

PLAYLISTS AND YOU

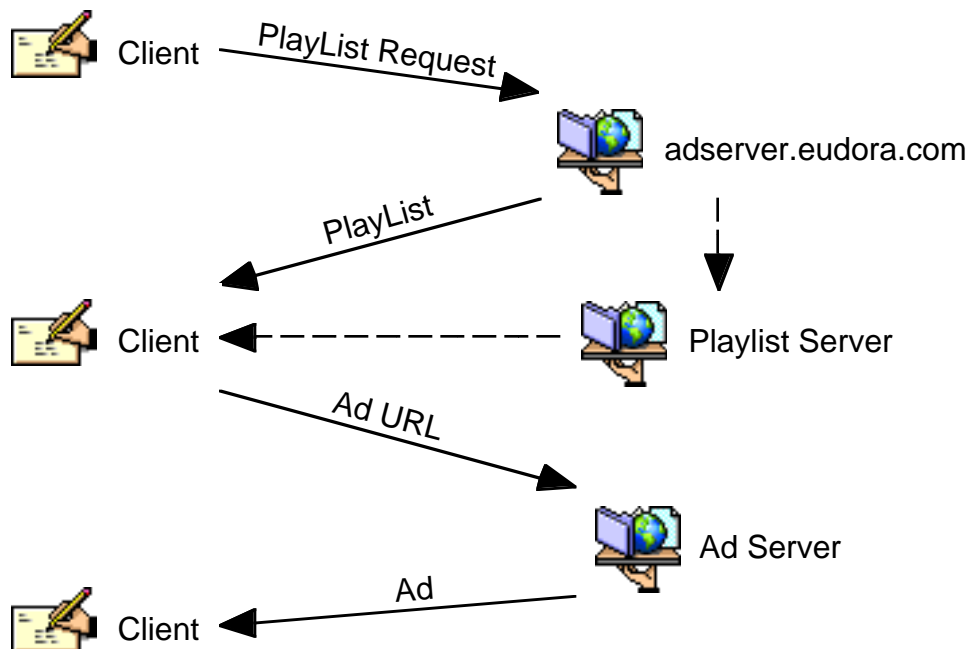
INTRODUCTION

The Playlist is a way to control the fetching and display of ads in Eudora. Its primary benefits are the separation of ad parameters from ad images, insulation of the Eudora client from intimate knowledge of ad image servers, and centralized server intelligence in ad distribution, without requiring user registration or centralized user databases.

PlayLists are extremely malleable objects. They can exert varying degrees of control over how the Eudora client behaves, from specifying the exact set of ads Eudora runs to simply transmitting abstract URI's that will choose their own ads. If PlayLists are used to their fullest advantage, they will give us a powerful tool in controlling ad display in Eudora; if they are later deemed irrelevant, they'll cost us one extra brief TCP connection per day.

THE BASIC PLAYLIST SYSTEM

The following chart illustrates the central use of the Playlist system; the client connects to a Playlist server (which may redirect to a different server), the Playlist server returns a playlist to the client, which then fetches the ads specified in the Playlist:



PLAYLIST REQUESTS

First we will discuss the playlist request, which is sent by Eudora to the Playlist server in order to initiate the ad fetch process. The playlist request is a block of XML¹ that gives the server the information it needs to build or select the proper playlist for the user. The information in the playlist request is as follows:

userAgent This is a string identifying the application requesting the playlist, its version number, and the platform on which it's

¹XML is an extraordinarily verbose way to transmit structured data. At least it's ASCII.

running.

playlist This identifies the playlist the client is currently using. This may have multiple values if the client is working off more than one playlist.

entry A list of the id's of the ads recently shown by this client. The entries are nested inside the playlist to which they belong.

Each entry can have zero or more of the following attributes:

active="0" meaning the ad is no longer being shown.

isRunout="1" meaning the ad is a runout ad. This saves the server having to do a lookup on the ad.

isSponsor="1" meaning the ad is a sponsorship ad, to be shown in place of the QUALCOMM.

isButton="1" meaning the ad is a toolbar button.

deleted="1" meaning the ad has been hidden by the user. This is allowed only for toolbar ads.

faceTime This lists the amount of face time the user has used in the last seven calendar days. This allows the server to determine how many ads the client is likely to be able to display. The value for the current day is the greater of today's value and last week's value for today.

faceTimeLeft This is a total of the amount of face time requested by the ads still left in the client's ad cache.

faceTimeUsedToday This is the amount of facetime the client has used toward displaying ads today. It can be used by the server to determine whether a date-critical ad can be shown today.

distributorID This id is used for the bounty system, so that the Playlist Server can identify and credit the ISP or other organization that gave away this copy of Eudora.

pastry This is a cookie the Playlist Server gave to Eudora in the past. It could contain any state the server wishes to save.

profile Profiling information entered from our web page.

screen.height The height of the display on which the ads are shown, in pixels.

screen.width The width of the display on which the ads are shown, in pixels.

screen.depth The depth of the display on which the ads are shown, in pixels.

playlistVersion The version # of the playlist spec implemented by this client.

Not all of these parameters are likely to be actively used at the same time; some are present to support particular modes of operation (see below), and will not be used in other modes.

Every playlist request is checksummed with MD5. Before computing the checksum, a secret seed is prepended to the request. The MD5 digest is encoded in hex and put in a "Checksum: " header in the request. The server may ignore requests that fail the checksum.

PLAYLIST RESPONSE

After the client makes a Playlist Request, the server replies with a Playlist Response. The Playlist Response is divided into two major sections; the ClientInfo section, which updates general client behaviors, and the Playlist itself, which describes the ads the client should fetch.

The Playlist Server may also return an empty response, meaning that the client should continue on its course with the ads it has.

Every playlist response is checksummed with MD5. Before computing the checksum, a secret seed is prepended to the playlist. The MD5 digest is encoded in hex and put in a "Checksum: " header in the response. Clients should ignore playlists that fail the checksum.

PLAYLIST RESPONSE - RESET

We have seen some clients become befuddled, due to old client bugs, server bugs, etc. Sometimes, the bad data inherited by even an updated client is too weird for the system to function properly. While we could make the client detect this, it seems better to leave the job to the server, which can be changed more easily. So, if the server detects that a client has wandered into the weeds and must be brought back into goodness, it responds with just a single command: **reset**. No ClientInfo should follow, no playlist should follow, just the reset command.

On receiving the reset command, the client should throw away all its ad databases and records, including playlists, facetime history, ad history, ad caches, etc. Everything should go back to what it would have been before the ad system was run for the first time. Link History is exempted from the reset command, both for reasons of practicality and because it is so user-visible. The only other item of ad data that reset does not affect is the ad failure counter, which should be retained across a reset. The client should then notice that it has no playlist, and make another request for one.

PLAYLIST RESPONSE - CLIENTINFO SECTION

The clientInfo section updates various client parameters. The parameters are:

- reqInterval** This is the number of hours the client should wait before checking for a new playlist. If ad turnover is high, this will be a small number. A sponsored freeware version might have a much higher number here, so that it checked for a new playlist only once a week or once a month. Clients may also check for new playlists if they have ads with nonzero showForMax values, and the ads have used up much of their time.
- histInterval** This value is the number of days the client must remember that it showed a particular ad. It will report this to the server so that the server can, at its discretion, choose not to show ads for competing services to that particular client.
- pastry** The abovementioned cookie. The server can store whatever state information it wishes in this cookie.

- flush** More command than parameter, if present it causes the client to discard an old playlist or ad. Flushed ads and playlists are removed completely, and no longer show up in ad histories.
- width** The width in pixels the client should make the ad window be.
- height** The height in pixels of same.
- facetimeQuota** The number of seconds of facetime the client should devote to regular ads, before moving to the runout ad.
- rerunInterval** The number of days an ad may be “rerun”; that is, shown for free after all other ads and the runout are exhausted. The time is measured from the last non-rerun showing of the ad.

The clientInfo section is a powerful feature of playlists. It allows us to control the application in a global way, including segueing smoothly from one ad model to another. It's hard to understand how this would be done if we didn't have some central place for the client to go to get instructions. *If this were the only benefit we got from playlists, it would make playlists worthwhile.*

PLAYLIST RESPONSE - THE PLAYLIST ITSELF

The playlist itself has one global value, which we will dispose of first:

- playlistID** This id is the one the client will tell the server it has the next time it connects.

The remainder of the playlist is a list of ads. Each ad is allowed to have many parameters, though not all of them will be used with a single ad, and it is possible that some of them will never be used at all.

Scheduling parameters:

- showFor** This is the number of seconds the ad should be shown for at any given time. This number might be small, like a TV ad (eg, 30), or large, more like a billboard (eg, 3600 for an hour all at once).
- showForMax** Maximum total amount of time to show this ad. The ad is exhausted after this time, and should be discarded once new ads arrive.
- dayMax** Maximum number of times per day to show this particular ad.
- blackBefore** The amount of time the ad window should be blank before the ad is displayed.
- blackAfter** The amount of time the ad window should be blank after the ad is displayed. BlackAfter runs concurrently with the blackBefore of the next ad, so that the actual time between ads is max(blackAfter,blackBefore), not blackAfter+blackBefore.
- startDT** Date/time (timezone optional) before which the ad should not run.

endDT Date/time (timezone optional) after which the ad should not run.

Ad Information:

adID A unique identifier for the ad in question. A 64-bit integer, the top 32 bits of which are a server authority id, the bottom 32 bits an identifier unique to the server authority.

title A human-friendly string used to refer to the ad.

src A URI indicating where to get the actual ad to show. This might be highly specific (eg, <http://media48.doubleclick.net/eudora/coke/drinkcoke.gif>) or it might be much more general (eg, <http://ads.doubleclick.net/eudora/ad.htm?1>). *Another important playlist feature is that we can pull ads from many different servers. We could, for example, run our own servers in parallel to DoubleClicks, and take ads from each.*

There can be a **checksum** attribute on the src tag. If present, its value is a hex-encoded MD5 digest of a secret seed and the ad data. The client may check this checksum against the ad data.

isButton Is this “ad” a toolbar button? If so, it will be scheduled separately from the main ads. The only scheduling parameters that are meaningful for toolbar buttons are startDT and endDT.

isSponsor Is this “ad” a sponsor placard? If so, it will be scheduled separately from the main ads.

isRunout Is this ad intended to be run after all other ads have exhausted their runs for a given day? There will only be one active isRunout ad in any client’s collection of playlists.

url The place to take the user if he clicks on the ad.

PLAYLISTS AND AD MODELS

It has been mentioned in passing that not all parameters are likely to be used at one time. In fact, playlists are flexible enough to support many ad models. Playlists are crucial to some ad models, to others they are helpful but not central, to still others they are marginally useful, but do not present significant impediments. The use of playlists does not predispose us toward a specific ad model; they can be used to support any model we choose. Indeed, they can even allow us to switch ad models midstream, should we decide to do so.

In the discussion that follows, we will outline several ad models, and show how playlists would be used for each model. In doing so, we will demonstrate the essential neutrality of the playlist concept to the ad model.

AD MODELS - PERSISTENT ADS

This is our current working model. In this model, the playlist provides the most functionality.

Persistent Ads	
PlayList Request	
faceTime	Used to determine how much advertising to send to client
faceTimeLeft	Not used
PlayList Response ClientInfo	
reqInterval	Relatively large; one or more days
flush	Used. Single playlist completely specifies list of ads client should have
PlayList Response Scheduling Parameters	
showForMax	Not used

One thing to notice here is how few of the parameters from any of the sections appear in the chart. That's because they are largely not relevant to the choice of model. The parameters will either be used or not, no matter what the model. For example, we can have blank space after an ad in any model, and we can eschew blank space after an ad in any model. Most of the parameters fall into this it-just-doesn't-matter category.

AD MODELS - SHORT-LIVED ADS

In this model, we accept many ads; either from many advertisers or only a few advertisers. Ads do not persist for many days; they're used up and discarded at a relatively rapid rate. Here is how this model is supported by playlists:

Short-Lived Ads	
PlayList Request	
faceTime	Not used
faceTimeLeft	Used to determine how many ads client should receive
PlayList Response ClientInfo	
reqInterval	Not used. Instead, client requests new playlist whenever ads "run low".
flush	Not used
PlayList Response Scheduling Parameters	
showForMax	Used to determine how long an ad runs

In this model, playlists will be used additively. Each time the client runs low on ads, it will ask for another playlist which will describe a few more ads to mix with the clients' existing ads. When ads use up their time, they're history. In this model, the playlist server really only serves to transmit parameters for ads. But that's ok; the parameters have to be transmitted somehow, after all.

AD MODELS - MIXED MODELS

Suppose we want to mix ad models; some long-running ads and some short-lived ads. How we handle this situation depends on the stoichiometry.

If we have mostly persistent ads and only a few short-lived ones, we merely crank up the reqInterval and use playlists as in the Persistent Ad Model. If we do this, we merely pick a few random ads to go on each playlist, and pick a few more random ads to go on the next playlist, which the client will fetch the next day.

If, on the other hand, we have only a few persistent ads and mostly short-lived ads, we will

use multiple playlists. One playlist will list the persistent ads, as above, and we'll fill the remaining facetime using playlists of short-lived ads.

CONCLUSION

The above discussion illustrates how playlists can be used to support widely differing ad models. The reason playlists can do this is that they're really only an extra level of server control and indirection between Eudora and its ads. It's hard to see how this can fail to gain us flexibility and freedom.