

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

# VoxCAST Usage and Implementation Guide

Version 1.8  
Updated 04.20.2010

## Contents

Overview.....	4
VoxCAST Purpose and Structure .....	5
Offloading Content to VoxCAST.....	7
Content Removal/Purging.....	8
Progressive Seek.....	9
Caching.....	11
Caching Overview .....	12
Caching Request Preconditions.....	13
Cache Lookups .....	13
Cache Key Generation.....	14
Cache Vary Negotiation.....	15
Conditional Cache Requests.....	15
Cache Freshness Checks.....	16
Cache Expiration Headers .....	16
Cache Expiration Logic .....	17
Serving a Cached Response.....	19
Caching a Response .....	19
Cache Storage Conditions.....	20
Stored Response Modifications .....	21
Access Control.....	23
Token-based Content Authentication.....	24
Token Authentication Settings.....	25
Token Generation .....	26
Special Considerations.....	27
Host and referrer access controls.....	27
Host access controls.....	27
Referrer access controls .....	28
Reporting .....	29
Portal Access.....	30
VoxCAST Real Time Logging.....	31
Frequency Settings .....	32
Format Settings.....	32
Log Fields .....	33
Changing settings via VoxCAST Customer Portal.....	34
Changing settings via hAPI .....	36
Retrieving Logs.....	36
Filenames.....	37
Through the portal .....	37
Through hAPI.....	37
Through other means .....	37

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

Appendix .....	37
Appendix A – Cache Status Codes .....	38
Appendix B – Cache Miss Reasons .....	38
Appendix C – HTTP Request Methods .....	39
Appendix D – VoxCAST Geographic Locations* .....	39
Works Cited .....	40

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

# Overview

## VoxCAST Purpose and Structure

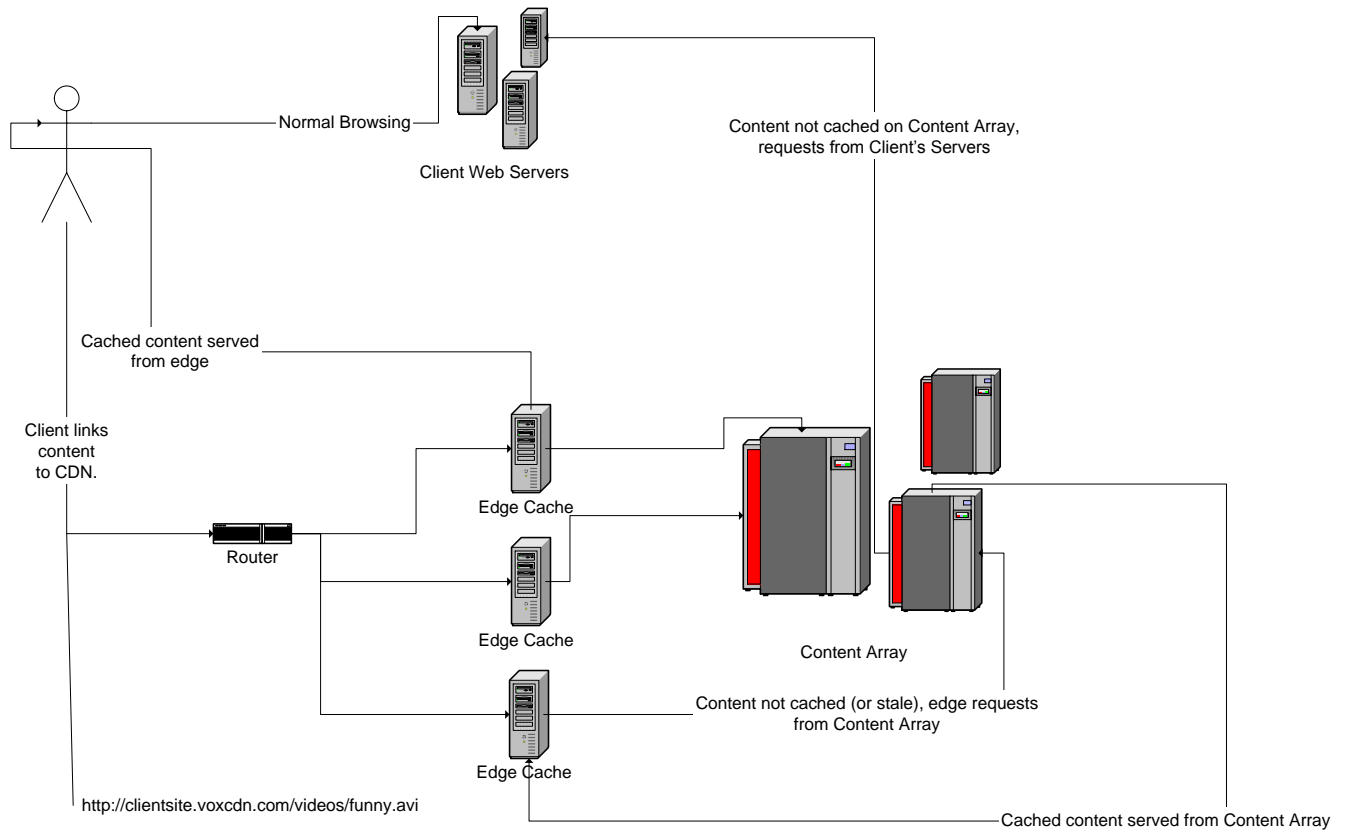
### **VoxCAST takes the pressure off.**

VoxCAST is a Content Distribution Network aimed at Internet websites that need additional bandwidth for media files or other specific auxiliary files accessed by standard web browsers. With the proper configuration, web users can request data from VoxCAST, which then requests it from the original content provider but also caches it for future requests. In this way, clients can offload the bandwidth intensive content from their sites, and focus their bandwidth and processing power on dynamic processes that are more specific to their core functionality – the things that they do best.

VoxCAST offers significant scalability, performance and resilience basis over traditional single location hosting. There are some changes that need to be done on a client's web server to make content cache-ready. There is some content that shouldn't be cached, like private user-specific information or dynamic data that changes even without the URL changing. It's hard for the CDN to differentiate between cacheable and non-cacheable content, so it's important to configure your server to increase the amount of content that can be cached, which is what helps to offload bandwidth onto the CDN.

Optionally, some of the work of configuring your server can be done for you by Voxel's `mod_cdn` ([http://www.voxel.net/labs/mod\\_cdn](http://www.voxel.net/labs/mod_cdn)), an Apache module that can rewrite URLs in your HTML content to point to VoxCAST, handle authenticating requests, and manage other CDN-related details. Even if you're using `mod_cdn`, you should read through this implementation guide to better understand how VoxCAST works, so you can use VoxCAST to your best advantage.

# VoxCDN Usage



## Offloading Content to VoxCAST

### Let VoxCAST do the dirty work.

If your web site serves any sort of large static files like images, video, sound files, documents, or downloads, you know that the actual serving of this data can take up a lot of bandwidth and processing power on your server. As a content provider, your goal should be creating or indexing the content, optimizing user experience, and configuring rules for who can or cannot access your content. The actual serving of the content is grunt work that shouldn't occupy your time.

On a typical online video website, there is an interface for browsing videos, searching them, user-specific content like recently viewed videos, etc. Within these pages, there may be embedded videos. There may be a web-based player, a browser plugin, or there may just be videos for download. In all of these cases, somewhere in the HTML code, there is a physical link to the video file, whether it be in an `EMBED` tag or just a hyperlink. After you have setup your VoxCAST account, the only thing you have to do to offload those video files to VoxCAST's Caching Network is to change those links from

<http://yourdomain.com/path/to/videofile.avi> to  
[http://\[CLIENT ID\].voxcdn.com/path/to/videofile.avi](http://[CLIENT ID].voxcdn.com/path/to/videofile.avi)

You don't have to change any server paths or disk layouts. When your account is first created, the cache for your site will be empty. As users request data through VoxCAST, it will relay those requests to your server and cache them throughout VoxCAST. The next time a user requests the same content, it will be served from the VoxCAST cache instead of being requested from your servers. VoxCAST will automatically map the request URLs to the content on your network, regardless of the directory layout of your data on your server.

## Content Removal/Purging

### Removal or purging of content from cache

Whether because of a site error, an updated version, a DMCA request, or any other reason, you may at some point want to remove specific content from our cache without waiting for it to expire. Luckily, this is very simple. Just log into the VoxCAST portal (<https://portal.voxcdn.com>) using your standard portal login and choose the 'Purge Content' link for the VoxCAST hostname in question. You will be presented with a form to enter the URL to the content you want removed. The offending content will be immediately removed from our cache, but will not be blocked for future requests.

This means that any new request for that content will force a request back to your origin web servers, which presumably would no longer offer the content or would offer some explanation for its removal. If the content you forcibly removed from cache is still available on your website, it will again be cached and you'll have to repeat the removal process.

VoxCAST supports purging by URL, by directory, or a whole-site purge.

Programmatic purging is also available as an action in hAPI (<http://api.voxel.net/docs/voxel.cdn.content.purge>).

## Progressive Seek

### Pseudo streaming of FLV files

VoxCAST OnDemand supports progressive streaming of Flash videos that allows seeking to different video offsets without needing to first download the entire video. You can also link to complete clips of cached content by sending a start and stop offset.

To make this work, your player needs to send a query parameter on seeks that conveys the byte offset within the video to seek to or stop at. This is a multi-step process that starts with encoding time offset information into your FLV videos and then completes with a player able to read and send that information to VoxCAST.

Encoding time metadata into your videos can be done with a free tool like flvtool2 (<http://www.inlet-media.de/flvtool2>).

If using flvtool2, you can just run the command: `flvtool2 -UP video.flv`

An example of a free player that supports reading time offset metadata is FLV-Scrubber ([http://www.topfstedt.de/weblog/?page\\_id=208](http://www.topfstedt.de/weblog/?page_id=208)).

The player will send the byte offset on seeks to the server as a query parameter. For FLV-Scrubber, it sends the start position in a parameter called 'start'.

This is completely configurable in the VoxCAST portal (<http://portal.voxcdn.org>), where you can set the MediaStartToken to whatever parameter your player sends the start position in. For FLV-Scrubber, you'd set MediaStartToken to 'start'.

Once the config change propagates throughout VoxCAST (about 15 minutes), then you should be able to properly seek ahead in FLV files without having to download the whole file, once the files are cached in VoxCAST.

You can read about how to prepare your FLV files, and configure your player, in the web pages that address how to do this with Apache or lighttpd. You do not need to worry about the server-configuration aspects (this is handled by VoxCAST), just the FLV metadata and the player configuration sections.

[http://www.mosalov.com/wiki/Flash\\_streaming\\_with\\_mod\\_flv](http://www.mosalov.com/wiki/Flash_streaming_with_mod_flv)

<http://blog.kovyrin.net/2006/10/08/lighttpd-memcoder-flvtool-for-streaming/lang/en/>

If you would like the file to automatically stop at some file position rather than downloading the whole video, for instance if you want to create a direct link to just a clip of a video, then you can set the MediaStopToken in the

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

VoxCAST portal to some named query parameter, and then send that parameter in your video link (or player) with a file position, and videos cached in VoxCAST will stop at that position rather than continuing to the end.

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

# Caching

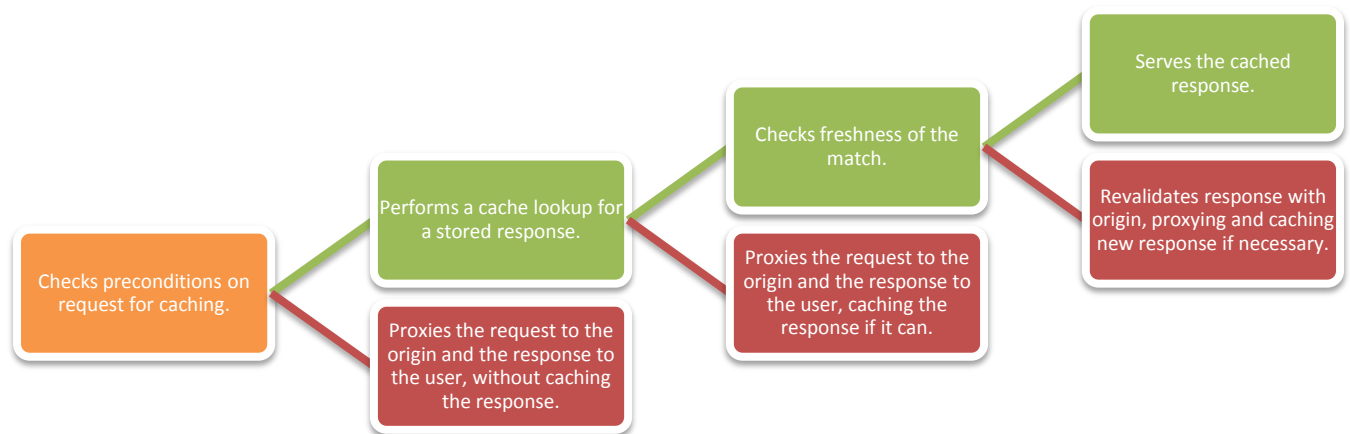
## Caching Overview

### How VoxCAST handles caching.

Caching is a complex set of decisions, both in deciding when something can be cached, and when it can be served from cache. There are many rules governing these decisions, based on HTTP caching standards (Fielding, 1999), VoxCAST modifications and settings, and request and response conditions.

VoxCAST settings and response headers can be used to guarantee a better cache-hit percentage, but you'll want to be careful that VoxCAST doesn't inadvertently cache content you don't want cached. A simple best-practice is to only use VoxCAST to deliver content you want cached. There is no scaling benefit in delivering uncacheable content through VoxCAST because VoxCAST will have to visit your server for every request.

The normal decision process uses the following diagram, with details of each step to follow.



## Caching Request Preconditions

### Tests VoxCAST performs on a request before doing a cache lookup.

There are some preconditions that decide when VoxCAST cannot and should not serve a response from cache or cache the response of the request, without having to examine the cache or the response from the origin. These are the most basic caching rules and are performed once a request is received.

They are as follows:

- The HTTP method is not 'GET'.
- The URL is longer than 8192 characters.
- The VoxCAST setting 'cache\_enable' is off.
- The request contains an 'Authorization' header and the VoxCAST setting 'cache\_ignore\_client\_auth' is off.
- The request contains a 'Cookie' header containing the value of the VoxCAST setting 'cache\_disable\_cookie\_string'.
- The VoxCAST setting 'auth\_required' is set and the request did not contain a valid token.

## Cache Lookups

### How VoxCAST performs a cache lookup.

Every item in the VoxCAST cache has a unique key that is used for storing and retrieving it. How this key is generated depends on various VoxCAST settings that allow different levels of URL-based granularity in the cache, accounting for different origin server configuration possibilities.

Once a request URL is translated into a key, additional negotiation can occur if the cached response included an HTTP 'Vary' header, which is used to tell browsers and VoxCAST that the server response will vary based on some request conditions. In those cases, VoxCAST will match the request conditions to the response Vary headers, and get the right response for the incoming request. There can also be content negotiation if the request included certain conditional HTTP headers, of the 'If-\*' family.

After request negotiation, a cached response still might not be served to a user if it is considered stale based on the response headers and VoxCAST settings. There are a series of freshness checks applied in a certain order, where some headers overrule others, allowing for complex configurations of both browser and VoxCAST caching guidelines.

## Cache Key Generation

If you have questions about why a known cached object isn't being returned on a request, it may be useful to understand the details about how cache keys are generated from an incoming request. There may be times where seemingly similar requests won't resolve to the same cache key.

The cache key is a hash of the concatenation of the hostname, the protocol scheme, and the request string.

If the VoxCAST setting 'cache\_by\_hostname' is enabled, the hostname in the cache key is the hostname used in the request, like `http://hostname/request`. Otherwise, the hostname is the 'server\_name' of the device, so that all server aliases cache to the same objects on a VoxCAST site.

If the VoxCAST per-location setting 'cache\_ignore\_scheme' is enabled, the protocol scheme used in the cache key will always be 'http'. Otherwise, it's the lowercase version of the protocol used in the request, either 'http' or 'https', thereby caching separate responses for SSL and non-SSL requests.

The request string used in cache keys is heavily affected by VoxCAST configuration options.

Here's some of the more obvious ones:

- 'cache\_rm\_consec\_slashes' – when enabled, consecutive forward slashes ( '/') in requests will be replaced with single slashes. Generally this should be enabled as the response rarely differs based on how many consecutive forward slashes are in the request.
- 'cache\_ignore\_query\_string' – when enabled, everything after the first '?' in the request string is removed when generating the cache key. This is useful for static files that might have query arguments that don't affect the response at all, like tracking or timing parameters.
- 'cache\_ignore\_token\_name' – when set, if the request query string contains a parameter of the form "token=xxx" where "token" is the value of this setting and "xxx" is anything, then "token=xxx" and everything after it in the request string are removed before generating the cache key.
- 'media\_start\_token' and 'media\_stop\_token' – when set, any parameter matching this setting has its key/value pair removed from the query string before cache key generation.
- The special parameters "vox\_timestamp" and "vox\_sig" have their key/value pairs removed.

After these preprocessing steps are finished, the query parameter key/value pairs are sorted alphabetically in the request string, so at this stage the ordering of parameters in a request is irrelevant for cache lookups.

Finally, the processed hostname, protocol scheme, and request string are concatenated together and hashed into a unique cache key.

## Cache Vary Negotiation

Once the cache key is generated, it's used to perform a cache lookup. If the object exists in cache, its response information is retrieved, including whether or not the initial response contained any 'Vary' headers. If it did, then VoxCAST stores another series of responses based on the incoming request header value for the varying header.

In that case, VoxCAST attempts to locate a response that matches the incoming request header values for the Vary fields performing another lookup with these values.

For example, if the initial response included a header like 'Vary: Accept-Encoding, Accept-Language', and this request included headers 'Accept-Encoding: gzip' and 'Accept-Language: en', then at this point VoxCAST would perform a lookup within this cache key area to see if there was a matching response for this combination of "Accept-Encoding: gzip, Accept-Language: en" values.

By default, VoxCAST will strip 'User-Agent' from response Vary directives, as the result is almost always not what was intended. Some old websites or default server configurations will instruct users to include "Vary: User-Agent, Accept-Encoding" in all responses to properly handle some possible compression issues under some scenarios in very old browsers. However, the User-Agent field not only contains the browser name, but also every combination of add-ons and modules installed. In practice, there are more than tens of millions of unique User-Agent strings in everyday use, so storing a different response for each one can effectively disable caching under most scenarios.

This behavior can be turned off for a site, if it's believed to be necessary, by contacting Voxel support.

## Conditional Cache Requests

Once an object is retrieved from cache, any conditional request directives are checked for compatibility with the response.

Conditional HTTP request headers include any of the following:

- If-Match
- If-Modified-Since
- If-None-Match
- If-Unmodified-Since

When any of these headers are included in the request, the response can either be an HTTP 412 – Precondition Failed, HTTP 304 – Not modified, or the normal response if the conditions are met.

VoxCAST's logic for reconciling these request conditions is as follows:

- If an "If-Match: xxx" is included in the request, and the cached response does not include an 'ETag' header matching the xxx, then HTTP 412 is returned.
- If an "If-Unmodified-Since: xxx" is included in the request, and the value of the 'Last-Modified' header in the response is greater than xxx or there was no 'Last-Modified' in the response, then HTTP 412 is returned.
- If an "If-None-Match: xxx" is included in the request:
  - And the request method is not GET and the cached response 'ETag' header matches xxx or xxx is "\*", then HTTP 412 is returned.
  - And the request method is GET:
    - And the cached response 'ETag' header matches xxx or xxx is "\*", then HTTP 304 is returned.
    - And the cached response 'ETag' header matches xxx, but begins with a 'W' (a weak ETag) and the request contains a 'Range' header, then conditions are met and the normal response is returned.
- If an "If-Modified-Since: xxx" is included in the request and xxx is less than the time of the request and greater than the value of the cached 'Last-Modified' header, then HTTP 304 is returned.

## Cache Freshness Checks

### How VoxCAST determines if a cached response is too stale to return.

Every cached object has an associated expiration date. There are many ways to set this on a per-URL basis, including several combinations of response headers and VoxCAST settings. Once this expiration date is reached, when a new request comes in, VoxCAST performs a revalidation process where it issues a conditional request to your site origin to look for a new response.

If the response is unchanged, VoxCAST will just extend the expiration date of the cached object and continue serving it. But if there's a new response from your origin, VoxCAST removes the object from cache and replaces it with the updated response, also serving that to the user whose request prompted the revalidation.

The method by which VoxCAST determines a cached object is expired can be a little complex, but it's worth understanding if you need to do advanced cache timing or are having trouble with stale content or maximizing your cache hits.

## Cache Expiration Headers

There are two main response headers that affect the expiration of a cached response: 'Cache-Control' and 'Expires'.

Expires is the simplest, as it provides a clear date and time to expire the response, with little other steps or knowledge needed. The easiest way to greatly increase the cacheability of your content is to always include the HTTP headers Last-Modified and Expires in responses from your origin.

Cache-Control gets more advanced, as it can have combinations of values for different settings. VoxCAST recognizes the following expiration-related values in Cache-Control response headers:

- 'max-age' – the maximum number of seconds an object is allowed to exist after being created. Age can also be sent as a response header to indicate the existing age of a response, otherwise the age starts at 0 when caching.
- 'no-cache' – the object is cached, but revalidated on every request.
- 's-maxage' – same as max-age, but it's meant for shared caches, like VoxCAST. Browsers are not supposed to honor this, so it can be used as a mechanism for setting different expiration dates for browser cache and VoxCAST cache.

There can also be Cache-Control headers in the request, which can affect whether VoxCAST determines a response is too stale. By default, VoxCAST ignores all client-provided Cache-Control headers, as honoring them would defeat the purpose of offloading traffic to VoxCAST. However, this behavior can be modified by disabling the VoxCAST site setting 'cache\_ignore\_cache\_control'. In that case, VoxCAST will recognize the following request Cache-Control values:

- 'max-age' – if the cached object's age in seconds is greater than this, it will be revalidated.
- 'max-stale' – if the cached object has been expired for less than this number of seconds, it will still be returned to the user. In this case, a 'Warning: 110 Response is stale' header will be added to the response.
- 'min-fresh' – if the cached object's expiration is less than this number of seconds away, it will be revalidated. This is for the case where the user doesn't want a response that is about to expire.
- 'no-cache' (also "Pragma: no-cache") – forces a revalidation of the cached object.

## Cache Expiration Logic

How the headers and settings interweave into a definitive fresh or not fresh determination can be a little complex to understand. Here, we'll relay the logic VoxCAST uses, which you can use as a lookup guide. This logic is applied in order, so it can be used to determine which settings and headers VoxCAST favors over others. Every time a cached response is determined to be stale, it is revalidated with the origin server, but only if the cached response contained either an 'ETag' or 'Last-Modified' header. Without one of those, conditional revalidations cannot happen and the stale object is simply removed and replaced with the new response.

- If the 'cache\_ignore\_cache\_control' setting is disabled and the request headers contain "Cache-Control: no-cache" or "Pragma: no-cache", any cached response are considered stale.
- If the response headers contain "Cache-Control: no-cache", it's always considered stale.
- If the response headers contain "Cache-Control: s-maxage=xxx":

- And the Age of the cached response is less than xxx, the object is considered fresh.
  - Except if the request header "Cache-Control: min-fresh=yyy" is present and the setting 'cache\_ignore\_cache\_control' is disabled and the Age of the cached response is not less than the result of (xxx - yyy), the object is considered stale.
- The calculation of max-age is done using the lesser of the values of the response header "Cache-Control: max-age=xxx" and the request header of the same name, if present and the setting 'cache\_ignore\_cache\_control' is disabled.
- If the Age of the cached response is less than max-age:
  - The object is considered fresh unless:
    - The request header "Cache-Control: min-fresh=yyy" is present and the setting 'cache\_ignore\_cache\_control' is disabled and the Age of the cached response is not less than the result of (max-age - yyy).
    - The request header "Cache-Control: max-stale=zzz" is present and the Age of the cached response is not less than the result of (max-age + zzz - yyy).
  - If the Age of the cached object is less than max-age + zzz, but greater than max-age, a response header is added, "Warning: 110 Response is stale".
- If there is no max-age or s-maxage, and there's a cached 'Expires' header in the response (see cache storage section for how this could be automatically added by VoxCAST), and the Age of the cached response is less than the result of ('Expires' date - current date + max-stale - min-fresh), the object is considered fresh.
  - If the cached object was only considered fresh because a present max-stale allowed it to be, a response header is added, "Warning: 110 Response is stale".
- If there was no 'Expires' header in the cached response, and no "Cache-Control: max-age" or "Cache-Control: s-maxage", and the Age of the cached response is greater than 1 day, then a response header is added, "Warning: 113 Heuristic expiration".

At this point, VoxCAST has determined whether a cached response is stale or fresh. Stale objects are not served to the user until a revalidation with the origin has resulted in either confirmation that the stale response is still valid, or a new response to replace the stale one. However, there are exceptions to this.

As part of VoxCAST's origin protection, if multiple simultaneous incoming requests are for the same stale cached object, only the first one at each location will result in a revalidation to the origin. Until the origin replies with a full set of response headers, the stale response will be served to new requests at that location. Once the response headers and first set of response bytes are received from the origin, the new response will be used without initiating new requests to the origin until it's again not deemed fresh for a request.

Also, if the setting 'preserve\_on\_revalidation\_error' is enabled and VoxCAST receives an error trying to revalidate a stale cached response, it will automatically mark the cached response as valid for another default time period. This way, if your origin is down, VoxCAST can at least continue serving whatever it has in cache, even if it's stale.

## Serving a Cached Response

Finally, an object has been found in cache and it is ready to be sent to the user. Typically, this just means the contents are read from cache, most of the original response headers are sent to the user, and the body data is sent. There are some headers that are always overridden by VoxCAST to identify itself and to show cache hits/misses. There are also some possible exceptions that could slow down serving of a cached item, which you may want to be aware of.

VoxCAST will always overwrite the following headers:

- 'Server:' will always have the value "VoxCAST".
- 'Date:' will always be the current GMT date and time.
- 'X-Cache:' will be either "HIT from VoxCAST" if the item is served from anywhere in VoxCAST's cache, or "MISS from VoxCAST" if it was not.
- 'Age:' will be the current calculated age of the cached object.
- 'Content-Length:' will be the actual length of the response.

VoxCAST tries to minimize the number of requests that reach your origin, and one of the methods for doing so is to 'piggy-back' incoming requests on in-progress responses from your origin for the same URL. This means that an incoming request may find an item in cache that is still being written, and may have to wait for your origin to finish serving it. Always including certain headers, like 'Content-Length', in your responses can allow VoxCAST to return a partial response to new requests quicker, since it won't have to calculate the size by waiting for the whole response.

Similarly, if your origin creates dynamic responses that may take longer to create, the faster VoxCAST receives the response headers, the faster it can start handling the requests and sharing the response from your origin.

## Caching a Response

The primary purpose of VoxCAST, and any CDN, is to store your content in a distributed manner and serve it to users so that your servers don't need to be capable of that level of scaling. VoxCAST achieves this by reverse proxying incoming uncached requests to your origin, caching the response, and serving that in subsequent requests for the same content.

The “Cache Lookups” section of this guide details how VoxCAST decides which URLs map to the same content in cache. Once that’s been determined and an item doesn’t exist in cache or isn’t fresh, a new response needs to be stored. Here, we’ll detail the conditions that must exist for VoxCAST to cache a response, the additional checks it performs when revalidating an already existing stale response, and some modifications VoxCAST makes to the response upon caching.

## Cache Storage Conditions

### How VoxCAST determines if content can be stored in cache.

Whether a response should be stored in cache is a decision based on factors from the origin response, your VoxCAST settings, and the request itself. The powerful flexibility can be utilized as much as your situation requires. You may never need to know how or when caching works for your site, but if you want the granularity and control, you’ll want to be familiar with these details.

The following is a list of the primary conditional checks performed before caching, along with the corresponding Cache Miss Reason Code (complete list in Appendix B) returned:

- If the response code is not one of 200, 203, 300, 301, 410 or 302 if an ‘Expires’ or “Cache-Control: max-age” header is present, then the response is not cached (code 15).
- If the response code is 302 but there’s no ‘Expires’ or “Cache-Control: max-age” response header, then the response is not cached (code 20).
- If the response included an ‘Expires’ header, but the date string was invalid, then the response is not cached (code 5).
- If the response included an ‘Expires’ header, but the date was in the past, then the response is not cached (code 6).
- If the setting ‘cache\_ignore\_query\_string’ is disabled and there is a query string in the request, but the response does not have an ‘Expires’ or “Cache-Control: max-age” header and the setting ‘cache\_store\_queries\_default\_expire’ is disabled, then the response is not cached (code 7).
- If the response code was 304 but the item was not already in cache, then the response is not cached (code 8).
- If the request method was ‘HEAD’ and the item was not already cached, then the response is not cached (code 10).
- If the response included “Cache-Control: no-store”, then the response is not cached (code 11).
- If the response included “Cache-Control: private”, then the response is not cached (code 12).
- If the request included an ‘Authorization’ header and the setting ‘cache\_ignore\_client\_auth’ is disabled and the response did not include one of “Cache-Control: s-maxage”, “Cache-Control: must-revalidate”, or “Cache-Control: public”, then the response is not cached (code 17).

- If the response included "Vary: \*", then the response is not cached (code 13).
- If the response code is 300 and the client is not using HTTP 1.1 and there's more than one choice available for the URL, then the response is not cached (code 19).
- If the setting 'cache\_max\_file\_size' is 0, then no responses are cached.
- If the value of response header 'Content-Length' is greater than the value of setting 'cache\_max\_file\_size', then the response is not cached.
- If a Content-Length header was present in the response and the fully saved response size was not equal to its value, the response is not cached.
- If the response is from a VoxCAST-initiated conditional revalidation request
  - And the response code was 304, then the response headers are cached but the existing cached response body is left unchanged.
  - And the response code was greater than 500 and the setting 'preserve\_on\_revalidation\_error' was enabled, the response is not cached.
  - And neither of the above conditions, then the cached file is removed and the new response is cached.

## Stored Response Modifications

### How VoxCAST modifies responses when caching.

When VoxCAST stores a cached response, it sometimes modifies some of the headers to affect the size and expiration time of the response. Knowing how this works can help understand how cache lookups fare when certain headers are not returned by the origin server.

The following list is when and how VoxCAST will modify responses:

- If the response did not include a 'Last-Modified' header, then VoxCAST will consider the Last-Modified value to be the current time for cache logic purposes, but will not add a Last-Modified header.
- If the response did not include an 'Expires' header, then for cache logic purposes it considers the value of 'Expires' to be the current time plus the lesser of the setting 'cache\_default\_expire' and "Cache-Control: max-age", if present. Note that according to the cache lookup freshness logic detailed earlier in this document, the 'Expires' header may not be the ultimate indication of when an object is considered stale. This mechanism merely provides a default value to fall back on if no other freshness condition applies and no 'Expires' header was present.
- If the item was not already in cache, VoxCAST removes request headers 'Range' and 'If-Range' before requesting the object from the origin. In this way, it avoids caching partial responses. The first response is always cached as the complete response, and all subsequent 'Range' requests are fully honored.

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

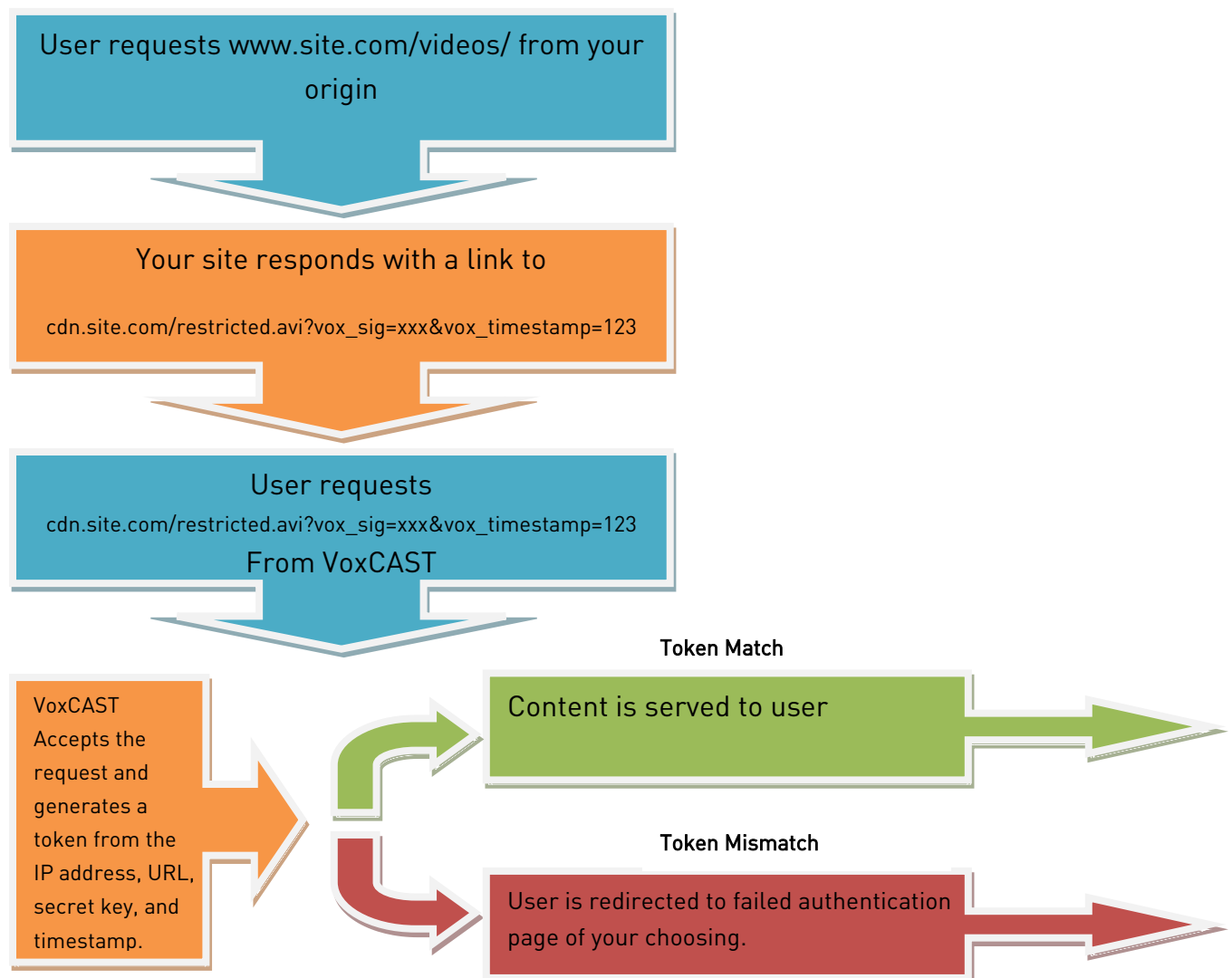
# Access Control

## Token-based Content Authentication

### Controlling access to your content.

As a content provider, sometimes you have content that you don't want to give everyone access to, but still want cached and widely available. The most likely scenario for this is when you have a subscription service and only registered, logged in users are allowed to access content. Another possibility is that you have leech detection scripts on your servers to avoid people linking directly to your content. Whatever the case, VoxCAST can work with your system to cache your content according to your authentication rules.

There is very little configuration necessary to get started, and VoxCAST can supplement whatever your existing authentication processes are. Once you have validated a user and determined they have access to content, simply tell VoxCAST that by including a special authentication token in the content URL. This token authorizes a particular IP address to access a specific URL until some future expiration time.



## Token Authentication Settings

Through the VoxCAST Customer Portal, or through the Voxel hAPI, you can configure token-based authentication on a per-URL-directory basis for your site. Options include: requiring authentication, setting the secret key, configuring the URL to redirect to for failed authentications.

Options are inherited from parent directories if not explicitly set in a sub-directory configuration.

In the VoxCAST Portal at <http://portal.voxcdn.com>, look at the Per-directory options on the configuration page:

**Per-directory options**

<i>Location</i> : The path of this Location	<input type="text" value="/"/>
<i>OriginName</i> : Origin server and path for this directory (e.g., origin.hostname.com/root/)	<input type="text" value="www.voxel.net"/>
<i>OriginMaxConnections</i> : Maximum simultaneous connections to the origin (> 0). Defaults to 2000.	<input type="text" value="2000"/>
<i>AuthenticationSiteWide</i> : Require (token-based) authentication for all requests regardless of headers.	<input type="checkbox"/>
<i>AuthenticationDefaultAlt</i> : URL for redirection upon failed authentication.	<input type="text" value="http://www.voxel.net/403/"/>
<i>AuthenticationKey</i> : Authentication key string used to compute tokens.	<input type="text" value="abc123"/>
<i>AWSS3Bucket</i> : Amazon S3 bucket name	<input type="text"/>
<i>AWSAccessKey</i> : Amazon AWS access key	<input type="text"/>
<i>AWSSecret</i> : Amazon AWS secret	<input type="text"/>

In hAPI, look at the documentation on <http://api.voxel.net/docs/> for `ondemand.locations.create/delete/read`. Calling `voxel.ondemand.locations.create` on an existing location will overwrite the settings for that location.

If you need more granularity, contact a VoxCAST integration specialist for some header-based options that may require some additional changes on your origin servers.

## Token Generation

VoxCAST token authentication consists of two special query parameters: the generated token and a timestamp for when the token expires.

When you configure your account for token-based content authentication, you'll set a secret key. This key is used in the token-generation process to guarantee the token received by VoxCAST was created by you. You can change your key in the VoxCAST portal or through the Voxel hAPI as indicated above.

The token will only be valid for a specific client IP address and URL pair, until the expiration time is reached.

The token should be a query parameter named 'vox\_sig' and the timestamp should be an ISO 8601 timestamp named 'vox\_timestamp'.

An example of an ISO 8601 timestamp is: "2009-02-20T12:10:43-0400".

The authentication token 'vox\_sig' should be a sha1 hash of:

- The IP address of the user.
- The base URL of the request, without the query arguments. This includes the protocol and everything preceding (but not including) the first '?'. For example, "http://www.example.com/example/url.php".
- All of the query string parameters (except vox\_sig, but *including* vox\_timestamp) and values, in alphabetical order by parameter name, concatenated together without separators, e.g., "key1value1key2value2...".
  - For example, "length=34&argument=seven&vox\_timestamp=2009-02-20T12:10:43-0400" would become: "argumentsevenlength34vox\_timestamp2009-02-20T12:10:43-0400".
- The secret authentication key.

## Special Considerations

Some users route their internet traffic through an HTTP proxy, which may change based on the location of a URL. In these situations, you need to be careful to use the most specific available IP by paying attention to X-Forwarded-For headers if they exist, in case the client is behind an HTTP proxy. VoxCAST will do the same in trying to verify the token.

For example, the PHP function (or similar logic in another language) found at [http://www.voxel.net/assets/voxcast\\_token\\_example](http://www.voxel.net/assets/voxcast_token_example) can be used by sending the complete url, the secret key, and the number of seconds the token should be valid for.

Then simply output the returned URL from that function in links to content like:

```
<a href="<?php echo make_signed_url('http://performancetest.voxcdn.com/medium/100_KB.dat?key=value',  
'edefbbf0ee', 900); ?>">link</a>
```

## Host and referrer access controls

In addition to the token based authentication scheme, VoxCAST allows you to restrict access to your content based on the IP address of the requester, or also based on the referring host (from the HTTP "Referer" header). Both types of access restriction may be managed in the VoxCAST customer portal, or by the `voxcast.ondemand.access_controls.*` hAPI methods. In this document we'll describe the hAPI approach.

### Host access controls

Access controls based on the client IP address (host) can be specified using either specific IPs, partial IPs or CIDR notation. You cannot add access controls based on the hostname of the client. The host access controls behave similarly to the well-known Apache Order, Allow, and Deny directives from `mod_authz_host`. By default, all hosts are allowed. To instead deny all client hosts by default for some location in your site, you can call `voxcast.ondemand.locations.update` with `allow_first=true`.

To allow or deny specific client hosts, you should call `voxcast.ondemand.access_controls.create`, setting `type=host`, `allow={true,false}` depending on whether you want the host(s) to be able to access your content, and `host` to one of:

- A complete IP address, to allow/deny a single host, e.g., 12.34.56.78

- A partial IP address, to allow/deny any hosts that match up to the end of the partial IP address, e.g., 12.34 (which is equivalent to 12.34.0.0/16 in CIDR notation)
- A netmask in CIDR notation, to allow/deny any hosts in the given range, e.g. 12.34.0.0/16.

To remove a host access control, call `voxcast.ondemand.access_controls.delete` with `type=host` and the same value of the host parameter you passed to create. You can see the currently set host access controls by calling `voxcast.ondemand.locations.list`.

## Referrer access controls

Access controls based on the referring URL are very similar to client host restrictions. Note that referrer restrictions rely on the end user's browser to set the HTTP Referrer header correctly, which may not always be the case. Commonly, content providers use referrer restrictions to prevent unauthorized linking directly to their content (e.g., images, videos). By default, all referrers are allowed. To instead deny all referrers by default for some location in your site, you can call `ondemand.locations.update` with `referrer_allow_first=true`.

To allow or deny specific referring URLs, you should call `voxcast.ondemand.access_controls.create`, setting `type=referrer`, `allow={true,false}` depending on whether you want the host(s) to be able to link to your content, and `host` to one of:

- A complete IP address (see above)
- A partial IP address (see above)
- A netmask in CIDR notation (see above)
- A hostname for the referring site, e.g., `www.leecher.com`, which prevents any URL on `www.leecher.com` from linking to your content

To remove a referrer access control, call `voxcast.ondemand.access_controls.delete` with `type=referrer` and the same value of the host parameter you passed to create. As with host access controls, you can see existing referrer access controls by calling `voxcast.ondemand.locations.list`.

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

# Reporting

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

## Portal Access

VoxCAST offers an online portal for configuring, managing and accessing statistics for your VoxCAST OnDemand and Live services. Please login at the following URL:

URL: <https://portal.voxcdn.com>

Username: Your Portal Username

Password: Your Portal Password

Once logged in, you will be presented with options pertaining to your various VoxCAST services, separated by hostname. The following options are currently available:

- View a dashboard of traffic-related statistics for your CDN services
- Configure common VoxCAST settings
- Purge content from VoxCAST's cache
- Configure access log format
- View and download access logs

Settings within the VoxCAST portal are straightforward and easy to use. If you have any questions, please contact our technical staff at [support@voxel.net](mailto:support@voxel.net).

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

## VoxCAST Real Time Logging

VoxCAST can provide full web log files for all CDN requests, just like if they were served from your own web server. Although the requests are handled on many distributed machines, the data from these requests gets consolidated into single files per hostname for ease of use.

The requests are recorded in near real time, and published into files for customer access at set intervals, as often as once a minute. The format is customer-configurable; you can choose from a selection of available fields and decide which are included, and in what order. The available fields include industry standard HTTP fields, like User-Agent and Request URL, as well as VoxCAST-specific fields like the cache status of a request, the reason a request failed caching, the location it was served from, etc.

The generated logs are accessible via the VoxCAST Customer Portal, hAPI, or optional methods such as NFS, rsync, etc.

## Log Settings

VoxCAST has a powerful, highly-configurable logging system that can be tailored to suit your individual needs. You can control how often your logs are published, what data is in them, and in what order. The following section details the available options, and how to change them. Note, all settings are on a per-customer basis, and will apply to all sites under your account.

### Frequency Settings

VoxCAST log files are made available (published) after an interval period has completed. You can configure the size of this interval in minutes. It not only determines how long you have to wait before seeing new log files, but also how much data goes into each file.

An Interval Size of 1 minute makes logs publish every minute, each only containing 1-minute of request history. Setting it to the maximum of 1 day means that 1 log file will have a whole day's worth of data, but you'll also have to wait until the end of the day to see any access logs.

Intervals are always begun and ended relative to minute 0 in a day, so 15-minute intervals begin and end on every 15-minute increment of each hour.

VoxCAST will maintain your historical logs only for up to the configured Max History setting number of days. After that, logs will not be available again, and cannot be recreated. If historical logs are important to you, it is recommended that you automate the retrieval of the logs and store them in your own system.

The following table shows the VoxCAST log frequency settings and their max/min/default values:

SETTING	MINIMUM	MAXIMUM	DEFAULT
Interval Size	1 minute	1440 minutes (1day)	15 minutes
Max History	1 day	7 days	5 days

### Format Settings

The format of the VoxCAST logs is user-configurable. You can choose the format type of the files, and what fields appear, and in what order they appear.

Format type can have the following values:

*W3C Extended Log Format (W3C) [<http://www.w3.org/TR/WD-logfile.html>]*

A format defined by the WWW Consortium, an international body that works to develop web standards. Fields are whitespace-separated (can be double quote encapsulated), with newlines separating entries. There is a header atop each file stating the date and the list and order of fields used in the file, similar to this:

```
#Version: 1.0
#Date: 12-Jan-2009 00:00:00
#Fields: time cs-method cs-uri
```

*Apache Combined Format (combined) [<http://httpd.apache.org/docs/2.2/logs.html#combined>]*

Fields are whitespace-separated (can be double quote encapsulated), entries are newline-separated. There are no headers in these files. In this format, the 'date' field contains a bracket-encapsulated ( '[]' ) date-time string, and the 'time' field is blank.

## Log Fields

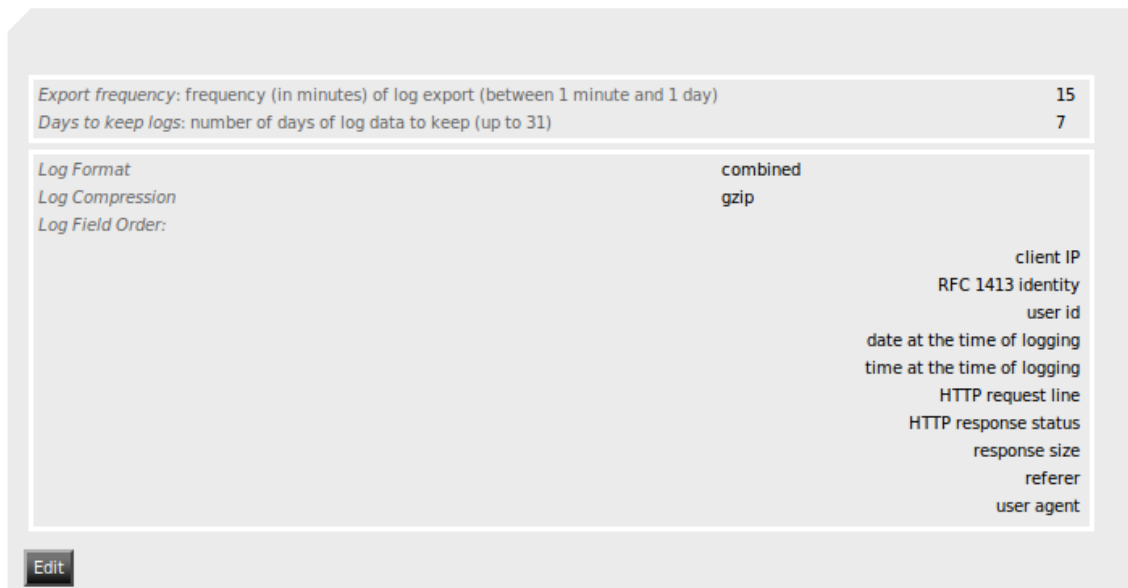
VoxCAST makes a pre-defined set of fields available; you can choose which to use and how they should be ordered.

NAME	DESCRIPTION
host	Client IP Address
ident	RFC 1413 Client Identity
userid	Client User Id
date	Date of Request (in combined format, this is date-time string)
time	Time of Request
request	HTTP Request Line
status	HTTP Response Code
bytes	Response Size in bytes
referer	HTTP Referer
user-agent	User-Agent of Client
cachestatus	VoxCAST-specific Cache Status Code (Appendix A)
cachemiss	VoxCAST-specific Cache Miss Reason Code (Appendix B)
time-taken	Time duration of the request
servername	Hostname of the site requested
method	HTTP Request Method (Appendix C)
uri	Complete Request URI ( <a href="http://xx.com?query1=value">http://xx.com?query1=value</a> )
uri-stem	Non-Query part of Request URI ( <a href="http://xx.com">http://xx.com</a> )
uri-query	Query string of Request (query1=value)

vox-query	Translated query string used by VoxCAST for caching
logdate	Date when log entry is written
logtime	Time when log entry is written
pop-location	Geographic Location of VoxCAST node content was served from (Appendix D)
request-bytes	Request Size in bytes

### Changing settings via VoxCAST Customer Portal

You can configure your log settings via the VoxCAST Customer Portal (<http://portal.voxcdn.com>) through the 'raw logs' link.



*Figure 1 Viewing log settings in VoxCAST Portal*

At first, you'll be presented with the current settings for your account. It lists your frequency settings, and the format settings, including a top to bottom list of the ordered fields appearing in your log files. To change any of these values, click the Edit button, and you'll be given a screen like below.

*Export frequency:* frequency (in minutes) of log export (between 1 minute and 1 day)

*Days to keep logs:* number of days of log data to keep (up to 31)

*Log Format* combined ▾

*Log Compression* gzip ▾

*Log Field Order:*

client IP	▾ [remove]
RFC 1413 identity	▾ [remove]
user id	▾ [remove]
date at the time of logging	▾ [remove]
time at the time of logging	▾ [remove]
HTTP request line	▾ [remove]
HTTP response status	▾ [remove]
response size	▾ [remove]
referrer	▾ [remove]
user agent	▾ [remove]

*Figure 2 Changing log settings in VoxCAST Portal*

You can change the log format, and add or remove fields as wanted. Settings will take effect after your next scheduled log publishing. For the log format, the top field appears first (left-most) in files, and then the fields continue in order as you move down.

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

## Changing settings via hAPI

You can also configure all log setting through the Voxel hosting API (hAPI). Full documentation is available online at <http://api.voxel.net/docs/>.

METHOD	DESCRIPTION
voxel.ondemand.logs.settings.list	Retrieve current frequency settings
voxel.ondemand.logs.settings.update	Change frequency settings
voxel.ondemand.logs.format.list	Retrieve current format settings
voxel.ondemand.logs.format.update	Change format settings
voxel.ondemand.logs.fields.list	Retrieve list of all available log fields

## Retrieving Logs

## Filenames

VoxCAST logs filenames include the hostname, and the UNIX timestamp start and end time of the logging interval.

```
[hostname].log.[timestamp of start]-[timestamp of stop].gz
```

## Through the portal

Just under the log settings section of the VoxCAST Customer Portal, in the 'raw logs' link, there's a list of links to log files, broken up by hostname. You can directly click on these links to download the files.

## Through hAPI

You can also retrieve log files via hAPI.

METHOD	DESCRIPTION
voxel.ondemand.logs.list	Retrieve a list of log files
voxel.ondemand.logs.get	Download a particular log file from the list

## Through other means

Under special circumstances, Voxel can also provide other means of accessing log data, through methods like rsync, NFS, etc. Please contact [support@voxel.net](mailto:support@voxel.net) for more information.

# Appendix

## Appendix A – Cache Status Codes

CODE	DESCRIPTION
0 or -	Request was not served from VoxCAST cache
1	Request was served from cache on a VoxCAST edge server
2	Request was served from VoxCAST edge cache after a revalidation
3	Request was served from cache on a middle-layer VoxCAST server

## Appendix B – Cache Miss Reasons

CODE	DESCRIPTION
0 or -	Object was not in cache
1	Invalid HTTP Method (Only GET requests are cached)
2	Internal Error
3	Request was a VoxCAST system command (Purge, etc)
4	Invalid Authentication token in request
5	Expires header was invalid
6	Content expires in the past
7	Query string was present in request, but Expires header was not in the response
8	Response was a headers-only 304 Not Modified from an upstream server and was not already in local cache. This is the result of an incoming conditional request (checking if browser-cached copy is still valid) that results in only headers being returned from the origin. VoxCAST will wait for a non-conditional request to attempt caching this object, where the origin will return both the response headers and body.
9	Response did not have any of: Last-Modified, Etag, Expires.
10	Request was a HEAD of an uncached object
11	Response contained Cache-control: no-store
12	Response contained Cache-control: private
13	Response contained Vary: *, meaning the response varies for all incoming requests. Caching would be useless as VoxCAST would still have to send every request to the origin.
14	Failed internal caching requirements.
15	Uncacheable HTTP response code
16	HTTP Authorization was present in the request and the option to ignore is not set
17	HTTP Authorization was present on a request, but response did not have one of Cache-control: (s-maxage or must-revalidate or public)
18	Request contains a cookie that matches the configured cache-ignore-cookie-string
19	Internal error

20 | Response was an HTTP 302 temporary redirect without an Expires header

## Appendix C – HTTP Request Methods

### METHOD

DELETE

GET

HEAD

POST

PUT

## Appendix D – VoxCAST Geographic Locations\*

LOCATION	STATE	COUNTRY	REGION
Amsterdam		Netherlands	Europe
Chicago	IL	United States	North America
Hong Kong		China	Asia
New York	NY	United States	North America
San Jose	CA	United States	North America
Singapore		Singapore	Asia
Washington DC	DC	United States	North America

\* Geographic locations can comprise multiple network nodes.

29 Broadway, 30th floor  
New York, NY 10006

Office  
212.812.4190

Fax  
212.812.4195

Email  
sales@voxel.net

[www.voxel.net](http://www.voxel.net)

## Works Cited

Fielding, e. a. (1999, June). *Caching in HTTP*. Retrieved from Hypertext Transfer Protocol -- HTTP/1.1:  
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html>