Screen9 Online Video Platform

API Specification, version 2.0 rev 21
Release Date: 08 July 2016

Table of Contents

1 Introduction

The Screen9 Online Video Platform API provides a complete solution to enhance a client's web site by adding advanced multimedia content. The platform supports video media and additional media types including audio and images as well as generic content (binary files). This document describes the technical details of the API for clients to use when implementing their web interface.

Content sharing involves interaction between the following three parties, so their names are defined here for clarity:

1.	Screen9	The platform provider, providing the API and hosting content.
2.	Client	The web site provider, using the API to enhance their site.
3.	User	A visitor, either to the client's content sharing website, or to an external site linking to the client's content, such as a blog.

1.1 Features Supported With Add-on Packages

Please note that this API document describes both standard features used by most clients and also some additional 'add-on' features that can be bought in addition to the standard service. Although both included and add-on features are described, access to some add-on features may depend on the package purchased for your accounts.

Below some of the available features in add-on packages are listed, please check with your Screen9 contact to confirm which are available for your accounts or if you want any more details about these.

- Adding and removing API users
- FTP upload support
- Live video or audio streaming
- Publishing time windows
- Encrypted playback streams using secure tokens
- HTTPS/SSL encrypted embed codes
- Third party Content Delivery Network (CDN) support
- Exporting to Youtube
- Content restriction
- Extended statistics
- Mixed media content management
- Cross account sharing of common content between multiple accounts
- Integration with ad servers
- Player customizations including coloring, language localization and the Screen9 Player Development SDK

1.2 System Overview

The Screen9 Online Video Platform API offers flexible methods to associate metadata to content objects. Such metadata includes both user-visible information that can be assigned by users or administrators, as well as hidden information that is used to organize content to enable intelligent retrieval. Some examples of metadata are listed below.

1. **Title**
   A text phrase identifying the content, for example "My new kitten".
2. **Description**
   A text sentence describing the content, for example "My new kitten's first steps outside".
3. **Category**
   The client may create a hierarchical tree of categories, for example there may be a category called "Animals" with a subcategory called "Pets". Typically, the client may provide the user with a selection of categories as part of the upload process.
4. **Filename**
   The uploaded filename, for example "kitten.avi".
5. **Tags**
   Tags are words or short phrases to describe content, for example "kitten" and "pet". Content may have multiple tags.
6. **Properties**
   A structure mapping arbitrary key names to text values may be assigned to content to support extended client features. For example, this could be used to maintain information about which language a video is using, by assigning a key name of 'language' with values such as 'eesti', 'português' or 'magyar', and then filtering the display of video results based on user preference. Properties are also used to control features of the Screen9 Online Video Platform API, see section 8.

Once content has been uploaded by its owner, other users can assign ratings to that media. All user ratings for each content object are aggregated to calculate an overall indication of other users' opinions.

1.3 Media types

The Screen9 Online Video Platform API supports binary, audio and image media as well as regular video objects. Binary objects are uploaded and can then be accessed by users. Screen9 will track usage but will not process these files in any way. For image objects, a thumbnail of the image is created. For audio and video files, the media is transcoded to one or several formats suitable for playback over the internet, and metadata such as duration or representative thumbnails (for videos) are extracted from the uploaded media.

2 API Overview

2.1 Client Identifier

To access the service for testing or production usage, Screen9 will assign each client an identifying number, referred to as a *custid*. This *custid* is a unique identifier and will be used not only for authentication, but also for billing purposes.

2.2 IP Addresses

For using this API, the client must register authorized IP addresses with Screen9, from which access to the API will be enabled. IP addresses can be registered in the Screen9 Account Console under Account Settings, Developer Tools.

2.3 XML-RPC

The service is available via an XML-RPC[1] server, which supports access from clients written in many programming languages, on many platforms. Please refer to the XML-RPC standard for more information about the data types used for parameters and responses, such as *array*, *bool*, *int, string*, and *struct*.

2.4 Text strings

All text strings should be passed using UTF-8 encoding.

2.5 Dates and times

All time stamps are defined according to the ISO-8601[2] standard. For example:

2009-01-01 12:34:56

For aggregated statistics, Monday is considered the first day of the week, and all periods such as days, weeks, months and years are UTC time zone.

2.6 Permaids

A *permaid* is a case-sensitive *string* type generated by Screen9, which uniquely identifies entities such as users, content objects and categories. These permaid values are intended to support permanent references to each individual entity. This means that clients can use them in URLs to link to a specific media object, category or user page, even if the details of the entity change. Such links would be hosted on the client's site, but pass the permaid value on to API XML-RPC calls. For example, the following URL demonstrates a dynamic CGI script on a client site being passed a permaid value, in this case a *mediaid*:

http://www.client.com/viewvideo.cgi?mediaid=ABCD1234

2.7 Error handling

Errors are reported to clients by exceptions through the XML-RPC interface. The exception will include a textual description and an identifying number which can be reported to Screen9 if further examination is required.

[1] http://www.xmlrpc.com
[2] http://en.wikipedia.org/wiki/ISO_8601

3 Uploading

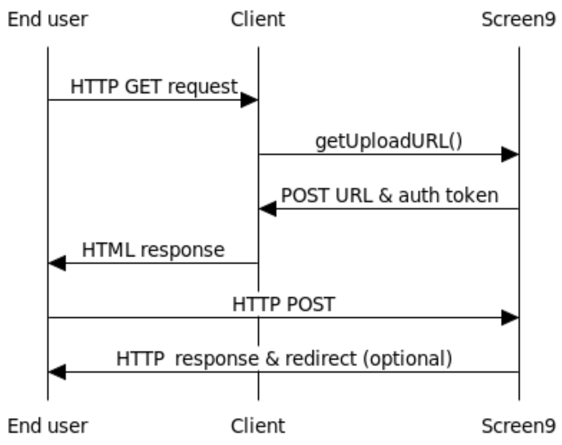
Users upload content by submitting an HTTP POST to a Screen9 web server with an authorization parameter. The authorization token is needed to verify that the upload was requested by a user visiting the client's site. Without it, a user could submit many files to the upload URL, without the client being able to impose any limit on the required bandwidth usage. The client will request the authorization parameter and an upload URL. This URL is then used on the client's web site, such that the user can upload a file using an HTTP POST submission.
The parameters below can or must be included when requesting an upload URL using the API method *getUploadURL*, and/or with the HTTP POST (see section 12.1, *Python XML-RPC client*).

HTML field name	Description	Requ-ired	Supported	
			URL request	**HTTP POST**
association	Create an association from the uploaded file to a specified *permaid*. The association parameter can be specified multiple times.	No	Yes	Yes
auth	The authorization token returned by getUploadURL.	Yes	No	Yes
categoryid	The category *permaid* for this object.	Yes	Yes	Yes
description	Maximum of 2000 characters.	No	Yes	Yes
endtime	The end time of a cuepoint in decimal seconds. Applies only to uploading of cuepoints. If endtime is left out in the POST then its value will default to the duration of the media.	No	Yes	Yes
failure_url	The URL to redirect to in case of uploading failure. A CGI parameter named 'message' includes a textual description of the cause of failure.	No	Yes	Yes
file	The file to be uploaded.	Yes	No	Yes
mediatype	One of 'audio', 'binary', 'cuepoint', 'image', 'subtitle' or 'video'.	Yes	Yes	Yes
language	Applies only to subtitles and specifies the language of the subtitle. Recommended to adhere to the ISO 639 language code standard.	Yes	Yes	Yes
property.x	A property named '*x*' is set to the field contents. See section 10.52 *SetProperty*. The parameter name is URI-decoded.	No	Yes	Yes
returnmediaid	If 'true', a parameter called 'mediaid' with the object *permaid* is appended to the success or failure URL. Example: http://client.com/v.cgi?mediaid=ABC	No	Yes	Yes
starttime	The start time of a cuepoint in decimal seconds. Applies only to cuepoints.	Yes	Yes	Yes

HTML field name	Description	Requ-ired	Supported	
			URL request	**HTTP POST**
success_url	The URL to redirect to after uploading successfully completes.	No	Yes	Yes
tag	A tag for the object. The tag parameter can be specified multiple times.	No	Yes	Yes
title	Maximum of 200 characters.	No	Yes	Yes
titleimage	A manually chosen title image file to be uploaded. Title images are restricted to the audio, cuepoint and video media type.	No	No	Yes
userid	The uploader's user *permaid*.	Yes	Yes	Yes[1]

[1]Not recommended, for security reasons.

Below is a diagram demonstrating the exchanges between all three parties during an upload from an end user using a web browser to the Screen9 Online Video Platform API via a client's web server.



1. The user requests an HTML page from the client's web server.
2. The client requests the upload URL and authorization token from Screen9 through the XML-RPC server.
3. Screen9 respond with both the URL and authorization.
4. The client responds to the user with an HTML form including the URL and authorization.
5. The user selects the file and enters appropriate metadata relating to it into the form, and then posts the file contents and metadata to a Screen9 web server.
6. Screen9 receives the file and metadata from the user, processes it, and finally optionally responds to the user with an HTML page to redirect to the client's specified success or failure page, as appropriate. If no redirect is specified Screen9 will respond with status 200 for successful uploads and status 400 if the upload didn't succeed.

In cases where the user is part of the client's organization, the *mediaid* of the newly uploaded content can be accessed in the POST response as an extension HTTP header called 'X-Picsearch-mediaid'. This *mediaid* will be empty if there was any failure during the upload, and details of the failure will be included as the message parameter to the *failure_url.* An example of the HTTP header for a successful upload is shown below:

```
X-Picsearch-mediaid: abcdef0123456789
```

For a solution with external users, the mediaid can instead be retrieved by setting the optional *returnmediaid* field (see above).

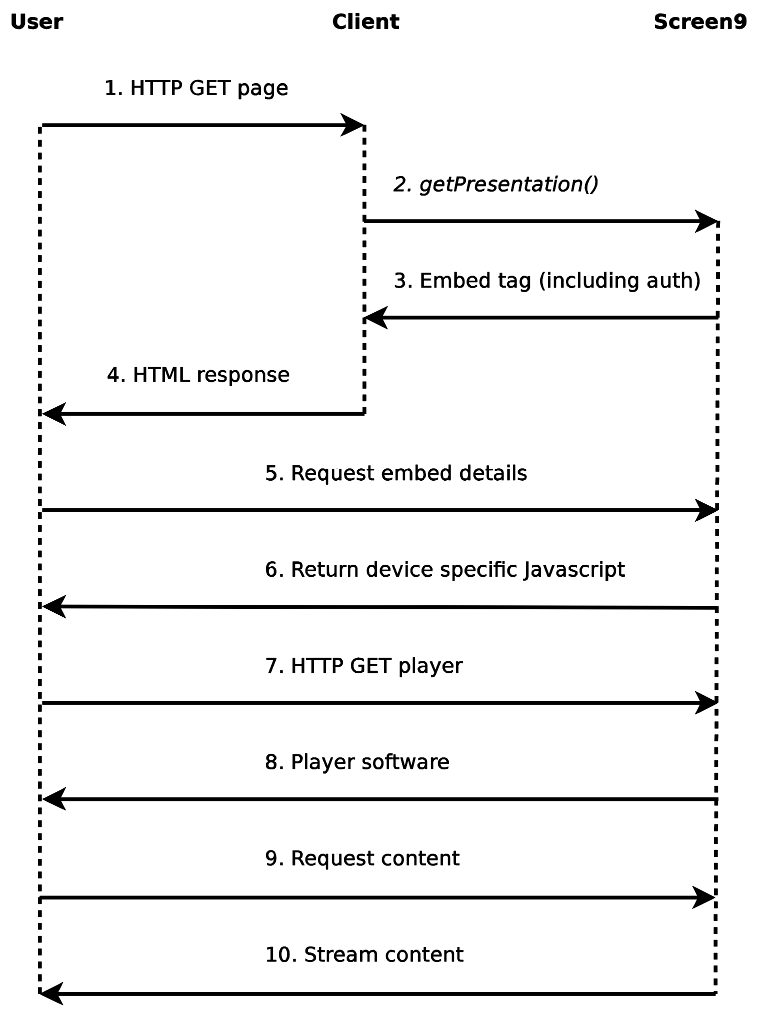
Note: In order to preserve text strings, the POST must encode them using the UTF-8 character set. For the typical case where the POST is performed from an html page, this would be achieved by including a tag such as the following in the <head> section:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

While uploading, it is possible to request the upload progress using the API method *getMediaDetails*. This requires the mediaid to be appended to the upload url in the HTTP POST using the url parameter *uploadid*. See section 12.2 for an example.

4 Playing media

Below is a diagram illustrating the flow of information when a user views a video. This example is for the recommended 'universal' embed technique which is recommended for most content links, but other similar techniques are also available. Media such as audio is presented in a similar manner.



1. The user visits a page on the client's web site, which includes embedded playback of video content.
2. The client requests the media content from Screen9 via the API, including the *mediaid* of the selected media.
3. Screen9 respond with an embed tag for the video player software, also including an authorization token to stream the video content from Screen9.
4. The client includes this embed tag in their HTML page response to the user.
5. The user's browser requests Javascript content from Screen9.
6. Screen9 perform device detection and return Javascript optimized for the user's viewing device, such as web browsers and smart phones.
7. For Flash enabled devices, the user's web browser requests the player software from Screen9. For HTML5 devices such as some smart phones, this step and the next are skipped.
8. Screen9 respond with the player software, unless this is not needed.
9. The player software running in the user's web browser requests the video content from Screen9 using the authorization token.
10. Screen9 validate the authorization and stream the video content to the user's player.

In the case of an anonymous or registered user visiting the client's site, all 10 steps are needed. However, if a client supports linking to content using *perma-links* on external sites such as blogs, then only steps 5 to 10 will be performed each time the external visitor plays the media.
In order to play media, the user either needs to have version 8.0 or higher of the Adobe Flash Player installed or be using a suitable HTML5 enabled browser.

4.1 Advertisements

It is possible to configure pre-roll and/or post-roll advertisements for a video. In this case, a video, or a random selection from a list of videos, will be played before and/or after the main video is shown. Please refer to section 8, *Properties*, for details.

5 Common Parameters

Some frequently occurring API call parameters are described here to avoid repetition.

5.1 Common

The common parameter is included for all methods as a convenient way to specify several fields common to most operations. It should be passed as a *struct*. If no user is associated with the method call in question, the browser specific fields are not required.

Parameter	Description	Type
browser	Version string submitted by the user's browser to the client's server.	*string*
custid	The client's assigned identifier.	*int*
encryption	The encryption algorithm to use. Set to 'ssl' to have the API return https urls.	*string*
refer	The referral address of the user's browser.	*string*
userip	The IP address that the user's browser is connecting from.	*string*
version	The protocol version to use, currently '2.0'.	*string*

5.2 Count

The *count* parameter specifies the maximum number of results a method should return. Count values above 100 will be capped (type *int*).

5.3 Start

The *start* parameter is used to select an offset into the result set, at which the results will start. The first result is counted as 1. For example, to construct the third page of results with 20 results per page, you would request a *count* value of 20 and a *start* value of 41. The maximum absolute value of *start* is 500. To list deeper into the result set, use the *reference* filter. When a reference is used, the start parameter can be negative, in which case results preceding the reference will be returned. In this case, *start* is the offset from the reference to the last returned entry. Using a negative start parameter while not using a reference will return items preceding the end of the list instead of the reference. For example, a start parameter of -1 will return the entries immediately preceding the reference, or end of list if a reference is omitted (type *int*).

5.4 Filters

Filters offer a method to restrict the objects returned by several calls to those that match certain criteria. The filters are passed as a *struct*. Multiple filters will further restrict results, using Boolean AND logic. Note that the *mediatype* filter is always required.

The *properties* filter is another *struct*. The member names of this *struct* should be the property key names to filter, and the values should be the values these properties must have. For example, you might pass a *struct* with a member named 'moderation' having a value of 'moderated' to the properties filter, to retrieve only moderated results. With an empty array, all objects that have a 'moderation' property will be returned. Using an array of strings instead of a single string, the filter will match any of those strings. An array of arrays each containing a single string will be matched as a substring search within the property value. Note that to match a single string as a substring, enclose it in two arrays.

The table below lists the different value types supported by the *properties* filter.

Value type	Filter mode	Example
single text string	Exact equality match	*'moderated'*
empty array	Key existence only	[]
array of strings	Boolean OR match	*['apple', 'banana']*
array of arrays	Sub-string OR match	*[['pple'], ['bana']]*

The *excludeproperties* filter is the inverse to the *properties* filter, excluding objects that have matching property keys and values. With an empty array, all media that have that property, regardless of value, will be excluded. If the value is an array of values instead of a single string, the filter will exclude all those values. As for the *properties* filter, strings enclosed in arrays will be matched as substrings.

The *tags* filter is an *array* of tags. Only media with all specified tags will match.

Member	Description	Type
association	Restricts results to media associated to this *permaid*.	*string*
categoryid	Restricts results to media within this category.	*string*
excludeproperties	Restricts results to media lacking properties in this *struct*.	*struct*
maxposted	Restricts results to media older than this date	*string*
maxproperties	Restricts results to media with properties whose values are lower than the specified values. The comparison is alphabetic and results are not guaranteed outside of the ASCII character range. Note that this means that time stamps can be used. The parameter has the same structure as the 'excludeproperties' and 'properties' parameters	*struct*
mediaids	Restricts results to media with these mediaids	*array (strings)*
mediatype	This filter is **required**. Results are restricted to media of this type. Available types include 'audio', 'binary', 'cuepoint', 'image', 'subtitle' and 'video'.	*string*
minnumratings	Restricts results to media with at least this many ratings.	*int*
minposted	Restricts results to media with this date or newer	*string*
minproperties	Restricts results to media with properties whose values are higher or equal to the specified values. See 'maxproperties'.	*struct*
properties	Restricts results to media matching properties in this *struct*.	*struct*
public	Restricts results to media that is (or isn't) public. Public is defined as having 'moderation' set to 'approved' and being inside the publish window (if it is set). See section 8. Properties: 'picsearch.moderation', 'picsearch.publishfrom' and 'picsearch.publishuntil'. Only public media is accessible using the Ajax API.	*bool*
reference	Restricts results to media before or after this *reference* media field (depending on the *start* parameter).	*string*
status	Restricts results to media with this processing status; one of 'incoming', 'processing', 'successful' or 'failed'.	*string*
userid	Restricts results to media owned by this user.	*string*
tags	Restricts results to media that have all the specified tags.	*array*

6 Media Fields

A media field is a single parameter name and value assigned to each content object. The 'fields' parameter is an *array* passed to some methods, specifying which of the fields associated with each media are to be included in the result for certain operations. The available fields are listed below, with supported content types indicated in check boxes. *Mutable* indicates that the field may be directly changed by clients via API methods. Note that the following list of parameters can be extended without prior notice. Items will however not be removed to maintain future version compatibility.

Parameter	Description	Mutable	Type	Audio	Binary	Image	Cuepoint	Subtitle	Video
audiocodec	The audio codec format of the uploaded file.	×	*string*	√	×	×	×	×	√
categoryid	The *permaid* of the category for this media.	√	*string*	√	√	√	√	√	√
categoryname	The name of the category for this media.	√	*string*	√	√	√	√	√	√
clones	All clones of this media (see 10.6 cloneMedia). Each element in the array is an array containing the custid and mediaid of each clone.	×	*array*	√	√	√	√	√	√
container	The container file format of the uploaded file.	×	*string*	√	×	×	×	×	√
description	A description of this media.	√	*string*	√	√	√	√	×	√
downloads_ started	The number of times this the viewing of this media has commenced.	×	*int*	√	√	√	×	×	√
duration	The play time for this media in milliseconds.	×	*int*	√	×	×	×	×	√
endtime	The end time of a cuepoint in decimal seconds.	√	*float*	×	×	×	√	×	×
filename	The filename of the uploaded file.	×	*string*	√	√	√	×	×	√
formats	Array of structs listing all available media formats. Each struct contains the members 'codec' (string) and 'filesize' (float). Video media may additionally contain members 'width', 'height' (integer pixels) and 'videobitrate', 'audiobitrate' (integer kbps). Audio media may additionally contain the member 'bitrate'.	×	*array*	√	√	√	×	√	√
height	The pixel height of the (transcoded) media.	×	*int*	×	×	√	×	×	√

Parameter	Description	Mutable	Type	Audio	Binary	Image	Cuepoint	Subtitle	Video
image	URL to the most representative image from this video or an uploaded image for the media. The empty string is used if the video is processing or the audio was not uploaded with an image. This field can be controlled with the property 'picsearch.titleimage', see section 8 Properties. This image is used as the title image in the media player.	×	*string*	√	×	×	√	×	√
images	Chronologically ordered image URLs from this video, or an empty array if the processing has not yet succeeded.	×	*array*	×	×	×	×	×	√
language	Contains the string representation for the subtitle's language. We recommend adhering to the ISO 639 standard.	×	*string*	×	×	×	×	√	×
live	Information about live event (requesting this field for other videos will return an empty struct). Members 'registrationdate', 'previewdate', 'startdate' and 'enddate' will correspond to registration time, preview start time, event start time and event end time, respectively; bool 'active' will indicate whether the event is currently active, and bool record will indicate whether the event will be recorded. RTMP ingestion URL list is presented in the member 'ingesturls' (the first URL in the list shall be preferred) and ingestion stream name is given in 'ingeststream'. If the event is being recorded, recording status is presented in the member 'recordingstatus' and can be one of 'recording', 'processing', 'uploading', 'stopping', 'done' and 'failed'.	×	*struct*	×	×	×	×	×	√
mediaid	The *permaid* identifying this media.	×	*string*	√	√	√	√	√	√
numratings	The number of ratings for this media.	×	*int*	√	√	√	×	×	√
originalsize	The size of the uploaded file in bytes.	×	*float*	√	√	√	×	√	√
posted	The creation time stamp for this media.	×	*string*	√	√	√	√	√	√
processed	The time stamp of when this media was (most recently) processed. This is useful for determining the outcome of re-processing (failed re-processing will not update this field).	×	*string*	√	√	√	×	√	√

Parameter	Description	Mutable	Type	Audio	Binary	Image	Cuepoint	Subtitle	Video
processing_ progress	Progress information for media which is either processing or re-processing. It will always contain a 'progress' member (float between 0.0 and 1.0). It might also contain a 'timeleft' member (integer estimated seconds) and a 'formats' member (per format progress struct similar to processing_progress).	×	*struct*	√	√	√	×	√	√
properties	The keys and values assigned to this media.	√	*struct*	√	√	√	√	√	√
rating	The average rating for this media.	×	*float*	√	√	√	×	√	√
reference	A marker that can be used in filters to make subsequent requests relative to the position of the corresponding media. It is only returned with the first and last entries.	×	*string*	√	√	√	×	×	√
starttime	The start time of a cuepoint in decimal seconds.	√	*float*	×	×	×	√	×	×
status	The status of this media; one of 'incoming', 'processing', 'successful' or 'failed'.	×	*string*	√	√	√	×	√	√
status_details	A human-readable string in English language which may contain more details about the status. It is intended for developers rather than end users.	×	*string*	√	×	×	×	√	√
status_type	A string classifying the status in more detail. For 'failed' media, the following classifications are currently supported (more may be added in the future): 'INVALID', for uploaded files that are not of the correct media type; 'UNSUPPORTED', for video files in formats not yet supported, and 'OTHER' when the error has not been automatically classified.	×	*string*	√	×	×	×	√	√
subtitles	Array of structs listing all associated and successful subtitles. Each struct contains the members: 'mediaid', 'language' and 'textdirection' (all of which are strings)	×	*array*	×	×	×	×	×	√
tags	The tags associated with this media.	√	*array*	√	√	√	√	√	√
textdirection	Either 'ltr' or 'rtl' representing left-to-right and right-to-left respectively.	×	*string*	×	×	×	×	√	×

Parameter	Description	Mutable	Type	Audio	Binary	Image	Cuepoint	Subtitle	Video
thumbnail	URL to a scaled down version of the image field for audio and video, or of the full image for images. This field can be controlled with the property 'picsearch.thumbnail', see section 8 Properties.	×	*string*	√	×	√	√	×	√
thumbnails	The same set of images available in the *images* field, in the same order, but scaled down.	×	*array*	×	×	×	×	×	√
title	The user's title for this media.	√	*string*	√	√	√	√	√	√
transcodedsize	Total storage used for stored files inbytes.	×	*float*	√	×	×	×	×	√
userid	The *permaid* of the media owner.	×	*string*	√	√	√	√	√	√
username	The user name of the media owner.	×	*string*	√	√	√	√	√	√
videocodec	The video codec format of the uploaded file	×	*string*	×	×	×	×	×	√
width	The pixel width of the (transcoded) media.	×	*int*	×	×	√	×	×	√

7 Options

Options are value strings associated with arbitrary key strings, affecting behavior for all content in an account. Customers may use options to set account wide meta data, for example “background color” = “black”.

Option keys starting with ‘picsearch.’ are reserved for controlling the behavior of the Screen9 Online Video Platform API, and should not be used by customers. Please note that the ‘picsearch’ prefix predates our Screen9 video brand and is maintained for continued compatibility.

8 Properties

Properties are value strings associated with arbitrary key strings and any object that has a *permaid*. Property keys starting with 'picsearch.' are reserved for controlling the behavior of the Screen9 Online Video Platform API, and should only be used as documented below. Customers may use properties to store custom metadata associated with media objects, for example "productid" = "3124".

Please note that the 'picsearch' prefix predates our Screen9 video brand and is maintained for continued compatibility.

Property key	Description
picsearch.cuepointtype	Cuepoints set by Account Console set this value to 'slide' for image slides and 'chapter' for chapters.
picsearch.moderation	Corresponds to the moderation gadget in the administration interface. Possible values are 'unmoderated', 'approved' and 'non-approved'.
picsearch.postroll	A list of video permaids, separated by commas (no spaces). Upon playback of the associated video, one of these post-roll videos will be selected randomly and played after the main video.
picsearch.preroll	A list of video permaids, separated by commas (no spaces). Upon playback of the associated video, one of these pre-roll videos will be selected randomly and played before the main video.
picsearch.publishfrom	Before this date show a countdown in the video player with time remaining until video is available.
picsearch.publishuntil	After this timestamp do not allow playback of this video.
picsearch.thumbnail	Used to override the 'thumbnail' media field. Set to the URL of an image thumbnail. The URL must be one of the images from the thumbnails media field or the 'picsearch.uploadthumbnail' property.
picsearch.titleimage	Used to override the 'image' media field. Set to the URL of an image. The URL must be one of the images from the 'images' media field or the 'picsearch.uploadtitleimage' property.
picsearch.uploadthumbnail	Set to the image URL of the scaled down title image uploaded with the media (if available).
picsearch.uploadtitleimage	Set to the image URL of the title image uploaded with the media (if available).
picsearch.website	Required by the *getRssFeed* API call. Must be set to the URL of a web page where the associated content is displayed.

9 Errors

Errors are reported from Screen9 to clients via XML-RPC exceptions, including a text value.

All errors are identified by an error code, consisting of three digits indicating the action, object and fault of the error. The action represents the type of action that the client was trying to perform. The object represents the entity required to perform the action. Finally, the fault represents the error that occurred when trying to perform the given action using the specified object.

The first three characters of all error texts are a three digit code representing this information in the order action, object and fault. The code details are listed below.

Actions	Object	Faults
0. unspecified	0. unspecified	0. unspecified
1. delete	1. category	1. duplicate
2. update	2. user	2. non-existent
3. rename	3. media	3. unsupported
4. lookup	4. comment (DEPRECATED)	4. not allowed
5. search	5. property	5. invalid
6. add	6. status	
7. move		

Below is a list of some error codes that may be reported, and their associated textual descriptions.

000: internal server error
114: Delete on category failed with error message: not allowed
122: Delete on user failed with error message: non-existent
132: Delete on media failed with error message: non-existent
142: Delete on comment failed with error message: non-existent
222: Update on user failed with error message: non-existent
224: Update on user failed with error message: not allowed
232: Update on media failed with error message: non-existent
233: Update on media failed with error message: unsupported
234: Update on media failed with error message: not allowed
242: Update on comment failed with error message: non-existent
311: Rename on category failed with error message: duplicate
412: Lookup on category failed with error message: non-existent
422: Lookup on user failed with error message: non-existent
432: Lookup on media failed with error message: non-existent
433: Lookup on media failed with error message: unsupported
442: Lookup on comment failed with error message: non-existent
611: Add on category failed with error message: duplicate
621: Add on user failed with error message: duplicate
631: Add on media failed with error message: duplicate

10 Available XML-RPC Methods

In the descriptions below, the following parameters (marked †) are not included, since they are so commonly used. Refer to sections 5 and 6 for more details.

Parameter	Type
common	*struct*
count	*int*
fields	*array*
filters	*array*
start	*int*
categoryid, mediaid and userid, where the entity corresponding to this *permaid* is obvious from the context	*string*

The error codes listed with each method are suggestions of possible errors that may be expected to require special treatment. However other errors may also be reported, so the lists are not guaranteed to be complete.

10.1 addCategory

Description	Adds a new category for categorizing media.
Usage	*addCategory(common†, categoryname, parentid)*
categoryname	The name of the new category to add (type *string*).
parentid	The *permaid* of the parent category, or an empty string to create a category in the root (type *string*).
Returns	The *permaid* of the new category (type *string*).
Error codes	412, 415, 611, 613

10.2 alterLiveEvent

Description	Alters attributes of a live event. Additionally, alterLiveEvent can be used to start or end a live event directly.
Usage	*alterLiveEvent(common†, mediaid†, options)*
options	A struct with options and corresponding values (type struct) controlling attributes of the live event. Use the key ‘enddate’ to specify new end time of the event. Use the key ‘endnow’ set to ‘true’ to end the event now. Use the key ‘previewdate’ to specify new preview start date of the event. Use the key ‘record’ set to one of ‘true’ or ‘false’ to specify whether the event should be recorded. Use the key ‘startdate’ to specify new start time of the event. Use the key ‘startnow’ set to ‘true’ to start the event now.
Returns	Nothing
Error codes	005, 232

10.3 assignRating

Description	Assigns a user’s rating value to media, replacing any previous rating from the same user. The call fails if users attempt to rate their own media.
Usage	*assignRating(common†, mediaid†, userid†, rating)*
rating	The rating from this user, as a number from 0 through 100 (type *int*).
Returns	The average rating for this media, including the additional vote from this call (type *float*).
Error codes	232, 235, 422, 424, 425, 604, 605

10.4 assignTag

Description	Assigns a new tag to a media.
Usage	*assignTag(common*†*, mediaid*†*, tag, userid)*
tag	The tag text to assign to the specified media (type *string*).
userid	The permaid of a user, or an empty string. If specified, the tag will only be assigned if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	232, 234, 235, 422, 424, 425, 601, 605

10.5 associateMedia

Description	Create associations between objects. This can be used to later select media that have been associated with a particular object, using the *association* filter in calls such as listMedia.
Usage	*associateMedia(common*†*, parentid, permaidlist)*
parentid	The *permaid* of the object to create associations to (type *string*).
permaidlist	An array of *permaids* to create associations from (types *array* and *string*).
Returns	Nothing.
Error codes	432, 435

10.6 cloneMedia

Description	Creates a new media instance that shares the storage of an already existing media. Metadata, such as title, description and similar will be copied to the new mediaid instance and can later be edited on its own. Any associations created for the original mediaid will not be cloned. The created clone can be used even after the original media has been deleted.
Usage	*cloneMedia(common*†*, mediaid, targetcustid, userid)*
mediaid	The media id of instance to clone (type *string*).
targetcustid	Id of the account in which to create the new media instance. The given account can be same as the original media belongs to, if not the given account must belong to the same customer. (type *string*).
userid	The *permaid* of a user, or an empty string. Specifying the empty string will skip checking user permissions, but is not allowed when cloning between accounts. If the *userid* has an associated user role, that role must permit viewing the given *mediaid*, and it must also be owned by a system login that permits uploading to the given *targetcustid*. If the *userid* does not have an associated user role, the *userid* must match the *mediaid* owner (type *string*).
Returns	The *mediaid* of the new media instance (type *string*).
Error codes	422, 424, 432, 433, 434, 435, 634, 635

10.7 confirmUser

Description	Verifies whether a user name has been registered, and if so, retrieves the user's *permaid*.
Usage	*confirmUser(common*†*, username)*
username	The username to confirm (type *string*).
Returns	The *permaid* of a registered user, or an empty string (type *string*).
Error codes	425

10.8 countMedia

Description	Counts the total number of media that match the specified filters.
Usage	*countMedia(common*†*, filters*†*)*
Returns	The count value (type *int*).
Error codes	405, 455

10.9 countMediaTags

Description	Counts the total number of tags for a media.
Usage	*countMediaTags(common†, mediaid†)*
Returns	The number of tags (type *int*).
Error codes	432, 435

10.10 countUsers

Description	Counts the number of registered users.
Usage	*countUsers(common†)*
Returns	The number of users (type *int*).

10.11 createCuepoint

Description	Create a cuepoint object. Cuepoints are timeline references to existing media (parentid) and can only be created for media of type audio and video. The created cuepoint will appear as an association to the parent media object.
Usage	*createCuepoint(common†, title, description, categoryid, userid, parentid, starttime, endtime)*
title	A title to describe the created cuepoint (type *string*).
description	A textual description of the cuepoint object (type *string*).
categoryid	Id of the category to which the cuepoint belongs (type *string*).
userid	Id of the user who owns the cuepoint (type *string*).
parentid	Media id of the media (video or audio) to which the cuepoint belongs
starttime	The start time in seconds on the timeline of the associated media (type *float*)
endtime	The end time in seconds on the timeline of the associated media (type *float*). If endtime is given a value less than zero then its value will default to the duration of the media.
Returns	The mediaid of the created cuepoint (type *string*)
Error codes	235, 422, 424, 425, 715

10.12 disassociateMedia

Description	Remove associations between objects.
Usage	*disassociateMedia(common†, parentid, permaidlist)*
parentid	The *permaid* of the object to remove associations to (type *string*).
permaidlist	An array of *permaids* to remove associations from (types *array* and *string*).
Returns	Nothing.
Error codes	432, 435

10.13 eventStatsByPeriod

Description	Retrieves the latest statistics for the most frequent video events. Note that this is not available for non-video media.
Usage	*eventStatsByPeriod(common†, period, event, count†, start†)*
period	One of 'day', 'week', 'month' or 'year' (type *string*).
event	The event type to report, only 'download_start' is recommended which includes partially viewed media in the case of videos (type string).
Returns	An *array* of event frequency statistics, sorted by frequency. Each element in the *array* is a *struct* with two members:
mediaid	The *permaid* of the video (type *string*).
numevents	The number of specified events that occurred for this media during the specified time period (type *int*).
Error codes	405

10.14 findCategory

Description	Retrieves the *categoryid* of a category given its category hierarchy.
Usage	*findCategory(common*†*, hierarchy)*
hierarchy	An *array* of *strings*, containing names of any parent categories, and finally the name of the category itself (types *array* and *string*).
Returns	The *categoryid* of the category (type *string*).
Error codes	512, 515

10.15 getAjaxAuth

Description	Request authorization token to enable use of the Screen9 Ajax library (see separate Screen9 Ajax API documentation for details).
Usage	*getAjaxAuth(common, userip, security, timeout)*
userip	Restrict access to this user IP (recommended). By submitting an empty string, global access is enabled (type string).
security	One of *'readonly'* or *'high'* (type *string*). The *'readonly'* level will not allow any calls that modify content. The *'high'* level allows modifying calls only if they cannot affect the functionality, such as allowing ratings to be set. Less restrictive security levels may be added in the future.
timeout	Enable access for this many seconds, or zero for unlimited access (type *int*).
Returns	An encrypted authorization token (type *string*)
Error codes	405

10.16 getBandwidthStats

Description	Retrieves detailed bandwidth usage stats, for video media only.
Usage	*getBandwidthStats(common*†*, mediaid, event, trackingid, location, startdate, enddate, period)*
mediaid	The *permaid* of a specific media to get stats for, or an empty string to get stats concatenated across all media (type *string*).
event	The event type, either 'display' to get the number of displayed players or 'download_start' for all started streams (type string).
trackingid	Restrict result to usage with this trackingid, or -1 to concatenate across all trackingids (type *int*).
location	Restrict results to usage from this country (not available for 'display' event), or empty string to concatenate across all countries. Countries should be expressed as two-letter strings in compliance with the ISO 3166-1-alpha-2 standard[1] (type *string*).
startdate	Restrict result to periods starting at this date or later (type *string*).
enddate	Restrict result to periods before or containing this date (type *string*).
period	The length of the period statistics are grouped by, chosen from 'hour', 'day', 'week', 'month' or 'year'. Note that 'hour' is only available if the *mediaid* parameter is empty, and that requesting a time range containing more than 400 periods will result in an error (type *string*).
Returns	An *array* with one *struct* for each period in the selected range. The *structs* have the following members:
startdate	The starting date for the period (type *string*).
impressions	The number of specified events within the period (type *int*).
bandwidth	The bandwidth usage in bytes within the period (type *double*).
Error codes	404, 405

[1] http://www.iso.org/iso/country_codes.htm

10.17 getImpressionStats

Description	Retrieves average usage statistics for hours in the day, collected across a specified time period.
Usage	*getImpressionStats(common*[†]*, event, trackingid, location, startdate, enddate)*
event	See *getBandwidthStats.*
trackingid	See *getBandwidthStats.*
location	See *getBandwidthStats.*
startdate	See *getBandwidthStats.*
enddate	See *getBandwidthStats.*
Returns	An *array* containing 24 *doubles*. The first *double* is the percentage of specified events occurring between 00:00 and 01:00 UTC, and so on for the following hours.
Error codes	404, 405

10.18 getMediaContentURL

Description	Generates a URL to access media content. This call returns a direct link to a media file.
Usage	*getMediaContentURL(common*[†]*, mediaid*[†]*, period, trackingid, options)*
period	The number of seconds that this link should remain valid. If a value of zero is supplied, the link will always be valid and never expire (type *int*).
trackingid	A positive integer used to track where the content is being shown. This may be different websites, or different pages on the same website, or a combination. Instances where no special tracking is required, such as on the main client site, should use zero. Allocation of other tracking ids is left at the client's discretion. The maximum allowed id is 100,000,000 (type *int*).
options	A *struct* with options and corresponding values (type struct). Use the key 'filename' to set the filename in the returned header to the associated value (type string). Note that this refers to the object filename and not the returned URL. Use the key 'protocol' set to values 'hls' or 'rtmp' to return HTTP Live streaming or RTMP Flash streaming links, respectively, or else HTTP progressive download links will be returned by default. Use the key 'userip' to bind the stream to only the specified end user IP address (using the format '1.2.3.4'), restricting access from unauthorized addresses.
Returns	A URL to the media file (type *string*).
Error codes	005, 432, 435

10.19 getMediaDetails

Description	Lists fields, with values, for a media object. Refer to section 6 for details.
Usage	*getMediaDetails(common*[†]*, mediaid*[†]*, fields)*
fields	See section 6, Fields. If an empty *array* is submitted, all available fields will be returned. An optional field member not stated in section 6 is 'upload_progress' that will return the progress of the upload defined by *mediaid* as a value between 0.0 when started to 1.0 when completed (type *float*), so 0.5 indicates 50% progress. When given, no other field members are allowed at the same time. Refer to section 3 for more information on uploads. (type *array*).
Returns	A *struct* with a member for each requested field (type *struct*).
Error codes	432, 435

10.20 getMediaDownloadURL

Description	Generates a URL to download media in the highest quality available. The returned URL is meant for downloading and it is not suitable for other purposes such as streaming (use getMediaContentURL for streaming instead).
Usage	*getMediaDownloadURL(common†, mediaid†, period, trackingid, options)*
period	The number of seconds that this link should remain valid. If a value of zero is supplied, the link will always be valid and never expire (type int).
trackingid	A positive integer used to track how the content is being downloaded. Instances where no special tracking is required, such as on the main client site, should use zero. Allocation of other tracking ids is left at the client's discretion. The maximum allowed id is 100,000,000 (type int).
options	A struct with options and corresponding values (type struct). Use the key 'filename' to set the filename in the returned header to the associated value. Note that this refers to the object filename and not the returned URL. Set the key 'source' to 'true' to prefer the original uploaded file if such is stored for the media in question. Use the key 'userip' to bind downloads to only the specified end user IP address (using the format '1.2.3.4'), restricting access from unauthorized addresses.
Returns	A downloadable URL to the media file (type string).
Error codes	005, 432, 433, 435

10.21 getMediaStreams

Description	Returns URIs to video streams in all available formats for use by video players in mobile cell phones. Note that mobile support is not enabled by default, please contact your Screen9 sales representative to add support.
Usage	*getMediaStreams(common†, mediaid†, period, trackingid, options)*
period	Number of seconds for which the returned streams will be valid, or 0 for an embed code that never expires (type *int*).
trackingid	Trackingid that will be associated with the generated embed code (type int).
options	A *struct* with options and corresponding values (type *struct*). Use the key 'userip' to bind the stream to only the specified end user IP address (using the format '1.2.3.4'), restricting access from unauthorized addresses. Please note that IP restriction works for HTTP and RTMP streams used most frequently, but cannot be enforced for RTSP links used by legacy mobile devices.
Returns	A *struct* with a key and value pair for each available format. The key is an identifier for the format used to encode this stream (type *string*). The available list of supported format names and precise codec and bitrate details is supplied separately by a Screen9 technical contact. The value is a URI to access the video stream encoded in the format associated with the key name (type *string*).
Error codes	005

10.22 getOption

Description	Gets the value of a client-adjustable option. Refer to section 7 for details.
Usage	*getOption(common†, key)*
key	The name of the option to retrieve (type *string*).
Returns	The value of the option (type *string*).
Error codes	402, 405

10.23 getPresentation

Description	Used to present media in different formats.
Usage	*getPresentation(common*[†]*, mediaid, period, options)*
period	Number of seconds for which the returned embed code will be valid, or 0 for an embed code that never expires (type *int*).
options	A *struct* containing options. See table below for supported members.
Returns	A *struct*, with keys and values depending on the *embedtype* option (see tables below).
Error codes	005

Option	Description	Type
autoplay	Set to '1' to auto-start playback of video content once the page loads, without waiting for user interaction.	*string*
containerid	The name of the HTML container which the player will be displayed inside. By default, the *mediaid* of the video will be used. Use this parameter if you already have such an id in your HTML page, or if you want to display several players with the same video on one page.	*string*
embedtype	The type of embed code that will be returned. Supported embed types include *component_links, playertag*, *playertag-rdfa* and *universal*. Unless there are specific requirements, *universal* is generally recommended. See the table below for details.	*string*
playerheight	Player height in the generated embed code.	*int*
playerwidth	Player width in the generated embed code. Set to the string value "100%" (*playerheight* should be set to -1) to generate a responsive embed code.	*int or string*
trackingid	Trackingid that will be associated with the generated embed code.	*int*
showembedcode	Set this to 'playertag' to make the player offer an embed code to the user, or 'none' to disable.	*string*
wmode	Sets the video rendering method in the Adobe Flash environment. The default 'opaque' is suitable for most cases, or 'transparent' may be necessary when using custom players using rounded edges. The final 'window' setting is not recommended.	*string*
userip	Binds the stream to only the specified end user IP address (using the format '1.2.3.4'), restricting access from unauthorized addresses.	*string*

embedtype	Description
component_links	Returns a struct with keys image, player and playerconfig
playertag	An HTML player tag that can be copied and pasted into most types of CMS editors and blogs, without requiring special posting privileges. The returned struct includes a key 'playertag' with the embed code as its string value.
playertag-rdfa	The same as playertag, but with additional RDFa mark up tags used to enhance search engine inclusion. The returned struct includes a key 'playertag-rdfa' with the embed code as its string value.
universal	A Javascript-based embed tag that automatically detects the user's web browsing device and shows the video in a compatible way. Note that support for the Apple iPad and mobile formats is not included by default, please contact Screen9 for more information

10.24 getProperty

Description	Gets the value of a property. Refer to section 8 for details.
Usage	*getProperty(common†, permaid†, key)*
Key	The name of the property to retrieve (type *string*).
Returns	The value of the property (type *string*).
Error codes	432, 435, 452, 455

10.25 getRating

Description	Returns the rating given by a particular user, for a particular media. If the user has not rated this media, the call will return with an error 402.
Usage	*getRating(common†, mediaid†, userid†)*
Returns	Rating, as a number from 0 through 100 (type *int*).
Error codes	402, 422, 424, 425, 432, 435

10.26 getReplacementUploadURL

Description	Returns the URL and authorization code used to receive replacement media uploads via HTTP POST. See uploading in section 3, *Uploading*, for details. The upload will replace an existing media object.
Usage	*getUploadURL(common†, mediaid, period, staticfields, mutablefields)*
mediaid	The existing media object to replace with the upload (type string).
period	The number of seconds that this upload URL will be valid. After this period, the URL may no longer be used to upload content (type *int*).
staticfields	A struct containing field names and values that will be set for the uploaded media. See section 3, Uploading for required and optional HTTP POST fields (type struct). For fields accepting multiple values, the struct value may be set to a list of strings and each string in the list will be set for the field name.
mutablefields	An array of HTTP POST field names that will be submitted together with the uploaded media. See section 3, *Uploading* for required and optional HTTP POST fields (type *array*).
Returns	A *struct* with the following members:
auth	The authorization string to be passed as a POST field (type *string*).
url	The URL to use for the HTTP POST upload (type *string*). This URL may change, so its value should not be cached.
Error codes	005, 234, 432

10.27 getRssFeed

Description	Returns a MediaRSS[1] feed. Only media with a picsearch.website property are included. Publishing MediaRSS feeds allows users to subscribe to your media, and can also increase the chances that web search engines will accurately match the included media with relevant search queries. Up to ten tags for each media are included as keywords in the feed. In addition, setting the title and description of each media to something descriptive will greatly help the ability of search engines to match these media in their results, and to present them to the user in an attractive fashion.
Usage	*getRssFeed(common†, order, count†, ttl, title, description, filters†, start†, options)*
order	The preferred sorting, one of 'downloads', 'posted', 'rating', 'title' or 'transcodedsize' (only for media of type audio or video) (type *string*).
ttl	Time-to-live, the number of minutes after the call is made that the feed will remain valid, or 0 for no time limit (type *int*).
title	A title for the feed, maximum 200 characters (type *string*).
description	A description for the feed, maximum 2000 characters (type *string*).
options	A *struct* with permitting to override defaults set in the account options starting with *picsearch.syndication*, see section 7. Supported members are *link*, *image.url*, *image.title* and *image.link* (types *struct* and *string*)
Returns	A MediaRSS feed (type *string*).
Error codes	005

10.28 getUploadURL

Description	Returns the URL and authorization code used to receive new media uploads via HTTP POST. See uploading in section 3, *Uploading*, for details.
Usage	*getUploadURL(common†, period, staticfields, mutablefields)*
period	The number of seconds that this upload URL will be valid. After this period, the URL may no longer be used to upload new content (type *int*).
staticfields	A struct containing field names and values that will be set for the uploaded media. See section 3, Uploading for required and optional HTTP POST fields (type struct). For fields accepting multiple values, the struct value may be set to a list of strings and each string in the list will be set for the field name.
mutablefields	An array of HTTP POST field names that will be submitted together with the uploaded media. See section 3, *Uploading* for required and optional HTTP POST fields (type *array*).
Returns	A *struct* with the following members:
auth	The authorization string to be passed as a POST field (type *string*).
url	The URL to use for the HTTP POST upload (type *string*). This URL may change, so its value should not be cached.
mediaid	The mediaid that the uploaded media will receive. To be used in the HTTP POST upload as well as when requesting upload progress.
Error codes	005

[1]For more information about Media RSS feeds, refer to http://en.wikipedia.org/wiki/Media_RSS

10.29 getUserLoyalty

Description	Reports distribution of users according to how many different days in a specified month they were accessing at least one media object. Users are distinguished by IP address, so for networks where many users identify with the same IP address, this will be inaccurate.
Usage	*getUserLoyalty(common*†*, trackingid, location, date)*
trackingid	See *getBandwidthStats.*
location	See *getBandwidthStats.*
Date	A date at the start of or within the requested month (type *string*).
Returns	An *array* of *doubles*. The first *double* is the percentage of users who streamed on only one day in the month; the second is the percentage who streamed during two different days, and so on. Consequently, the last number is the percentage of users who streamed at least one object every day of the month.
Error codes	405

10.30 getUserStatus

Description	Returns the status of a user, and the time the status was last changed.
Usage	*getUserStatus(common*†*, userid*†*)*
Returns	A *struct* with the following members:
created	A time stamp for when the user was created (type *string*).
lastupdate	A time stamp for when the user status was last set (type *string*).
Status	One of ‘active’ or ‘inactive’ (type *string*). Users set to ‘inactive’ are prohibited from modifying any content.
username	The user name (type *string*).
Error codes	422, 425

10.31 listCategories

Description	Lists categories in a given parent category.
Usage	*listCategories(common*†*, filters, count*†*, start*†*)*
filters	A struct to limit the returned categories. Category filters are analogous to media filters, please refer to section 5.4. Supported filters are listed in the table below.
Returns	A *struct* for each category (types *array* and *struct*). The *struct* members are:
categoryid	The category *permaid* (type *string*).
categoryname	The category name (type *string*).
Error codes	405, 412, 415

Option	Description	Type
association	Restricts results to categories associated with this *permaid.*	*string*
excludeproperties	Restricts results to categories lacking properties in this *struct.*	*struct, string*
maxproperties	Restricts results to categories whose values are lower than the specified values.	*struct, string*
minproperties	Restricts results to categories whose values are higher or equal than the specified values.	*struct, string*
parentid	Restricts results to categories within a parent category.	*struct, string*
properties	Restricts results to categories with properties matching this *struct.*	*struct, string*

10.32 listCategoryCounts

Description	Lists the sub-categories and their number of media, for a given parent category. Note that only media matching the filters will be counted.
Usage	*listCategoryCounts(common†,parentid, filters†, count†, start†)*
parentid	A parent categoryid, or an empty string to list root categories (type *string*).
Returns	A *struct* per category (types *array* and *struct*). The *struct* members are:
categoryid	The category *permaid* (type *string*).
categoryname	The category name (type *string*).
count	The number of media in the category (type *int*).
Error codes	405, 415

10.33 listMedia

Description	Lists media objects within selected filter rules. Make sure to use *reference* filters when listing deep into the result set.
Usage	*listMedia(common†, fields†, filters†, order, count†, start†)*
order	The preferred sorting, one of 'downloads', 'posted', 'rating', 'title' or 'transcodedsize' (only for media of type audio or video) (type *string*).
Returns	A *struct* for each item, with the requested fields (types *array* and *struct*).
Error codes	405, 435

10.34 listProperties

Description	Lists all property names set for any objects.
Usage	*listProperties(common†, count†, start†)*
Returns	An *array* of *strings*.
Error codes	405

10.35 listTags

Description	Lists the most frequently used tags that have been set for any media. The 1000 most used tags are available with this call.
Usage	*listTags(common†, count, start)*
Count	Number of tags to return (maximum 100).
Start	Offset to return less frequently used tags (max 1000).
Returns	An array of tags (types *array, string*).
Error codes	405

10.36 listUsers

Description	Lists registered users.
Usage	*listUsers(common†, count†, start†)*
Returns	A *struct* for each user (types *array* and *struct*). The *struct* members are:
userid	The *permaid* of the user (type *string*).
username	The name of the user (type *string*).
Error codes	405

10.37 moveCategory

Description	Moves a category to another parent category.
Usage	*moveCategory(common†, categoryid†, parentid)*
parentid	The *permaid* for the destination category, or an empty string to move the category to the root (type *string*).
Returns	Nothing.
Error codes	710, 711, 712, 713, 714, 715

10.38 moveMedia

Description	Moves a media object to another category.
Usage	*moveMedia(common*†, *mediaid*†, *categoryid*†, *userid*)
userid	The permaid of a user, or an empty string. If specified, the media will only be moved if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	232, 235, 422, 424, 425, 712, 715, 732, 734, 735

10.39 registerUser

Description	Registers a user with the service.
Usage	*registerUser(common*†*, username)*
username	The name of the new user to register (type *string*).
Returns	The *permaid* of the new user (type *string*).
Error codes	621, 623

10.40 registerLiveEvent

Description	Registers a new live event in the system.
Usage	*registerLiveEvent(common*†*, categoryid, userid, title, description, startdate, options)*
categoryid	Id of the category to which the event belongs (type string).
userid	Id of the user who owns the event (type *string*).
title	A title to describe the created event (type *string*)
description	A textual description of the event media (type *string*)
startdate	Start time of the event
options	A *struct* with options and corresponding values (type *struct*). Use the mandatory key ‘enddate’ to specify end time of the event. Use the key ‘channel’ to request a specific live channel. Use the key ‘country’ to override automatic country detection from IP provided in the common parameter. Values shall be provided as a lowercase two-letter ISO 639-1 country code. Ingest server location in the cloud will be optimized for the provided country. Use the key ‘image’ set to value ‘default’ to account default title image. Use the key ‘moderation’ set to one of ‘unmoderated’, ‘approved’ and ‘non-approved’ to control moderation of the event. Corresponds to the moderation gadget in administration interface. Use the key ‘record’ set to value ‘1’ to record the event
Returns	The mediaid of the newly created video media (type *string*).
Error codes	005, 412, 422, 424

10.41 removeCategory

Description	Removes a previously added category. This is only supported if the category has no media or categories in it, otherwise the removal will fail.
Usage	*removeCategory(common*†*, categoryid*†*)*
Returns	Nothing.
Error codes	112, 114, 115

10.42 removeMedia

Description	Removes a previously created media, and any references to it. This function can remove objects of any type, for example: audio, binary cuepoint, image and video.
Usage	*removeMedia(common†, mediaid†, userid)*
userid	The permaid of a user, or an empty string. If specified, the media will only be removed if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	132, 134, 135, 422, 424, 425

10.43 removeOption

Description	Removes a previously set option.
Usage	*removeOption(common†, key)*
Key	The option key name (type *string*).
Returns	Nothing.
Error codes	102, 105, 235

10.44 removeProperty

Description	Removes a property from an object.
Usage	*removeProperty(common†, permaid†, key, userid)*
Key	The key name of the property to be removed (type *string*).
Userid	The permaid of a user, or an empty string. If specified, the *permaid* must be a *mediaid*, and the property will only be removed if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	152, 154, 155, 232, 235

10.45 removeRating

Description	Removes a user’s rating for a media object.
Usage	*removeRating(common†, mediaid†, userid†)*
Returns	Nothing.
Error codes	102, 232, 235, 422, 424, 425

10.46 removeTag

Description	Removes a previously assigned tag from a media object.
Usage	*removeTag(common†, mediaid†, tag, userid)*
tag	The text of the tag to be removed (type *string*).
userid	The permaid of a user, or an empty string. If specified, the tag will only be removed if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	102, 105, 232, 235, 422, 424, 425

10.47 removeUser

Description	Removes a user. Not recommended, consider using *setUserStatus* instead. All objects belonging to the user must be removed prior to this call.
Usage	*removeUser(common†, userid†)*
Returns	Nothing.
Error codes	124, 422, 425

10.48 renameCategory

Description	Renames a category.
Usage	*removeRating(common†, categoryid†, categoryname)*
categoryname	The new name for the category (type *string*).
Returns	Nothing.
Error codes	311, 312, 313, 315

10.49 search

Description	Searches selected metadata, matching arbitrary text.
Usage	*search(common†, text, constraints, fields†, filters†, order, count†, start†)*
text	The text phrase to search for (type *string*).
constraints	An *array* of fields to search within, chosen from 'title', 'description' and 'tags' (type *array*).
order	The preferred sorting, one of 'downloads', 'posted', 'rating', 'title' or 'transcodedsize' (only for media of type audio or video) (type *string*).
Returns	A *struct* with the following members:
results	An *array* containing *structs* with the requested fields for each item.
count	The total count of media that match this search (type *int*).
Error codes	405, 455, 505

10.50 setMediaDetails

Description	Sets a media field to a new value.
Usage	*setMediaDetails(common†, mediaid†, field, value, userid)*
field	The field to set, chosen from 'title' or 'description' (type *string*).
value	The new text value to assign (type *string*).
userid	The permaid of a user, or an empty string. If specified, the media will only be changed if *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	232, 234, 235, 422, 424, 425

10.51 setOption

Description	Sets an option.
Usage	*setOption(common†, key, value)*
key	The name of the option. Refer to section 7 for details (type *string*).
value	The content of the option (type *string*).
Returns	Nothing.
Error codes	205

10.52 setProperty

Description	Sets a property value for an object. Existing values will be replaced.
Usage	*setProperty(common†, permaid†, key, value, userid)*
key	The name of the property key, e.g. 'moderation' (type *string*). Keys are limited to 2000 characters in size. Key names starting with 'picsearch.' are reserved for controlling features provided by Screen9.
value	The value of the property, e.g. 'offensive' (type *string*). Values are limited to 2000 characters in size.
userid	The permaid of a user, or an empty string. It can be given when setting a media property, which will then only be set if the *userid* matches the media owner (type *string*).
Returns	Nothing.
Error codes	232, 235, 254, 255

10.53 setUserStatus

Description	Sets the status for a user. The user status is intended to be used to track whether a user breaks the policy of a client site, for example by uploading copyrighted or offensive video content. Additionally, setUserStatus can be used to update the user's *lastupdate* time stamp, which is retrieved with getUserStatus.
Usage	*setUserStatus(common*†*, userid*†*, status)*
status	The new status value, chosen from 'active' or 'inactive' (type *string*).
Returns	Nothing.
Error codes	222, 225, 265

11 Performance considerations

This section deals with performance considerations, and provides pointers for how to minimize end-user latency when rendering a web page using XML-RPC API methods.

11.1 Minimizing the number of round-trips

The most important consideration is how to minimize the number of round-trips between the client web server and the Screen9 XML-RPC server. A web server making one XML-RPC call will stall until the response is returned, and if the server does many such calls, the end user experience will suffer. There are essentially two ways of reducing the number of such round-trips without compromising functionality:

1. Use complex calls to retrieve all the necessary information about objects at once, instead of first retrieving the objects, and then requesting information about them. Note, for example, that it is possible to select which fields will be returned with *listMedia*, rather than first listing media and then using *getMediaDetails* on each of the results.
2. Use XML-RPC multicalls. The XML-RPC standard makes it possible to group many independent method calls, submit them all at once, and then retrieve the results grouped as one response. The only limitation is that these calls cannot depend on results from each other.

Combining these two approaches, it should be possible to render even complex page designs at the limited cost of one or two round-trips.

11.2 Minimizing retrieval and transfer of unnecessary data

Avoid requesting all available data for objects. Use the *fields* parameter to limit the response to data that is actually required. Use *reference* filters to perform listings deep into the result set.

Avoid requesting data that will later be filtered away by web server code. Consider the use of filters in the API, and make use of the *start* parameter. Consider adding custom properties to use for filtering. If you conclude that it is not possible to perform the desired filtering using existing API functionality, don't hesitate to ask your Screen9 contact for assistance.

12 Code Examples

This section provides examples of code to be used for accessing the API.

12.1 Python XML-RPC client

This is an example of an XML-RPC client written in the Python programming language. The client connects to the server and lists some user names, then chooses one user to demonstrate upload functionality. Since sending a multipart form request is not trivial, external tool (curl) is called from Python to perform the actual upload.

```
#!/usr/bin/env python

from xmlrpclib import Server
import subprocess

# Connect to XML-RPC server
server = Server("http://xmlrpc.screen9.com")

# File to upload
FILE_TYPE = 'video'
FILE_NAME = 'path/to/movie.mp4'

common = {
    'browser': 'unknown',
    'refer': 'unknown',
    'userip': '1.1.1.1',
    'custid': 123456,
    'version': '2.0'}

# List users and pick the first one for upload
users = server.listUsers(common, 100, 1)
for user in users:
    print user['userid'], ':', user['username']
upload_user = users[0]['userid']

# Find a category called 'Default'
default_category = server.findCategory(common, ['Default'])

# Get the upload URL and parameters
staticfields = {'mediatype': FILE_TYPE}
mutablefields = ['userid', 'categoryid']
upload = server.getUploadURL(common, 300, staticfields, mutablefields)

# The provided fields should match 'mutablefields' in getUploadURL call
above
fields = {
    'auth': upload['auth'],
    'categoryid': default_category,
    'file': '@' + FILE_NAME, # '@' makes curl read the file contents
    'userid': upload_user}

# This example uploads a file using curl utility, make sure it is installed
args = ['curl', '-v']
for field in fields.iteritems():
    args += ['-F', '%s=%s' % field]
args.append(upload['url'])
subprocess.call(args)
```

12.2 Uploading from HTML page

This is an example of an HTML form used for uploading video content. Note that page headers must specify the encoding as UTF-8 in order to preserve text input strings.

```
<form action="http://xyz.screen9.com/example/upload.py?uploadid=abcdefgh"
 method=post enctype=multipart/form-data>
 <table>
  <tr><th>Title:</th>
   <td><input type="text" name="title" class="text" /></td></tr>
  <tr><th>Description:</th>
   <td><textarea name="description"></textarea></td></tr>
  <tr><th>Tags:</th><td>
   <input type="text" name="tag"><input type="text" name="tag"/>
  </td></tr>
  <tr><th>Category:</th><td>
   <input type="radio" name="categoryid" value="ABC123"/>Movies
   <input type="radio" name="categoryid" value="ABC124"/>Music
   </td></tr>
  <tr><th>File:</th>
   <td><input type="file" name="file" class="text" /></td></tr>
  <tr><td><input type="submit" value="Upload Video" /></td></tr>
 </table>
 <input type="hidden" name="success_url"
  value="http%3A//www.client.com/redirect-success.cgi" />
 <input type="hidden" name="failure_url"
  value="http%3A//www.client.com/redirect-failure.cgi" />
 <input type="hidden" name="auth" value="xxxx0000" />
</form>
```

If the upload failed for an unknown reason, an error description could be passed to the specified failure URL in the form below.

```
http://www.client.com/redirect-failure.cgi?message=unknown
```

In the above example, the optional parameter *?uploadid=<mediaid>* is appended to the upload url. This is required to support requests about upload progress. See section 3 for details on how to request upload progress.

12.3 Playing media content with a web player

The most straightforward way of presenting the media player is to generate HTML embed code using the method *getPresentation* with *embedtype* set to *playertag*. For further details, see API method *getPresentation*.

12.4 Presenting image and binary content to end users

For images and binary objects that have been uploaded into the system, the method getMediaContentURL can be used to retrieve a link to that can be placed in a web page or otherwise presented to the user.

12.5 Creating and listing cuepoints

This is an example that shows how to create a cuepoint for a given video and then list all cuepoints associated with the same video.

```
common = /*setup common struct*/
title = 'This is the title of my cuepoint'
description = 'This is where I write a longer description'
categoryid =  # Category id
userid =  # Id of user who is adding the cuepoint
mediaid = # Id of the video/audio to which I'm adding a cuepoint
starttime = 5.0 # Create the cuepoint at 5 seconds.
endtime = starttime

# create the cuepoint
try:
   cuepoint_media_id = server.createCuepoint(common, title, description,
categoryid, userid, mediaid, starttime, endtime)
except xmlrpclib.Fault, f:
   print f.faultString
print cuepoint_media_id

# List 10 first cuepoints ordered by upload
filters = {'association': mediaid, 'mediatype': 'cuepoint'}
cuepoints = server.listMedia(common, ['mediaid'], filters, 'posted', 10)
print cuepoints
```

12.6 Creating and listing subtitles

This example shows how to request an upload url and auth token to add a subtitle to a video.

```
# mediaid is the id of the video to which we are adding the subtitle
# userid is the id of the user performing the operation
# categoryid is the category of the subtitle
statics = {'mediatype': 'subtitle','association': mediaid,
           'userid': userid, 'categoryid': categoryid}
mutables = ['language', 'file']
subtitleUpload = server.getUploadURL(common, 3600, statics, mutables)
# subtitleUpload['url'] is the URL to post the subtitle file to
# subtitleUpload['auth'] is the authentication token required to do the post

# This example shows how to list all subtitles for a media and
# get the mediaid, language and textdirection of each subtitle
response = server.getMediaDetails(common, mediaid, ['subtitles'])
```

13 Revision History

Previous changes to previous revisions of this API version 2.0 document are listed below. Please note that unless explicitly deprecated, backwards compatibility with previous versions is maintained with new releases.

13.1 Revision 1, 23 May 2008

- Added 'display' event to getBandwidthStats.

13.2 Revision 2, 3 June 2008

- Added pre-roll and post-roll properties.

13.3 Revision 3, 28 August 2008

- 6 Media fields: document status_type and status_details fields. This reports information for why uploads fail to transcode such as an unsupported video format or a non-video file format. This information can be offered as an explanation to end users.
- 12.3 Viewing video content with web player: new-style example includes flashversion parameter in Javascript. This enables dynamic selection of video codecs for maximum quality depending on the Flash version installed by users.
- 3 Uploading: document titleimage field
- 5.4 Filters: document excludeproperties filter
- 6 Media fields: document image and images fields
- 8 Properties: document that properties are now general

13.4 Revision 4, 3 September 2008

- 1.1 System Overview: revise properties description for consistency with VDS usage
- 5.4 Filters: document ORed (exclude) properties, remove reference to moderation example
- 6 Media fields: Clarify status_details is for developers, not end users.
- 10.12 findCategory: include method documentation. This facilitates migration from VSS 1 to 2 as referenced in the migration documentation.
- 10.43 removeUser: include method documentation

13.5 Revision 5, 27 October 2008

- 5.3 Start: limit start parameter to 500, document negative start values
- 5.4 Filters, 6 Media Fields: document new references
- 8 Properties: document picsearch.moderation property
- 11.2 Minimizing retrieval and transfer of unnecessary data: suggest references as performance improvement
- 12.2 Uploading from HTML page: clarify that UTF-8 must be specified in form page headers
- 12.3 Viewing video content with web player: correct missing psSWFObject parameter

13.6 Revision 6, 23 January 2009

- 10.23 getMediaStreams - new method to access video content on mobile cell phones.

13.7 Revision 7, 6 February 2009

- (Internal changes only.)

13.8 Revision 8, 13 July 2009

- Renamed VSP to CSP

13.9 Revision 9, 21 January 2010

- Added new members in fields table: audiocodec, container, filename, originalsize, videocodec
- Added new members in filters table: maxposted, maxproperties, mediaids, minposted, minproperties
- Added substring filter documentation
- Documented 'src' codec option
- Documented generating SEI RDFa embed codes using getPresentation
- Updated example embed code using psswfobject to use API method getPresentation
- Documented new feature: upload progress
- Documented getting results from the end by specifying a negative start parameter.

13.10 Revision 10, 3 June 2010

- 6 Media Fields - new fields with details of available media formats and processing status (formats, processed, processing_progress)
- 7 Options, 8 Properties - document upload callbacks (picsearch.uploadcallback, picsearch.uploadcallbackactions)
- 10.30 listCategories - document 'anyparent' option, supporting flat list of all categories.

13.11 Revision 11, 8 July 2010

- 1 Introduction – add reference to separate add-on guides, remove redundant address setup and logotype text.
- 1.1 System Overview, 1.2 Media types, 2.7 Permaids, 3 Uploading, 5.4 Filters, 10.1 addCategory, eventStatsByPeriod, 10.16 getBandwidthStats, 10.18 getMediaContentLink, 10.19 getMediaContentURL, 10.26 getRssFeed – replace video only references with generic media to include audio.
- 4 Viewing videos – rename to ‘Playing media’, mention audio.
- 6 Media fields – add audio column, mention audio, add new audio fields.
- 7 Options – new audio options.
- 8 Properties – new audio properties.
- 10.23 getPresentation – add *component_links* for flexible embedding, document *showembedcode*, clarify returned struct.
- 12.1 Python XML-RPC client, 12.5 Sample XML-RPC request – update examples for video media.
- 12.3 Viewing video content with web player – renamed with ‘playing media’

13.12 Revision 12, 27 September 2010

- Rename ‘Content Sharing Platform’ (CSP) to ‘Video Platform API’ throughout the document for consistency with other published material.
- 6 Media Fields – update *thumbnail* and *image* fields with audio media support.
- 8 Properties – document *thumbnail* and *titleimage* properties controlling *image* field.
- 10.3 assignRating – document the return value.
- 10.16 getAjaxAuth – new method supporting authorization for separate Screen9 Ajax API, designed to facilitate implementing highly interactive dynamic web pages including videos.
- 10.30 listCategories – remove description of unused *options*, add new *filters* parameter.
- 10.36 listTags – add documentation for method.

13.13 Revision 13, 26 January 2011

- Introduce new Screen9 brand for video products.
- 4 Playing media – update example for playback with recommended universal embed code.
- 10.20 getMediaContentURL – document RTMP streaming support.
- 10.24 getPresentation – added ‘universal’ value for ‘embedtype’ option, supporting automatic detection of many video playback devices including both computer web browsers and smart phones.
- 10.24 getPresentation – documented ‘containerid’ parameter, supporting multiple players per HTML page
- 12.1 Python XML-RPC client – simplified and improved code example.

13.14 Revision 14, 20 September 2011

- 1 Introduction – remove references to migration from Picsearch to Screen9.
- 1.1 Features Supported With Add-on Packages – new section.
- 3 Uploading – replace *X-Picsearch-mediaid* with *X-Screen9-mediaid*.
- 3.1 Subtitle Support – new section describing subtitle captions support.
- 7 Options, 8 Properties – document *geolocking* and *secureticketing* options and properties to restrict access within countries and enhance security of content delivered to players.
- 7.1 Secure Streaming – new section regarding RTMPE streams.
- 8 Properties - add *publishfrom* and *publishuntil* properties supporting publishing window times for time limited playback.
- 10.5 associateMedia – reference associating subtitle objects.
- 10.19 getMediaContentLink – deprecated call, please use getPresentation instead.
- 10.20 getMediaContentURL, 10.22 getMediaStreams, 10.24 getPresentation – add ‘userip’ to options for restricting IP access to content.
- 10.24 getPresentation – add *autoplay, subtitles* and *wmode* options supporting automatic playback start, subtitles and alternative video rendering modes.
- 10.31 listCategories – correct *filters* parameter, remove deprecated *parentid* parameter.

13.15 Revision 15, 16 December 2011

- Cosmetic fixes only.

13.16 Revision 16, 12 March 2012

- 1.2 System Overview, 5.5 Comment type, 10.2 assignComment, 10.9 countMediaComments, 10.11 countUserComments, 10.33 listMediaComments, 10.42 removeComment – deprecate unused support for comments.
- 3.1 Subtitle Support, 10.5 associateMedia, 10.24 getPresentation – move subtitle support into separate documentation.
- 7 Options – add showsuggested for recommended videos.
- 8 Properties – correction.
- 10.14 eventStatsByPeriod, 10.17 getBandwidthStats – remove deprecated download_complete event type.
- 10.27 getUploadURL – clarify specifying multiple values.

13.17 Revision 17, 1 February 2013

- 3. HTML field table – added endtime and starttime field names.
- 6. Media fields – Added cuepoint mediatype column and mediafields starttime and endtime.
- 8. Properties – Added picsearch.cuepointtype key.
- 10.5 createCuepoint – new API call.
- 10.39 removeMedia – Function can also operate on cuepoint objects.
- 12.5 Added example to show creating and listing of cuepoints.

13.18 Revision 18, 6 March 2013

- 10.5 cloneMedia added
- Function calls createNode and createCuePoint were moved to correct alphabetical sorting
- 3 Uploading – replace *X-Screen9-mediaid* with *X-Picsearch -mediaid*.
- Added size limit of 2000 to property keys and values in 10.50 setProperty.

13.19 Revision 19, 26 May 2013

- 3 Uploading, failure_url and success_url are no longer required parameters.
- 5.1 Added parameter *encryption* to common struct table.
- 10.21 getPresentation documented how to create responsive embed codes.
- 12.1 Added better upload example.
- 1.3 and 6 removed deprecated Node object.
- 2.7, 9, 10 remove references to deprecated functionality comments.
- 10.25, 10.31 and 10.46 document additional sort orders *title* and *transcodedsize*.
- 10.19 getMediaStreams added parameters period and trackingid.
- 10.24 Added getReplacementUploadURL.
- 3 and 6 added information about subtitles.
- 7 and 8 removed internal options and properties.
- 12.6 Added subtitles example.

13.20 Revision 20, 20 November 2013

- Added filter ‘public’ to Media filters table 5.4.
- Added media field ‘clones’ to Media fields 6.
- Removed example 12.7 sample xml-rpc response

13.21 Revision 21, 08 July 2016

- 10.2 alterLiveEvent added
- 10.18 getMediaContentURL – updated *options* parameter description for ‘protocol’
- 10.20 getMediaDownloadURL added
- 10.40 registerLiveEvent added
- 12.1 Updated Python code example
- 6 Media fields –Added *live* parameter