tracking purposes. When Companion creative are not included in the Wrapper response, only Companion tracking events (which are optional) need to be provided.

The structure for a Companion in a Wrapper response that is used for tracking purposes is illustrated in the following diagram:



An Inline Companion clickthrough can be tracked in the Wrapper using the `<CompanionClickTracking>` element. See section 3.1.5.2 for details about when to use the clickthrough and click-tracking elements.

Ad Server Implementation Note	The `<CompanionClickTracking>` element can be used in a Wrapper to track clicks on InLine Companion creative. However, correct association of the Inline clickthrough to the appropriate Wrapper tracking element may not be an exact match when multiple Companions are offered.

When multiple Companion creative are included in the Inline response, identifying which Companion clickthrough event should be associated with the Wrapper tracking element can be difficult. The audio player may associate Inline Companion clickthrough activity to Wrapper <CompanionClickTracking> events at its own discretion. The Companion id attribute may be a useful association if provided, or the audio player can match width and height attributes.

Audio Player Implementation Note	The audio player must attempt to associate Inline Companion clickthrough activity with appropriate <CompanionClickTracking> elements in the Wrapper if provided. Methods for association are at the video player's discretion.

The structure for a DAAST Wrapper response that serves Companion creative directly within the Wrapper is identical to the structure for an Inline DAAST response and is illustrated in section 3.2.2.1.

When the audio player displays a Companion Ad from creative that was provided directly within the <Wrapper> element, the audio player should track the Companion Ad the same way it would track an Ad provided in an <InLine> element.

3.3.1.6 Wrapper Conflict Management and Precedence

When Companion creative are included directly in the Wrapper response, conflict may occur. In a DAAST Ad, whether served with multiple Wrappers or in one Inline response, all creative offered is intended to be part of the same creative concept, and the audio player should attempt to display all creative presented in the response (or in a chain of responses). However, when conflict occurs, the audio player should favor creative offered closest to the Inline response.

For example, if a Wrapper contains Companion creative and the Inline response also contains Companion creative, the Companion creative in the Inline response should be selected (unless both creative can be displayed without conflict).

In another example, if the Inline response is absent of any Companion creative but two or more Wrappers contain Companion creative, then creative for the Wrapper served closest to the Inline response should be favored. However, if multiple creative can be served without conflict, the audio player should attempt to display whatever creative it can.

3.3.2 Error Reporting

The <Error> element enables the audio player to provide feedback to ad servers when an Ad cannot be served. In DAAST 1.0, detailed error codes and specifications for format are provided to enable detailed error logging for better ad serving diagnostics.

Providing more detailed error codes enables stronger diagnostics and enables better technology development over time. If ad servers can collect more detailed information about why their ads or specific creative couldn't be served, they can improve their systems to produce fewer errors.

The <Error> element is an optional element nested within the <InLine> or <Wrapper> element. It is used to track errors for an Ad. An error for an Inline Ad that is part of a chain of wrapper ads will produce an error for each of the wrappers used to serve the Inline Ad.

An <Error> element is also provided at the root DAAST level and is primarily used to report a "No Ad" response. See section 3.3.2.4 for more information.

3.3.2.1 Ad Server Details: <Error> Element

An `<Error>` element includes a URI that provides a tracking resource for the error. This error-tracking resource is called when the audio player is unable to display the Ad.

The following example is a sample DAAST response that includes the `<Error>` element for an Inline Ad.

```
<InLine>
      …
      <Error>
            <![CDATA[http://adserver.com/error.gif]>
      </Error>
      …
</InLine>
```

If the ad server wants to collect more specific details about the error from the audio player (as listed in section 3.3.2.3), an `[ERRORCODE]` macro can be included in the URI.

3.3.2.2 Audio Player Details

If an error occurs while trying to load an Ad and the `<Error>` element is provided, the audio player or system acting on behalf of the audio player must:

Request the error source file using the URI provided.

Replace the `[ERRORCODE]` macro, if provided, with the appropriate error code listed in the table in section 3.3.2.3. At a minimum, error code 900 (Unidentified error) can be used, but a more specific error code benefits all parties involved.

If the Ad was served after a chain of Wrapper Ad responses, the audio player or system acting on behalf of the audio player must also return error details as listed above for each Wrapper response that also includes error parameters. Macro responses must be correctly percent-encoded per RFC 3986.

The following table lists DAAST 1.0 error codes and their descriptions.

3.3.2.3 DAAST Error Codes Table

Code	Description
100	XML parsing error.
101	DAAST schema validation error.
102	DAAST version of response not supported.
200	Trafficking error. Audio player received an Ad type that it was not expecting and/or cannot display.
201	Audio player expecting different DAAST ad format.
202	Audio player expecting different duration.
203	Audio player expecting different size.

Code	Description
300	General Wrapper error.
301	Timeout of DAAST URI provided in Wrapper element, or of DAAST URI provided in a subsequent Wrapper element. (URI was either unavailable or reached a timeout as defined by the audio player.)
302	Wrapper limit reached, as defined by the audio player. Too many Wrapper responses have been received with no InLine response.
303	No Ads DAAST response after one or more Wrappers.
400	General Linear error. Audio player is unable to display the Linear Ad.
401	File not found. Unable to find Linear/MediaFile from URI.
402	Timeout of MediaFile URI.
403	Couldn't find MediaFile that is supported by this audio player, based on the attributes of the MediaFile element.
405	Problem displaying MediaFile. Audio player found a MediaFile with supported type but couldn't execute. MediaFile may include: unsupported codecs, different MIME type than MediaFile type, unsupported delivery method, etc.
406	Couldn't find MediaFile of requested duration. Audio player calls this only if and when the ad opportunity presents itself. See section 3.3.4 for defining the duration of the break when the `[BREAKLENGTH]` macro is provided.
600	General CompanionAds error.
601	Unable to display Companion because creative dimensions do not fit within Companion display area (i.e., no available space).
602	Unable to display Required Companion.
603	Unable to fetch CompanionAds/Companion resource.
604	Couldn't find Companion resource with supported type.
900	Undefined Error.
901	General API error.

3.3.2.4 No Ad Response

When the ad server does not or cannot return an Ad, the DAAST response should contain only the root <DAAST> element with optional <Error> element, as shown below:

```
<DAAST version="3.0">
      <Error>
            <![CDATA[http://adserver.com/noad.gif]>

      </Error>
</DAAST>
```

The DAAST <Error> element is optional but if included, the audio player must send a request to the URI provided when the DAAST response returns an empty InLine response after a chain of one or more wrapper ads. If an [ERRORCODE] macro is included, the audio player should substitute with error code 303.

Besides the DAAST level <Error> resource file, no other tracking resource requests are required of the audio player in a no-ad response in either the Inline Ad or any Wrapper ads.

3.3.3 Industry Icon Support

Several initiatives in the advertising industry involve using an icon to provide some extended functionality such as to communicate with consumers or otherwise fulfill requirements of a specific initiative. Often this icon and its functionality may be provided by a vendor, and is not necessarily served by the ad server or included in the creative itself.

One example of icon use is for compliance to certain Digital Advertising Alliance (DAA) self-regulatory principles for Interest-Based Advertising (IBA), previously known as Online Behavioral Advertising (OBA). This section provides an overview of how audio players can support the use of icons in a general manner while using the DAA's Advertising Option icon, commonly known as the "AdChoices" icon, as a specific example.

General Implementation Note	Supporting industry icons in digital audio is only required if the audio player has a built-in visual component in which the icon can be displayed.

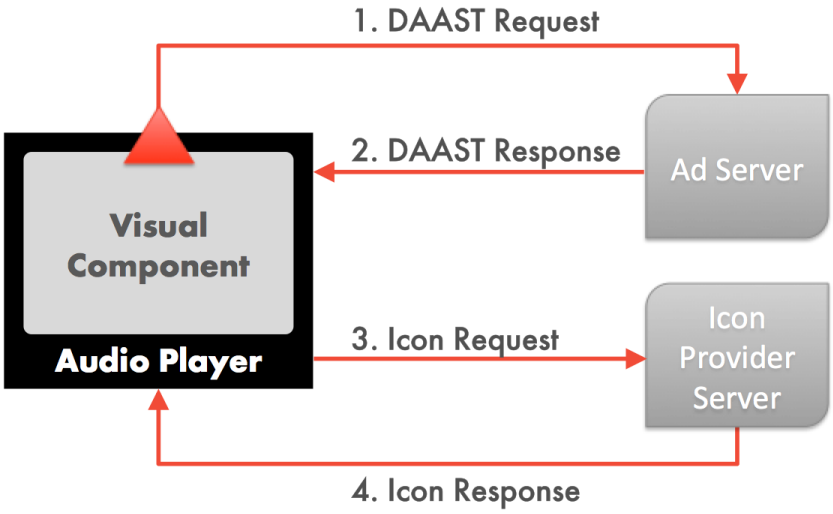
3.3.3.1 Icon Use Case: AdChoices for Interest-Based Advertising (IBA)

The DAA sets forth principles that endeavor to give consumers a better understanding of, and greater control over, ads that are customized based on the consumer's interests. This control is made available to the consumer in the form of the AdChoices icon, which is displayed in a prominent location in or around the Ad creative. When a consumer clicks the icon, they may be offered: information about the ad server and data providers used to select the Ad, options to learn more about IBA, and the ability for consumers to opt out from receiving IBA ads in the future.

3.3.3.2 The <Icons> Element

DAAST 1.0 includes support for the <Icons> element, which is offered under the <Linear> creative element for both Inline and Wrapper ads.

The following diagram illustrates the general process for how the <Icons> element is called in a DAAST response:



The Icon Provider Server may be the same server that serves the DAAST response but more commonly is a vendor that serves the icon from its own systems.

When the <Icons> element is included in the DAAST response, the player displays the object when the accompanying linear ad is played.

Audio Player Implementation Note	Since a vendor often serves icons and may charge advertising parties for each icon served, the player should not pre-fetch the icon resource until the resource can be displayed. Pre-fetching the icon may cause the icon provider to falsely record an icon view when the icon may not have been displayed yet.

3.3.3.3 Attributes for the <Icon> Element

The <Icons> element was designed to support multiple industry initiatives that involve icons. The most prevalent initiative at the release of DAAST 1.0 was the AdChoices program in the US. However, other programs exist and future programs may develop.

To support multiple icon programs, the <Icons> element may include multiple <Icon> elements. Each <Icon> element includes attributes to tell the player how to display the icon when multiple icons are included as well as what to do when multiple icons of the same program are served with an Ad (as when a chain of wrapper ads each include their own icons). Details about handling precedence and icon collisions can be read in section 3.3.3.6. Required attributes indicate program, size and display location. Addition optional attributes enable other details for the video player.

The following <Icon> element attributes are required:

- **program:** Identifies the industry initiative that the icon supports. When icon elements of multiple programs are served in a chain of Wrapper ads, the player uses this information to display only one icon from each program.

- **height:** The height (in pixels) of the icon to be displayed.
- **width:** The width (in pixels) of the icon to be displayed.
- **xPosition:** The horizontal alignment location (in pixels) that the player uses to place the top-left corner of the icon relative to the ad display area (not necessarily the player display area). Accepted values are "`left`," "`right`," or a numeric value (in pixels). A value of "`0`" (zero) is the leftmost point of the display area.
- **yPosition:** The vertical alignment location (in pixels) that the player uses to place the top-left corner of the icon relative to the display area. Accepted values are "`top`," "`bottom`," or a numeric value (in pixels). A value of "`0`" (zero) is the topmost point of the display area.

The `xPosition` and `yPosition` attributes are used to position the icon relative to the display area. If the display area is resized, these values should be used to reposition the icon relative to the new display area of the Ad.

The following `<Icon>` element attributes are optional:

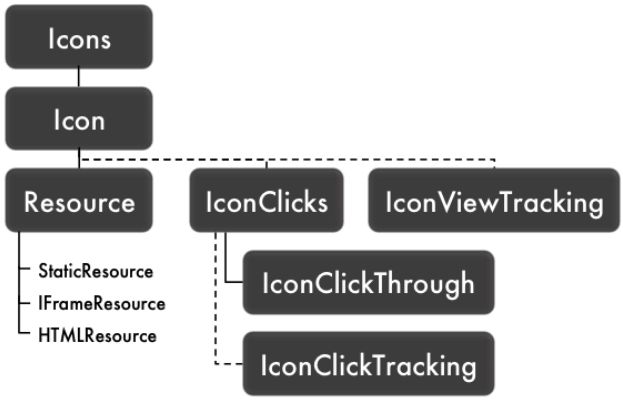
- **apiFramework:** The API needed to execute the Icon creative, if applicable.
- **offset:** Start time (in `HH:MM:SS` or `HH:MM:SS.mmm` format) for when the player should display the icon. The time indicated is offset from when the icon's associated `<Linear>` creative is first played.
- **duration:** The amount of time (in `HH:MM:SS` or `HH:MM:SS.mmm` format), for which the player should display the icon. If not present, the video player should display the icon while the ad is played or until the listener interacts with the ad or the icon, if applicable.

Audio Player Implementation Note	The audio player should display the icon element for as long as the ad is played or until the listener interacts with the ad or the icon. If the `duration` attribute is included, then the icon should be displayed for the duration indicated.

3.3.3.4 Structure of the <Icons> Element

The `<Icons>` element is a container for one or more `<Icon>` elements. Each `<Icon>` element must contain a resource element that is one of: `<StaticResource>`, `<IFrameResource>`, or `<HTMLResource>`. The resource element must contain a CDATA element that includes the URI to the icon resource file. Optional tracking elements are also provided so that the icon provider can track views and clicks.

The following diagram illustrates the structure of the <Icon> element. Dashed connector lines represent optional elements.



3.3.3.5 Icon Clicks and Tracking

One common goal of programs that use ad icons is to provide consumers with information. This information can be built into the resource file implemented for the icon, or an additional URI can be provided that opens a page when the user clicks the icon. The <Icon> element includes an optional <IconClicks> element used to load an informational page in a new window as well as track when the ad is clicked.

The <IconClicks> element is optional, but if provided must include one <IconClickThrough> element that contains a CDATA-wrapped URI to the information page. The player must load this page in a new window when the user clicks the icon. If the Icon resource is a scripted file, such as Flash™, the resource file may handle the clickthrough details and may not need to use the <IconClicks> element.

Optionally, the <IconClicks> element may also include one or more <IconClickTracking> elements used for tracking clicks, each with its own CDATA-wrapped URI to a tracking resource. When the user clicks the icon, the player simultaneously loads the <IconClickThrough> URI in a new window and the tracking resources for any <IconClickTracking> elements. The following example shows a simplified section of a DAAST response where the <Icon> element uses the <IconClicks> element.

```
<Icon>
      …
      <IconClicks>
            <IconClickThrough>
                  <![CDATA[http://iconprovider.com/info]>
            </IconClickThrough>
            <IconClickTracking>
                  <![CDATA[http://iconprovider.com/click.gif]>
            </IconClickTracking>
      </IconClicks>
</Icon>
```

To track icon views (similar to tracking impressions for ads), the <Icon> element provides an optional <IconViewTracking> element. This element should contain a CDATA-wrapped URI to a tracking resource file that the player must load only *after* the icon is visible to the user. If a time offset value is indicated using the offset attribute in the <Icon> element, the video player must withhold loading the tracking resource until the time indicated.

The following example focuses on the <IconViewTracking> element in a DAAST response where the <Icon> element includes the offset attribute:

```
<Icon [required attributes] offset="00:00:05">
      …
      <IconViewTracking>
            <![CDATA[http://iconprovider.com/view.gif]>
      </IconViewTracking>
</Icon>
```

The tracking resource for the <IconViewTracking> element in the example above should not be loaded until 5 seconds after the creative served with the icon is visible to the user.

3.3.3.6 Precedence and Conflict Management:

As an ad goes through a delivery chain, companies may include their own icon element in their wrapper responses. Sometimes these multiple icon elements are all for the same program and the player must decide on only one icon to display. When icon elements represent more than one program, one icon from each program should be displayed.

The player can use its own business rules to decide which icon to display, along with any specific program recommendations. For example, when multiple AdChoices icons are offered, the DAA program recommendation is to select the icon that is closest to the creative.

If no other rules govern the selection of which icon to display, the player should choose the one closest to the creative. That is, if the <Icon> element is included within the Inline Ad, then that icon is the closest to the creative. However, if the Inline Ad contains no <Icon> element, but the last Wrapper Ad in a chain of Wrappers did contain the <Icon> element, then the icon from that last Wrapper Ad is the one closest to the creative.

When multiple icons from more than one icon program are included in a chain of Wrapper ads, the player must decide which icon from each program should be displayed. Again, the player can use its own business rules; however, the icons must not overlap each other. If all program icons use the same xPosition and yPosition values, the player can use width and height attribute values to offset coordinates relative to the display area.

3.3.3.7 Icons in Companion Ads

The existing DAAST elements for `<CompanionAds>` can each include multiple `<Companion>` elements, respectively, which enables ad servers to include icons with these creative types without using the `<Icon>` element.

The following example of a creative element shows how an industry icon can be implemented for Companion ads:

```
<Creative>
      <CcompanionAds>
            <Companion>
                  <!--link to Companion creative-->
            </Companion>
            <Companion>
                  <!--link to industry icon-->
            </Companion>
      </CompanionAds>
</Creative>
```

3.3.4 Macros

Sometimes ad servers would like to collect metadata from the audio player when tracking event URIs are accessed. For example, the position of the audio player playhead at the time a tracking event URI is accessed is useful to the ad server and is data that can only be known at the time of the prescribed tracking event. This data cannot be built into the URI at the time the DAAST response is built and served.

The following macros enable the audio player to provide certain details to the ad server at the time tracking URIs are accessed.

[ERRORCODE]: replaced with one of the error codes listed in section 3.3.2.3 when the associated error occurs; reserved for error tracking URIs.

[CONTENTPLAYHEAD]: replaced with the current time offset "`HH:MM:SS.mmm`" of the audio content.

[BREAKLENGTH]: replaced with the amount of time "`HH:MM:SS.mmm`" remaining for the ad break or pod. Used with error code 406.

[CACHEBUSTING]: replaced with a random 8-digit number.

[ASSETURI]: replaced with the URI of the ad asset being played.

When replacing macros, the video player must correctly percent-encode any characters as defined by RFC 3986.

DAAST doesn't provide any guidance on URI format, but using the `[CACHEBUSTING]` macro simplifies trafficking, enabling ad servers to easily search and replace the appropriate macro for cache busting.

4 Human Readable Schema

Element	Attributes	Required	Section
DAAST	**version**	**Yes**	**3.1.1**
/Error		No	
DAAST/Ad	**id, sequence**	**Yes**	**3.1.2**
DAAST/Ad/InLine		**Yes***	**3.1.4**
/AdSystem	version	No	3.1.4.2
/AdTitle		Yes	3.1.4.1
/Description		No	3.1.4.2
/Advertiser		No	3.1.4.2
/Expires		No	3.1.4.2
/Pricing	model, currency	No	3.1.4.2
/Survey		No	3.1.4.2
/Error		No	3.1.4.2
/Impression	id	Yes	3.1.4.1
/Category		Yes	3.1.4.1
/Creatives		Yes	3.1.6
/Creative	id, sequence, adId, apiFramework	Yes	3.1.6
/CreativeExtensions		No	3.1.6.3
/CreativeExtension			3.1.6.3
/Linear	skipoffset (optional)	Yes	3.2.1
/AdParameters	xmlEncoded	No	3.2.1.5
/Duration		Yes	3.2.1.2
/MediaFiles		Yes	3.2.1.3
/MediaFile	id, delivery, type, bitrate, minBitrate, maxBitrate, width, height, scalable, mantainAspectRatio, codec, apiFramework, sampleRate, tracks	Yes	3.2.1.4
/TrackingEvents		No	3.2.1.7
/Tracking	event	No	3.2.1.7
/AudioInteractions		No	3.2.1.6
/ClickThrough	id, method, actionType	No	3.2.1.6
/ClickTracking	id, method, actionType	No	3.2.1.6
/CustomClicks	id, method, actionType	No	3.2.1.6
/Icons		No	3.3.3.2
/Icon	program, width, height, xPosition, yPosition	Yes*	3.3.3.3

/Icons	duration, offset, apiFramework	No	3.3.3.3
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	Yes*	3.3.3.4
/IconClicks		No	3.3.3.5
/IconClickThrough		No	3.3.3.5
/IconClickTracking	id	No	3.3.3.5
/IconViewTracking		No	3.3.3.5
/CompanionAds	Required (3.2.2.4)	No	3.2.2
/Companion	id, width, height, assetWidth, assetHeight, expandedWidth, expandedHeight, apiFramework, adSlotID, logoTile, logoTitle, logoArtist, logoURL	No	3.2.2.1 (attributes in 3.2.2.5)
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	Yes*	3.2.2.2
/AdParameters	xmlEncoded	No	0
/AltText		No	0
/CompanionClickThrough		No	3.2.2.7
/CompanionClickTracking	id	No	3.2.2.7
/TrackingEvents		No	3.2.2.7
/Tracking	event	No	3.2.2.7
/Extensions		No	3.1.4.2
/Extension	type	Yes*	3.1.4.2
VAST/Ad/Wrapper	**followAdditonalWrappers, allowMultipleAds, fallbackOnNoAd**	**No**	**3.1.3** (diagram) 3.3.1.1 (general) 3.3.1.2 (attributes)
/AdSystem	version	No	3.3.1.1
/DAASTAdTagURI		Yes	3.3.1.1
/Error		No	3.3.1.1
/Impression	id	Yes	3.3.1.1
/Creatives		No	3.3.1.1
/Creative	id, sequence, adId	No	3.3.1.1
/Linear		Yes	3.3.1.4
/TrackingEvents		No	3.2.1.7
/Tracking		No	3.2.1.7
/AudioInteractions		No	3.2.1.6
/ClickTracking	id, method, actionType	No	3.2.1.6

/CustomClicks	id, method, actionType	No	3.2.1.6
/Icons		No	3.3.3.2
/Icon	program, width, height, xPosition, yPosition	Yes	3.3.3.3
	duration, offset, apiFramework	No	3.3.3.3
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	No	3.2.2.2
/IconClicks	creativeType (StaticResource only)	No	3.3.3.4
/IconClickThrough		No	3.3.3.5
/IconClickTracking		No	3.3.3.5
/IconViewTracking		No	3.3.3.5
/CompanionAds	required	No	3.2.2
/Companion	id, width, height, assetWidth, assetHeight, expandedWidth, expandedHeight, apiFramework, adSlotID, logoTile, logoTitle, logoArtist, logoURL	No	3.2.2.1 (attributes in 3.2.2.5)
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	No	3.2.2.2
/AdParameters	xmlEncoded	No	0
/AltText		No	0
/CompanionClickThrough		No	3.2.2.7
/CompanionClickTracking		No	3.2.2.7
/TrackingEvents		No	3.2.2.7
/Tracking	event	No	3.2.2.7
/Extensions		No	3.3.1.1
/Extension	type	Yes*	3.3.1.1

*Either one of listed elements is required or the requirement for the element is dependent on whether a parent element is used. In the case of Inline creative, at least on element of either Linear, or CompanionAds is required.

5 DAAST Terminology

The following terminology may apply to either digital video or digital audio environments. However, for the purposes of this document, the following terminology is described in the context of digital audio and the DAAST specification.

Ad–Stitching Server: A server that request ads on behalf of an audio player and inserts received ads into a stream of content that is served to the player. Upon serving the ad-stitched stream to the player, the ad-stitching server sends relevant tracking information on behalf of the player.

Ad Pod: An ad Pod is sequence of Linear ads played back-to-back, like a commercial break with multiple ad spots in an audio program.

Aggregator: An audio channel that collects and organizes audio content from other sources. Aggregators benefit listeners by providing a variety of content in one channel. The audiences for an aggregated channel of content offers greater reach for advertisers than a focused channel would provide.

Audio Player: An audio playback environment used to manage audio programming. Audio players are provided by an audio aggregator or can be custom-built by an audio content publisher.

Content Distribution Networks (CDN): Vendors that provide the technical heavy lifting of moving streamed digital audio content around the Internet and ensuring scalability and performance for listeners.

Companion Ad: Commonly, a display banner or rich media ad that appears within a visual component in an audio player or on a webpage where the audio player is implemented. Companion ads may remain on the page after the related in-stream ad ends. A companion ad can also be a skin that wraps the audio player.

Clickthrough: A URL forwarding page that opens the advertisers target webpage when a listener clicks (or taps) a companion ad that displays in the audio player or on the player's webpage.

DAAST: An acronym for Digital Audio Ad Serving Template, which is an IAB guideline and XML schema that describes the XML structure for an audio ad response. DAAST-enabled audio ad players can receive audio ad responses from any ad server.

DAAST Tag: A URI that returns a DAAST response in XML format when called.

DAAST Wrapper: A DAAST ad response from an ad server that points to another ad server for the ad (sometimes referred to as the downstream DAAST response).

Digital Audio: Audio programming available to consumers on a streaming basis, delivered via the wired and mobile Internet.

Digital Audio Ad: An audio ad played in the context of a digital audio program.

InLine Ad: A DAAST ad response that contains all the information needed to play a digital audio ad. No additional calls to other ad servers are needed after a DAAST InLine ad response is received.

In-Stream Audio Ad: An ad that plays within a streaming audio player.

Linear Ad: Linear ads are like radio commercials and can play before the audio programming plays (pre-roll), during a break in the audio program (mid-roll), or after the audio program ends (post-roll). Using an API or other technology, Linear ads can be made interactive using technology such as voice recognition, mobile device actions such as “shake” or “tilt,” or simply taking advantage of a visual component the listeners can click or tap.

Master Ad: For audio ad campaigns use companion ads, the linear portion of the ad unit that plays within the audio content is referred to as the master ad for the companion.

Podcasts: A podcast is an audio program that listeners can access on portable devices or their computers in an offline mode.

Primary Ad Server: The first ad server that the audio player calls to for ad content.

Pureplay Stream: Personalized and on-demand, streamed audio services that create playlists based on user preferences of artists, tracks, or genres.

Redirect: See “DAAST Wrapper”

Satellite: This system transmits audio with orbiting communication satellites rather than ground-based transmission.

Terrestrial Stream: Live digital audio streams of terrestrial radio stations.

Tracks: the channels of audio within a media file that make use of different speakers in a player’s speaker system. For example, a media file with one track is a file that plays in mono; two tracks plays in stereo using both the left and right speakers; a file with five channels plays in surround sound making use of five different speakers in the player’s sound system.

VAST: The Video Ad Serving Template is an IAB guideline and XML schema that describes the XML structure for a video ad response. VAST enables video ad responses to come from any ad server.

Vendor Ad Server: The ad server that the audio player calls after receiving a DAAST wrapper response (redirect) from the primary ad server or other vendor ad servers. Vendor ad servers may include agency servers, ad networks, or audio ad exchange platforms. Eventually, an ad server must provide a DAAST response that includes all the creative elements needed to display the ad.

VMAP: Video Multi Ads Playlist is an IAB guideline that describes the XML structure for a playlist of video ads sent from an ad server to a video player. The VMAP specification and guideline can be applied in the audio environment, but was not intended for use with audio.