Lightspeed Live Stream
Web API Reference

Live Stream 3.1.6

August 2022

324452

Copyrights and Trademark Notices

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Microsoft. Microsoft, Windows NT|2000|XP|XP Professional|Server 2003|Server 2008 |Server 2012|Server 2016, Windows 7, Windows 8, Media Player, Media Encoder, .Net, Internet Explorer, SQL Server 2005|2008|2012, and Windows Media Technologies are trademarks of Microsoft Corporation.

SharpSSH2. SharpSSH2 Copyright (c) 2008, Ryan Faircloth. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Diversified Sales and Service, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Swagger. Licensed from SmartBear.

Telerik. RadControls for ASP.NET AJAX copyright Telerik All rights reserved.

VoiceAge. This product is manufactured by Telestream under license from VoiceAge Corporation.

x264 LLC. The product is manufactured by Telestream under license from x264 LLC.

Xceed. The Software is Copyright ©1994-2012 Xceed Software Inc., all rights reserved.




MPEG Disclaimers

MPEGLA MPEG2 Patent

ANY USE OF THIS PRODUCT IN ANY MANNER OTHER THAN PERSONAL USE THAT COMPLIES WITH THE MPEG-2 STANDARD FOR ENCODING VIDEO INFORMATION FOR PACKAGED MEDIA IS EXPRESSLY PROHIBITED WITHOUT A LICENSE UNDER APPLICABLE PATENTS IN THE MPEG-2 PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, LLC, 4600 S. Ulster Street, Suite 400, Denver, Colorado 80237 U.S.A.

MPEGLA MPEG4 VISUAL

THIS PRODUCT IS LICENSED UNDER THE MPEG-4 VISUAL PATENT PORTFOLIO LICENSE FOR THE PERSONAL AND NON-COMMERCIAL USE OF A CONSUMER FOR (i) ENCODING VIDEO IN COMPLIANCE WITH THE MPEG-4 VISUAL STANDARD ("MPEG-4 VIDEO") AND/OR (ii) DECODING MPEG-4 VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL AND NON-COMMERCIAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION INCLUDING THAT RELATING TO PROMOTIONAL, INTERNAL AND COMMERCIAL USES AND LICENSING MAY BE OBTAINED FROM MPEG LA, LLC. SEE HTTP://WWW.MPEGLA.COM.

MPEGLA AVC

THIS PRODUCT IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD ("AVC VIDEO") AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

MPEG4 SYSTEMS

THIS PRODUCT IS LICENSED UNDER THE MPEG-4 SYSTEMS PATENT PORTFOLIO LICENSE FOR ENCODING IN COMPLIANCE WITH THE MPEG-4 SYSTEMS STANDARD, EXCEPT THAT AN ADDITIONAL LICENSE AND PAYMENT OF ROYALTIES ARE NECESSARY FOR ENCODING IN CONNECTION WITH (i) DATA STORED OR REPLICATED IN PHYSICAL MEDIA WHICH IS PAID FOR ON A TITLE BY TITLE BASIS AND/OR (ii) DATA WHICH IS PAID FOR ON A TITLE BY TITLE BASIS AND IS TRANSMITTED TO AN END USER FOR PERMANENT STORAGE AND/OR USE. SUCH ADDITIONAL LICENSE MAY BE OBTAINED FROM MPEG LA, LLC. SEE HTTP://WWW.MPEGLA.COM FOR ADDITIONAL DETAILS.

Limited Warranty and Disclaimers

Telestream, LLC (the Company) warrants to the original registered end user that the product will perform as stated below for a period of one (1) year from the date of shipment from factory:

Hardware and Media—The Product hardware components, if any, including equipment supplied but not manufactured by the Company but NOT including any third party equipment that has been substituted by the Distributor for such equipment (the "Hardware"), will be free from defects in materials and workmanship under normal operating conditions and use.

Warranty Remedies

Your sole remedies under this limited warranty are as follows:

Hardware and Media—The Company will either repair or replace (at its option) any defective Hardware component or part, or Software Media, with new or like new Hardware components or Software Media. Components may not be necessarily the same, but will be of equivalent operation and quality.

Software Updates

Except as may be provided in a separate agreement between Telestream and You, if any, Telestream is under no obligation to maintain or support the Software and Telestream has no obligation to furnish you with any further assistance, technical support, documentation, software, update, upgrades, or information of any nature or kind.

Restrictions and Conditions of Limited Warranty

This Limited Warranty will be void and of no force and effect if (i) Product Hardware or Software Media, or any part thereof, is damaged due to abuse, misuse, alteration, neglect, or shipping, or as a result of service or modification by a party other than the Company, or (ii) Software is modified without the written consent of the Company.

Limitations of Warranties

THE EXPRESS WARRANTIES SET FORTH IN THIS AGREEMENT ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. No oral or written information or advice given by the Company, its distributors, dealers or agents, shall increase the scope of this Limited Warranty or create any new warranties.

Geographical Limitation of Warranty—This limited warranty is valid only within the country in which the Product is purchased/licensed.

Limitations on Remedies—YOUR EXCLUSIVE REMEDIES, AND THE ENTIRE LIABILITY OF TELESTREAM, LLC WITH RESPECT TO THE PRODUCT, SHALL BE AS STATED IN THIS LIMITED WARRANTY. Your sole and exclusive remedy for any and all breaches of any Limited Warranty by the Company shall be the recovery of reasonable damages which, in the aggregate, shall not exceed the total amount of the combined license fee and purchase price paid by you for the Product.

Damages

TELESTREAM, LLC SHALL NOT BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OR INABILITY TO USE THE PRODUCT, OR THE BREACH OF ANY EXPRESS OR IMPLIED WARRANTY, EVEN IF THE COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF THOSE DAMAGES, OR ANY REMEDY PROVIDED FAILS OF ITS ESSENTIAL PURPOSE.

Further information regarding this limited warranty may be obtained by writing:
Telestream, LLC
848 Gold Flat Road
Nevada City, CA 95959 USA

You can call Telestream via telephone at (530) 470-1300.

Part number: 324452

Date: August 2022

Contents

Overview

The Lightspeed Live web APIs—Stream, Capture, and Source —enable you to monitor and control your Lightspeed Live system within a broader, web services-based system or create your own custom application.

You can also create a web services-based program to control streaming and capture beyond the functionality or capability of Telestream's general-purpose Lightspeed Live Capture and Stream web applications, to meet your organization's requirements.

- Lightspeed Live Stream Web API
- Lightspeed Live Capture Web API
- Lightspeed Live Sources Web API
- Ports for Lightspeed Live Server Access
- Using Shared vs. Dedicated Components
- Establishing Authorization for Stream Operations
- Obtaining Help for Stream & Source Operations
- Operation & Response Formats

Note: This reference assumes that the programming environment being used by the developer includes a library that abstracts the process of operation submission and responses through the HTTP protocol.
If your environment does not include a library to perform this abstraction then you will have to directly format your operations to adhere to the HTTP protocol.
See Hypertext Transfer Protocol for details.

Lightspeed Live Stream Web API

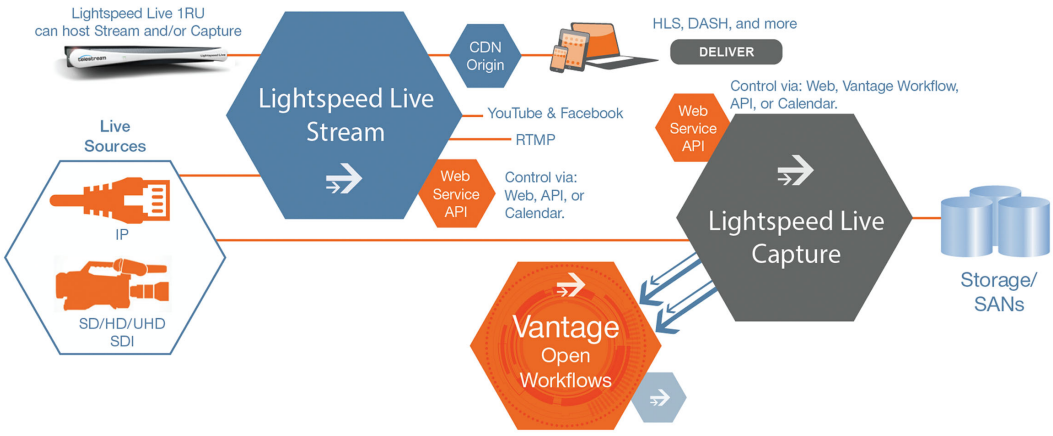
Lightspeed Live Stream enables you to encode source files in real-time, streaming them via the Lightspeed Live Stream web application or through the HTTP web service *Stream Operations* described in this reference.

Overview

Use of the Stream API may require the use of an API token for security purpose. See *Establishing Authorization for Stream Operations* for details on how tokens can be enabled or disabled and how to use it in a custom Live Stream application.

The Live Stream web API is implemented in the Lightspeed Live Stream web service, and is implemented over HTTP as a REST interface, with most results returned in JSON format. A few operations return thumbnails in JPEG format or XML files intended for import/export.

You can use the Lightspeed Live Stream web service operations in a program to control and monitor streaming on the Lightspeed Live Server. Programs written to implement Lightspeed Live streaming can supplement the features provided by the general-purpose Lightspeed Live Stream web application provided by Telestream.



Note: Stream operations can only be executed in a secure environment. A *token* field must be implemented in the HTTP header for each Stream operation you execute, supplying the authorization token generated by the Lightspeed Live server.
For an overview of Stream security and implementation, see *Establishing Authorization for Stream Operations*.

Lightspeed Live Stream supports adaptive bit rate encoding for SD, HD and UHD sources into AVC and HEVC. Input support is available for SDI as well as IP sources, offering future-proof operation as delivery mechanisms change. Output can be delivered via RTMP or as HTTP Live Streaming (HLS) and MPEG DASH packages.

Stream operations include both GET and POST operations, and they are organized into functional categories to facilitate easier implementation.

Each of the Lightspeed Sources operations (end points) are described in *Source Operations*.

Limits of the API

The Live Stream API is generally intended for control and monitoring. You can not create most Live Stream components or configure them using the API. Before you can control and monitor streams on the Lightspeed Live Server, you must first create and configure them using the Lightspeed Live web application.

Live Stream Component Hierarchy

When managing and controlling components in Live Stream, it is helpful to understand the organization and hierarchy of components:

- Machine > Source > SourceTrack
- Machine > Source > TextTrack
- Program > Rendition > Segment > Material
- Encoder > Stream
- Package > Variant
- Channel > CalendarEvent

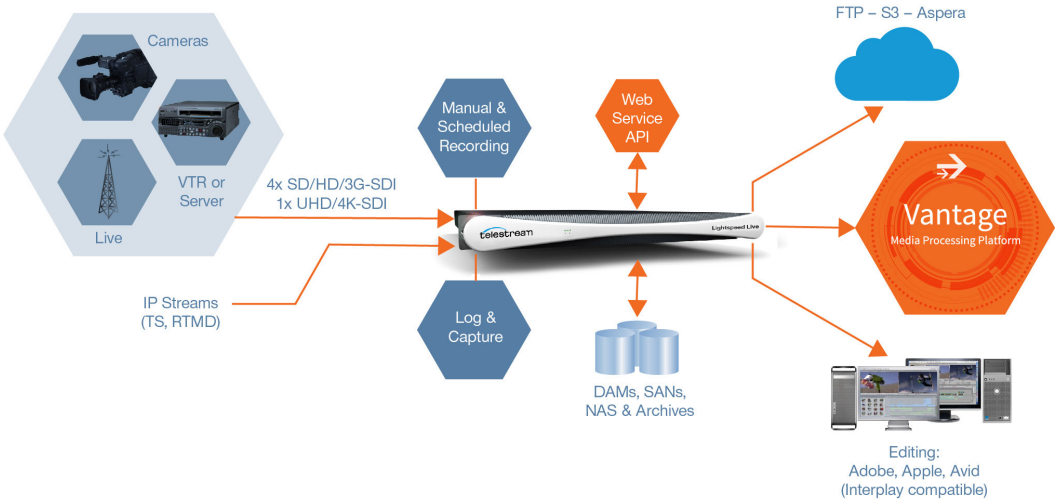
Whenever you are targeting a specific component, you must first identify each of the superior components in the chain by GUID. For example, if you are operating on a specific segment, you must first obtain the GUID of the program, then the rendition, and finally the segment you are targeting. This sequence of operations is documented in each operation's Operation Sequence topic.

Using Live Stream Groups via the API

A Live Stream group is a set of Live Stream servers organized into a single, functional system, enabling you to increase scalability without significantly increasing complexity. Where reference is made to a Live Stream server, it may be a single server or a group of servers—there is no difference in functionality or reporting.

Lightspeed Live Capture Web API

Lightspeed Live Capture enables recording of live streaming media in a Lightspeed Live Capture workflow to be controlled manually via the Lightspeed Live Capture web application or through the HTTP web service *Capture Operations* described in this reference. The Capture web API is implemented in the Lightspeed Live Capture web service.



Lightspeed Live Capture workflows can be configured to publish a web service that enables media clips to be recorded using this web services operation set.

In order to respond to Capture API operations, the Capture action in the workflow must be configured with a Web Service trigger (see the Lightspeed Live Capture User Guide), and the specified port (default: 17000) must be utilized as the target port in each operation. When the workflow is activated in Lightspeed Live Capture, the web service listens for requests on the specified port. (The user will be warned if the port is in use by another Capture workflow or other service on the host computer.)

Your client program can control any Live input source on the Lightspeed server by communicating with the target Capture workflow—which in turn is configured for a specific input and web service port. Your application may be designed to target a specific input source by communicating on a specific port (thus, a specific workflow and SDI/IP input) on the domain or it may be more broadly-designed (using the Live APIs in conjunction with the Vantage SDK) to obtain a list of currently-running workflows, and present those to the user for selection dynamically.

Note: Depending on the requirements of your Capture application, you may also integrate the Vantage SDK in your Capture program along with the Lightspeed Live Capture web service. Integrating the Vantage SDK enables you to programmatically control the target workflow as well as the capture operation; starting and stopping the workflow, testing its status, querying job results, and submitting jobs, for example.

Lightspeed Live Sources Web API

The Lightspeed Live Sources Web API supports the insertion of SCTE-35 and ID3 tags. Use of the Sources API to insert tags can be performed during streaming or capture sessions, in conjunction with the Stream API and the Capture API. The Sources API is implemented in the Source service.

The Live Source API is implemented over HTTP as a REST interface, and results (if any) are returned.

The Lightspeed Sources web API is described in *Source Operations*.

Ports for Lightspeed Live Server Access

These ports are used to access various services on Live Stream servers:

- *Port 8089*—default port for accessing the Live Stream web application on a Live Stream server.
- *Port 8090*—default port for accessing the Source Manager web application on a Live Stream server.
- *Port 17000*
  - Default port for accessing the Lightspeed Live server via the Capture API.
  - Default port for accessing the Capture API help system.
- *Port 18000*
  - Default port for accessing the Lightspeed Live server via the Stream API.
  - Default port for accessing the Stream API help system.
- *Port 15000*
  - Default port for accessing the Lightspeed Live server via the Sources API.
  - Default port for accessing the Sources API help system.

The ports for accessing the Lightspeed Live server via the Stream API and help, as well as the Source API can be changed in the Lightspeed Live Stream web app.

Under Settings, update the Live Stream REST API Server port or the web app Server port. After changing a port, you must restart the services for the new port number to take effect.

Using Shared vs. Dedicated Components

Operations on two types of Stream and Source components—media sources and output channels—are associated with a port directly on a specific Lightspeed Live server and thus, must be executed directly on that server. For example, when you are controlling a media source, you must execute the operation using the DNS name or the IP address of the Live Stream server where the source was created.

Operations on other components—programs, encoders, and packages, for example—are not tied to a specific server; they are available to all servers in the group, and they can be shared. Thus, you can address the operation to any server in the group and the operation will execute properly, regardless of where the component was created and configured.

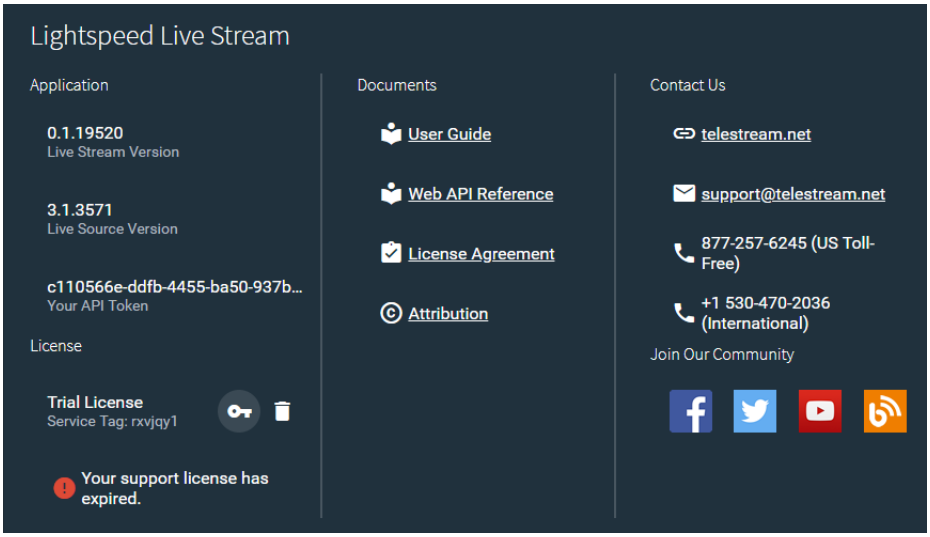
Establishing Authorization for Stream Operations

Use of Lightspeed Live Stream via the Live Stream API (including accessing Stream help topics) may require the use of an API token for security purposes.

Note: For complete details on enabling or disabling the use of API tokens, see the Live Stream User's Guide > Using the Live Stream Web Application chapter > Advanced topic.

When Require API Tokens is enabled in the Live Stream Web app, an API token is required to access and utilize Lightspeed Live Stream via the API in a custom program. When disabled, they are not required to use Lightspeed Live Stream via the API. Generally, you should determine if you do or do not require API token usage, and design the program accordingly.

When Require API Tokens is enabled, Administrator and API users can view (and copy) their unique and permanent API token in the Web app's Settings About panel.



API tokens are generated for two types of users: Administrators and API users. Administrator and API users both have complete access to the API. In the context of API utilization, there is no distinction—you could either use a token from an API user or an Administrator user.

The token does not display entirely in the column. You can triple-click the token and copy it and then paste it into your program or some other file for delivery to a programmer, for example.

Follow these steps to use the target server via the Live Stream API with an API token:

1. **Obtain the API token**: The token does not display entirely in the column. Triple-click the token and copy it and then paste it into your program or some other file for delivery to a programmer, for example.
2. **Use the token to control streaming**: Add a *token* field to the HTTP header for each Stream operation you execute, and insert the API token in the field.

Obtaining Help for Stream & Source Operations

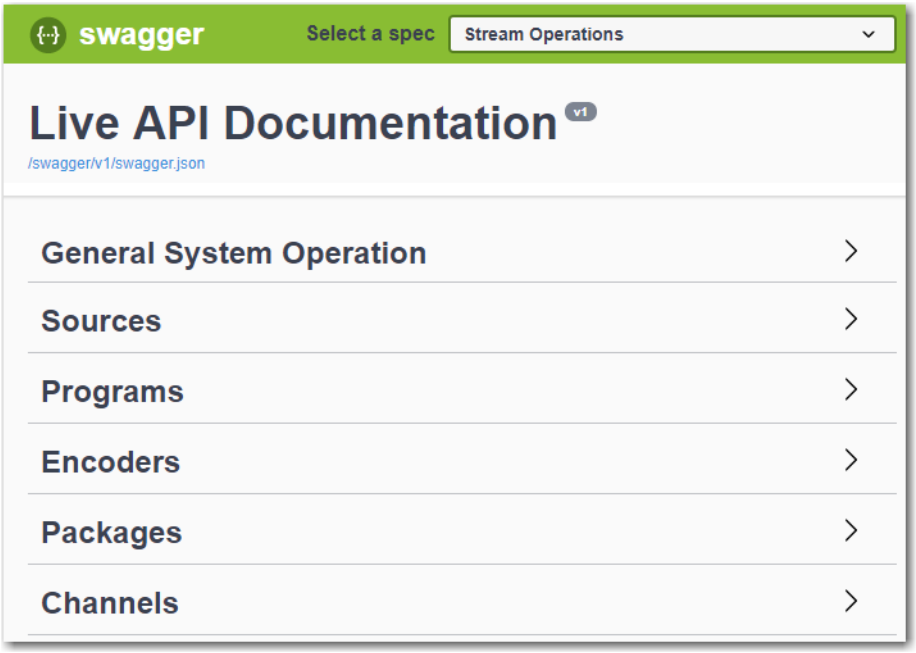
Stream and Source operations web API help shows you the operations organized by service, how they are used, and the required/optional arguments. You can use Help to test operations before implementing them in your program. You can also access Help directly in Postman.

Displaying API Help for Live Stream

To display Help for Live Stream, execute the *help* operation in a Web browser at the root level: `http://<host>:<port>/help`. For example: `http://10.1.1.1:18000/help` or `http://mylivestreamserver:18000/help`.

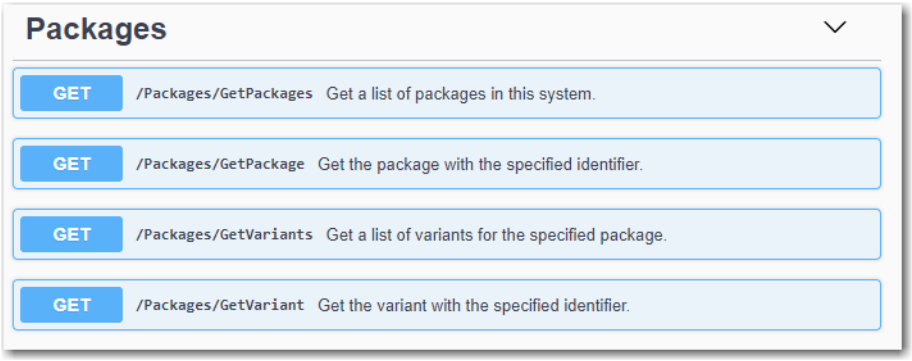
Note: Port 18000 is the default port for accessing the API. To view or change this port, see *Ports for Lightspeed Live Server Access*.

Live Stream displays a web page of the Live Stream operations by functional category:



Displaying the Operations in a Specific Category

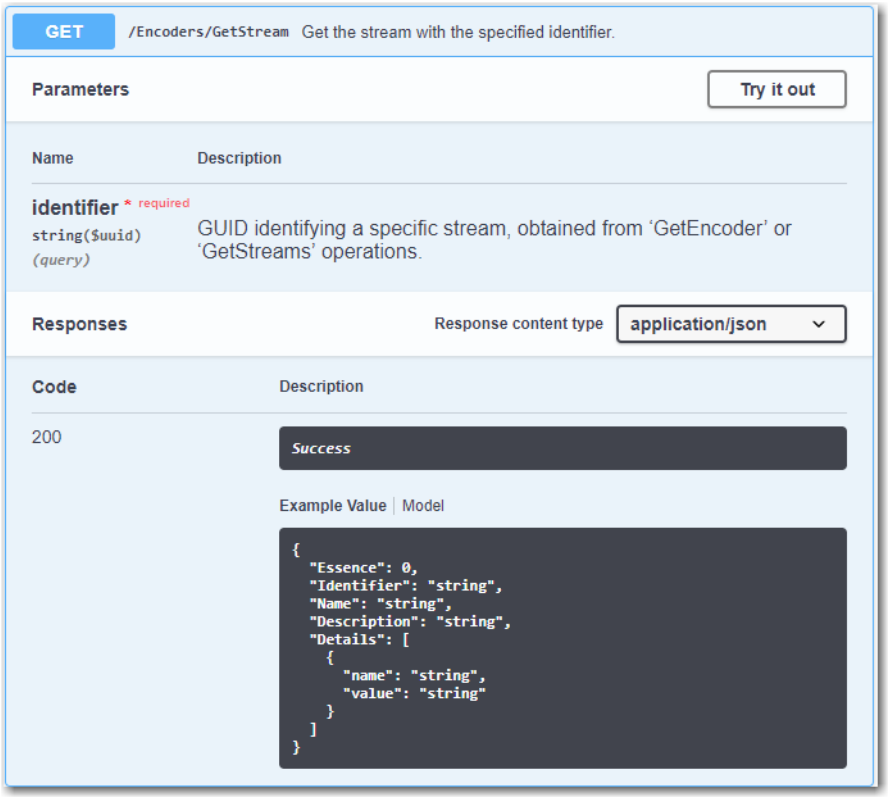
To display the operations in a specific category, click on the service to open the panel and display its operations:



When you're done, you can click the service again to close the list.

Displaying Operation Details

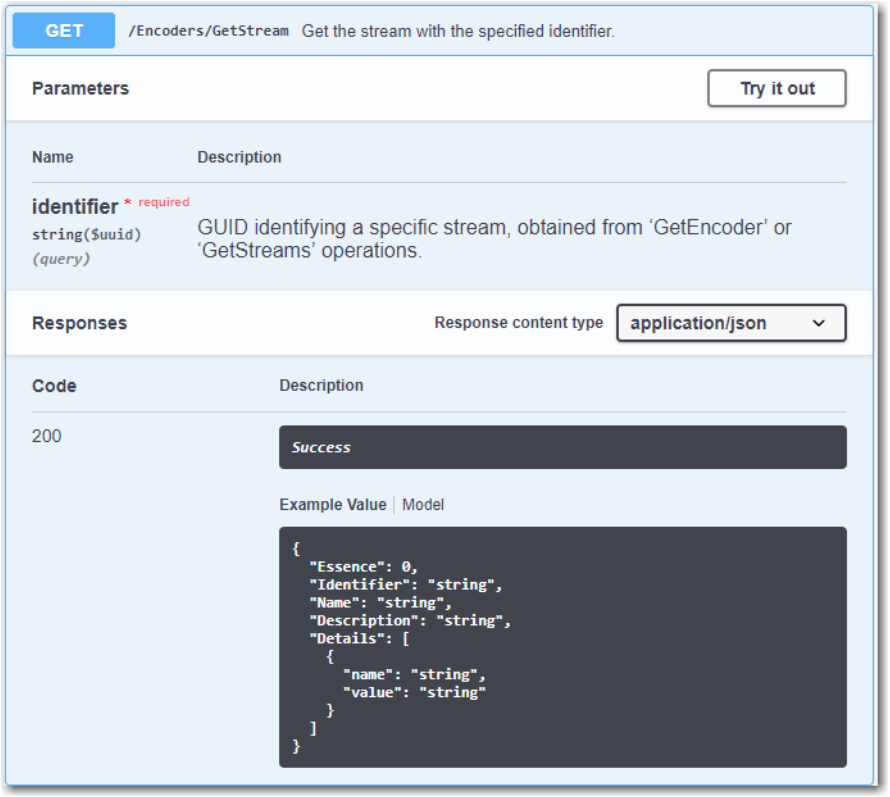
To display an operation's details, click on the operation to open the panel and display its parameters and responses:



When you're done, you can click the operation again to close the details panel.

Testing an Operation

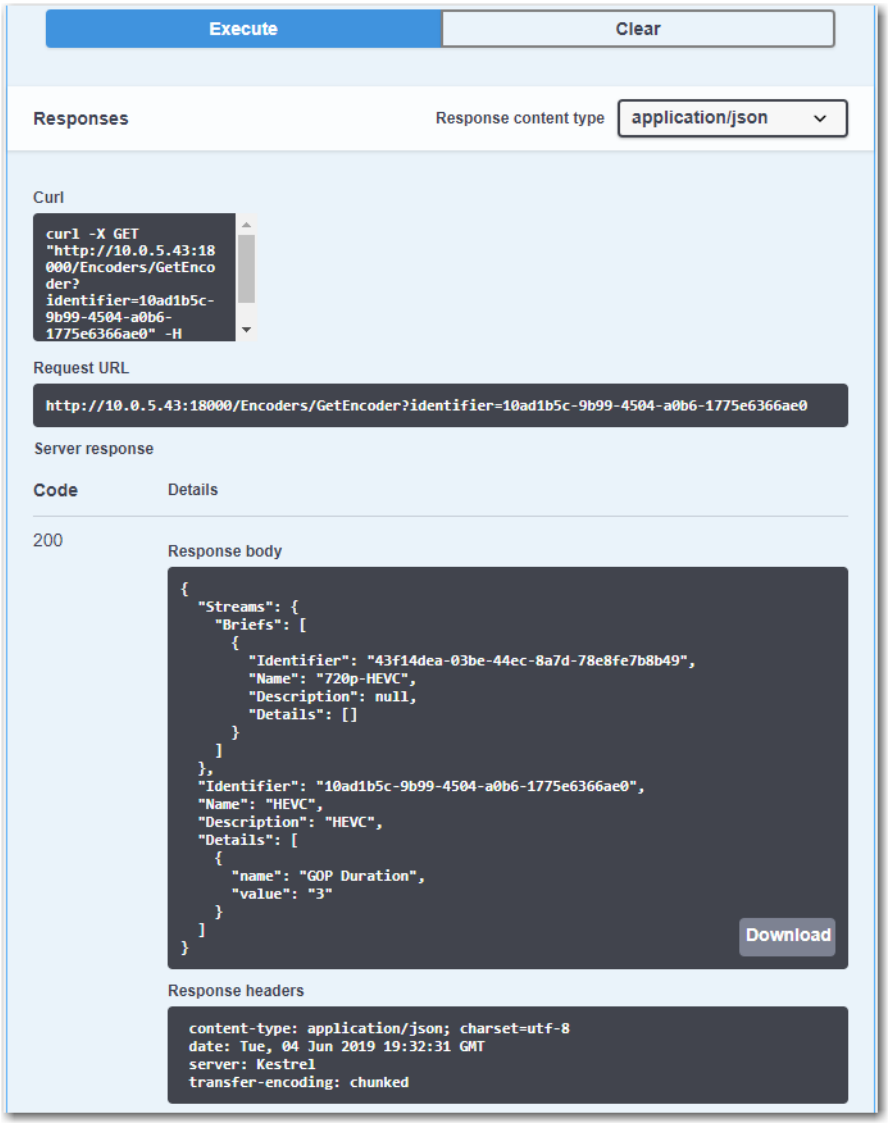
To execute an operation directly in the Help system for testing, debugging, or troubleshooting, click on the operation to open the panel and click the Try It Out button at the top right:



The response content type should be set to application/json.

Provide parameter values as required, and click Execute.

Live Stream uses cURL to execute the operation and returns the server response. Help displays the URL and the response—the response code, and the body and header.



You can also save the response body as a file—click on Download to save the file for use in other programs.

Exporting Live Stream and Sources Help in Postman

If you use Postman and want in-application access to the Live Stream or Live Sources help, you can export it from a Lightspeed Live server and then import it into Postman. To export the API help as a file, click on the */swagger/v1/swagger.json* link directly below the Web page's title. Import this file to use it directly Postman (in Collections, as *Live API Documentation*). You just need to customize the operation to address your specific Lightspeed Live server.

Operation & Response Formats

The Lightspeed Live APIs are a RESTful implementation. Most Lightspeed Live web service operations are invoked using an HTTP GET request in the following form:

```
http://<host>:<port>/<API Category>/<operation>
[?<parameter>=<value>[&<parameter>=<value>]]
```

Operations that alter operational data in the Live Stream server are POST operations; such as *AddCalendarEvent*, for example. Live Stream POST operations use an application/JSON-formatted request body for transferring parameters to the Live Stream web service.

Lightspeed Live responds to operations with an HTTP status line (for example: *200 OK* or *404 Not Found*), HTTP headers, and either XML or JSON-formatted responses in the body. Responses vary, based on the operation and parameters.

Stream API and Source API responses are primarily in JSON (*application/json*) format (although a few operations return data as XML, ZIP files, JPEG thumbnails, etc.)

The Capture API includes POST operations as well, although responses are in XML format.

Topics

Operation Keyword Terms

Keyword terms in each operation are shown in this reference surrounded by less than and greater than symbols (<>); they are placeholders in the operation's description, to be replaced by values you specify.

Note: Operation names are not case-sensitive. For example, you can specify *Encoders/GetStreams* or you can specify *encoders/getstreams* to execute the *GetStreams* operation.

When an operation has required and/or optional parameters, they are displayed as name/value pairs in the query portion (?) of the request. Additional parameters are separated by an ampersand (&). Parameters in brackets ([]) are optional:

```
http://<host>:<port>/record/start
[?<parameter>=<value>[&<parameter>=<value>]]
```

Here are the keyword terms you'll encounter in parameters:

Term	Description
host	The Windows domain name or the IP address of the Lightspeed Live Capture or Stream server you are targeting. For example: *localhost* | `LightspeedServer` | `192.168.1.23`.
port	The TCP port number assigned to the web service. When using the Live Capture API, the port number used in the operation identifies the workflow whose jobs you are controlling—thus, when you have multiple workflows, each workflow should be configured with a different port if they are going to run concurrently. The port (default: 17000) is displayed and configured in the Capture action inspector of the target workflow. (See the Lightspeed Live Guide or man page for the Capture action in your Vantage workflow.) The following API server ports are selectable in the Live Stream web app settings. • Live Stream API server port (default: 18000) • Source API server port (default: 15000)
parameter	An optional, named parameter defined by the web service operation. Parameters are listed and described in the associated table in each operation.
value	The value for the associated parameter.

Use of GUIDs/UUIDs in Operations

A GUID (Globally Unique IDentifier)—referred to as UUID in Capture operations—is used in most operations to target or identify a specific instance of a component. For example, a stream or program. The important property of a GUID is that each value is globally unique, enabling you to identify a specific target using the GUID. The value is generated by an algorithm, developed by Microsoft, which assures this uniqueness.

A GUID is a 16-byte binary data type that can be logically grouped into the following subgroups: 4byte-2byte-2byte-2byte-6byte.

The standard textual representation is {12345678-1234-1234-1234-1234567890AB}.

For example, `ad1c45b7-67fb-419d-8c5b-8ba474bd6dfd`.

Note: If the GUID you supply in an operation is not properly formed, the operation fails to execute and returns an HTML XML advising of the Request Error.

Using GUIDs in Capture Operations

When you are operating on a specific clip, the GUID is first obtained when you begin capturing the clip, via the Start operation. In the response, the GUID is returned in the UUID element.

Use the clip's GUID when you use Modify, MarkIn/MarkOut, EditIn/EditOut, Stop, and Status operations to identify the target clip.

Using GUIDs in Stream Operations

When you are requesting information about a specific component from the system (such as a source track), the GUID is the identifier of that component.

You obtain GUIDs for specific components in your system using the Get operation for the component set (without a GUID): *GetPrograms*, for example.

When you are retrieving multiple components (for example, all tracks), the GUID is the identifier of the parent component.

For example, in this hierarchy: *Machine* > *Source* > *Source Track*...

If you want to retrieve a specific Source or Source Track, you provide its GUID identifier. If you want to retrieve all of the Sources that belong to a Machine or all of the Source Tracks that belong to a Source, you supply the identifier of the Machine or Source.

Use of Timecodes in Operations

Timecodes are used as parameters and are also returned as part of responses.

Timecodes can be specified in either drop frame (for example: 01:00:10;00) or non-drop frame format (for example: 01:00:10:00). The recorded file's timecode format always matches the input source's timecode regardless of the type of timecode notation used within a Web API command. Timecode notation types are treated identically and have no effect on the output file's timecode.

The timecode source is determined by the configuration of the source input. The timecode may be Source, Computer Clock, Free Run, Analog LTC, or RS422, as defined in the Vantage Domain Console's Capture Inventory Capture Source settings.

Response Formats

Response formats vary by API and by operation.

Capture Operation Response Formats

When you execute a Capture operation, the Capture service executes the operation and returns an XML response. For example:

```
<Response>
  <UUID>27638f24-2bb1-4ce6-8816-5a05a5e05897</UUID>
  <PercentCompleted>0</PercentCompleted>
  <Progress>0</Progress>
  <ActionDuration>3239.9733066</ActionDuration>
  <FPS>29.97002997003</FPS>
  <Start>11:05:06;09@29.97</Start>
  <End></End>
  <MarkIn>11:05:06;09@29.97</MarkIn>
  <MarkOut>11:59:06;09@29.97</MarkOut>
```

```
  <Excluding>False</Excluding>
  <Name>SDI-2 - Web UI LSL-PM - SDI Input 2.8</Name>
  <MasterMobid>060a2b3401010101010f0013-000000-56967d89-b107-
0919-060e-2b347f7f2a80</MasterMobid>
  <SourceMobid>060a2b3401010101010f0013-000000-56967d89-b108-
0919-060e-2b347f7f2a80</SourceMobid>
  <HorizontalResolution>1920</HorizontalResolution>
  <VerticalResolution>1080</VerticalResolution>
  <FrameRate>29.97002997003</FrameRate>
  <Channels>16</Channels>
  <State>Opened</State>
  <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
  <EngineState>Running</EngineState>
  <EngineTime>11:02:38;01</EngineTime>
  <XMLRevision>2</XMLRevision>
</Response>
```

Stream and Source Operation Response Formats

When you execute a Stream or Source operation, the Stream | Source web service returns an application/JSON response. For example:

```
[
  {
  "Description":null,
  "Identifier":"89d2be4b-be21-4838-a551-522cce299fbe",
  "Name":"LL-PM-1"
  },
  {
  "Description":null,
  "Identifier":"4bd2be89-2c24-3947-a432-484bca2387fba",
  "Name":"LL-PM-1"
  },
  {
  "Description":null,
  "Identifier":"43b2ff5b-ac29-48573-c443-567cad734efa",
  "Name":"LL-PM-1"
  }
]
```

Reserved Characters in Value Strings

This list of reserved characters can cause a variety of problems in parameters—you should not use them in parameter values:

```
" < > # % { } | \ ^ ~ [ ] ` ; / ? : @ = & +
```

During processing, certain characters are omitted; others truncate the remainder of the string or are changed to a space character. In some circumstances, this error is displayed: “Request Error - The server encountered an error processing the request. See server logs for more details.”

Stream Operations

You can use the Lightspeed Live Stream web service operations in a program to control and monitor streaming on the Lightspeed Live Server. Programs written to implement Lightspeed Live streaming can supplement the features provided by the general-purpose Lightspeed Live Stream web application provided by Telestream.

Note: Stream operations can only be executed in a secure environment. A *token* field must be implemented in the HTTP header for each Stream operation you execute, supplying the authorization token generated by the Lightspeed Live server.
For an overview of Stream security and implementation, see *Establishing Authorization for Stream Operations*.

Stream operations are organized into functional categories to facilitate easier implementation.

- General System Operations
- Sources Operations
- Programs Operations
- Encoders Operations
- Packages Operations
- Channels Operations
- Social Media Platform Channel Management Operations

Note: To display help for Stream operations, enter `http://<host>:<port>/help`. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

General System Operations

Use the following operations to perform system-level tasks and obtain system-level information, including a list of servers in the system, Live Stream services, and system

settings, as well as authentication operations which are required for using social media platform operations.

- GetMachines
- GetClusterMembers
- GetSystemSettings
- ImportSystemSettings
- ClearAllSettings
- GetAuthentications
- GetAuthentication
- GetNewAuthenticationDetails
- CompleteAuthentication

Note: All System operations are located at the root: `http://<host>:<port>/`.

In this category, t he authentication operations are arranged hierarchically by precedent—their typical operational order:

GetMachines

GetSystemSettings | Import System Settings | Clear All Settings

GetClusterMember

GetAuthentications
|
GetAuthentication
|
GetNewAuthenticationDetails
|
CompleteAuthentication

Note: To display help for system operations, execute `http://<host>:<port>/help` and open the General System Operation panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

GetMachines

This GET operation identifies the Live Stream server (or in the case of a group, all of the Live Stream servers in the group) by GUID. This operation is a prerequisite for Sources operations which require a machine GUID, plus *GetMachineStatistics*.

To execute this operation, use the Windows domain name or the IP address of the Live Stream server or any Live Stream server in the group.

URL Format

GetMachines has the following format:

```
http://<host>:<port>/GetMachines
```

Operation Sequence

No other operations are required before you can execute this operation.

Results

Upon success, *GetMachines* returns a Live Stream server document with an array of servers.

Example

```
http://mylivestreamserver:18000/GetMachines
```

Typical Response

In this response, three servers are listed with their Identifier and Name. You can extract each machine's GUID from the Identifier and use it to connect and monitor or control its resources. Of course, in a standalone system, only one server object is returned.


```
[
  {
  "Description":null,
  "Identifier":"89d2be4b-be21-4838-a551-522cce299fbe",
  "Name":"LL-PM-1"
  },
  {
  "Description":null,
  "Identifier":"4bd2be89-2c24-3947-a432-484bca2387fba",
  "Name":"LL-PM-1"
  },
  {
  "Description":null,
  "Identifier":"43b2ff5b-ac29-48573-c443-567cad734efa",
  "Name":"LL-PM-1"
  }
]
```


GetClusterMembers

This GET operation returns a list of the Live Stream systems in the target cluster. Live Stream systems can be grouped together into a *cluster*, so that their configuration data is shared for redundancy purposes. (The Sources and Channels are created on a server-by-server basis, and remain unique to each server in the cluster. Only system settings are replicated across all servers.)

This operation has no parameters.

GetClusterMembers returns the servers in the cluster, with their names and IP addresses.

URL Format

GetClusterMembers has the following format:

```
http://<host>:<port>/GetClusterMembers
```

Operation Sequence

No operations must be executed before you can execute this operation.

Results

On success, *GetClusterMembers* returns an array, with one object for each system in the Live Stream cluster.

If there are none, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/GetClusterMembers
```

Typical Response

In this response, the Live Stream server has no systems in its cluster.


```
[
  {
    "Identifier": "00000000-0000-0000-0000-000000000000",
    "Name": "QA-VL-LIVE-2",
    "Description": null,
    "Details": [
      {
        "name": "MemberId",
        "value": "8391412250680132526"
      },
      {
        "name": "Peer URL",
        "value": "http://10.0.25.58:2380"
      },
      {
        "name": "Client URL",
        "value": "http://10.0.25.58:2379"
      }
    ]
  },
  {
    "Identifier": "00000000-0000-0000-0000-000000000000",
    "Name": "server0",
    "Description": null,
    "Details": [
      {
        "name": "MemberId",
        "value": "10276657743932975437"
```

```
      },
      {
        "name": "Peer URL",
        "value": "http://10.0.25.63:2380"
      },
      {
        "name": "Client URL",
        "value": "http://10.0.25.63:2379"
      }
    ]
  }
]
```

GetSystemSettings

This GET operation exports the entire Live Stream system configuration in XML format. This XML can be used for a variety of purposes, including archiving the settings for later use, importing directly into this or another system using the Import button in the Live Stream web app or via *ImportSystemSettings*.

This operation has no parameters.

After this operation has executed, you can save the XML for later use.

URL Format

GetSystemSettings has the following format:

```
http://<host>:<port>/GetSystemSettings
```

Operation Sequence

No other operations are required before you can execute this operation.

Results

Upon success, *GetSystemSettings* returns the system configuration of the target system in XML format.

Example

```
http://mylivestreamserver:18000/GetSystemSettings
```

Typical Response

In this response, this multi-machine XML is returned (shown here truncated for brevity).

```
<Domain>
  <Configuration>...</Configuration>
  <User>...</User>
  <Machine d1p1:identifier="f1c541f6-d003-41a0-b627-e6fa0add9c95"
d1p1:name="LS-SVR" d1p1:leftaligncheckboxes="false">
    <d1p1:Parameter>...</d1p1:Parameter>
```

```
...
    <Source d1p1:identifier="f7338552-a221-4853-a8cb-60de6371ae1b"
d1p1:name="LS-SVR - SDI Input 4" d1p1:description=""
d1p1:leftaligncheckboxes="false">
      <d1p1:Parameter type="boolean" identifier="a1962c94-8a40-
4d6f-a192-dec61e449bcb" name="10-Bit Video" description="Causes
the source to capture 10 bit video rather than 8 bit"
enabled="true" enabledinvariants="false" disableable="false"
browsable="true" optionseditable="false" row="0" column="0"
columnspan="1">...</d1p1:Parameter>
...
      <SourceType>sdi</SourceType>
      <Statistics>...</Statistics>
      <Tracks>...</Tracks>
      <TextTracks/>
    </Source>
  </Machine>
</Domain>
```

ImportSystemSettings

This POST operation loads a previously-exported Live Stream system configuration (in XML format) into the target Live Steam system. The configuration can be an archived version that was stored earlier or it can be a system-generated XML.

Note: You can only export system settings directly from the Stream Web app.

You can use this operation to load a complete, new configuration or you can import source, programs, encoder, or channel settings, or any combination of these. Excluded elements remain unchanged.

To delete existing settings before loading other configurations, use *ClearAllSettings*.

An orchestration system with archived system settings of a specific configuration or auto-generated configurations could set up a new/replacement component/instance. A redundancy feature could also spin up replacements for failed or unavailable servers/instances.

This operation has no parameters.

URL Format

`http://<host>:<port>/ImportSystemSettings`

Body Format

The configuration should not include the XML declaration element. It must include the Configuration element and the Domain element. All other elements are optional.

Identifiers for all objects are changed upon import, so attempting to import a single channel, for example, requires modifying the configuration to use the identifiers in the Persistence service for the Channel's packages, assigned sources, etc.

Operation Sequence

No other operations are required before you can execute this operation.

Results

Upon success, *ImportSystemSettings* re-configures the Live Stream server with the new configuration and returns this success message: "Import Successful!". Upon failure, an appropriate message is returned to identify the problem.

ClearAllSettings

This POST operation removes all components which are added during an import: Programs, Encoders, Packages, Channels, non-SDI sources, and any associated objects (Variants, Renditions, etc.). This operation should be executed when the system is idle—no streaming is underway.

Note: This operation is permanent—there is no undo. Consider using *GetSystemSettings* before executing this operation so that you have a backup copy.

This operation is intended to delete existing settings before loading other configurations using *ImportSystemSettings*.

This operation does not affect global settings, SDI sources, and non-imported objects including presets, API tokens, and social media authentications.

This operation has no parameters.

URL Format

```
http://<host>:<port>/ClearAllSettings
```

Request Body

No request body is required for this operation.

Operation Sequence

No other operations are required before you can execute this operation.

Results

Upon success, *ClearAllSettings* returns s success message: "Success". Upon failure (for example, a channel is running), an appropriate message is returned to identify the problem.

GetAuthentications

This GET operation returns a list of social media account records in Lightspeed Live Stream. This operation is a prerequisite to executing *GetAuthentication*.

This operation has no parameters.

Note: You can add social media account records to Lightspeed Live Stream using *GetNewAuthenticationDetails* and *CompleteAuthentication*.

URL Format

GetAuthentications has the following format:

```
http://<host>:<port>/GetAuthentications
```

Operation Sequence

No operations must be executed before you can execute this operation.

Results

Upon success, *GetAuthentications* returns a list of social media account records in the target system.

Example

```
http://10.9.9.9:18000/GetAuthentications
```

Typical Response

In this response, a social media account record for Facebook Live is returned.

```
{
  "Description":"Facebook Live",
  "Identifier":"14702b58-7b1b-47a5-bc08-887cb38f6f40",
  "Name":"Facebook Live"
}
```

GetAuthentication

This GET operation returns a specific social media account record from Live Stream. This operation is a prerequisite to *GetNewAuthenticationDetails*.

After this operation has executed, you can save the record in your program for later use. For example, you can use it for streaming to Facebook Live, etc.

URL Format

GetAuthentication has the following format:

```
http://<host>:<port>/GetAuthentication?identifier={AUTHENTICATON
GUID}
```

Operation Sequence

Execute the following operation to obtain the list of social media account records to obtain the required GUID for this operation:

GetAuthentications (social media account records)

Parameters

Parameter	Description
identifier	GUID; identifies a specific social media account record.

Results

Upon success, *GetAuthentication* returns an array of social media account records.

Example

```
http://mylivestreamserver:18000/
GetAuthentication?identifier=14722b58-7e1b-47b5-bc01-447ac18b6f44
```

Typical Response

In this response, the social media account record is returned.

```
{
  "Description": null,
  "Identifier": "14733b58-7a1b-47b5-bb01-803cb18a6f34",
  "Name": "authentication",
  "AuthenticationType": 1,
  "ExpiresIn": 5183988,
  "Username": "John Doe"
}
```

GetNewAuthenticationDetails

This GET operation returns an authentication code for authenticating a new user with the specified online video platform to create a new Social Media Account record in Live Stream, using *CompleteAuthentication*.

Creating a Social Media Account record via the API is a 3-step process:

1. First, execute *GetNewAuthenticationDetails*.
2. Authenticate the code, following the steps required for the platform, which is described in the Live Stream User Guide, in Authenticating Social Media Accounts.

3. Finally, execute *CompleteAuthentication* to add the record.

URL Format

GetNewAuthenticationDetails has the following format:

```
http://<host>:<port>/GetNewAuthenticationDetails?type={TYPE}
```

Operation Sequence

No other operations are required before you can execute this operation.

Parameters

Parameter	Description
type	Keyword that identifies the online video platform: facebooklive | youtubelive.

Results

Upon success, *GetNewAuthenticationDetails* returns an authentication code and details for the target platform. The process of authentication varies by platform.

Example

This example is requesting an authentication code for Facebook.

```
http://10.9.9.9:18000/GetNewAuthenticationDetails?type=facebook
```

Typical Response

In this response, this response is returned for use in Facebook. The URL is provided, along with the authentication code to use, and the Identifier for use in *CompleteAuthentication*.

```
{
  "Description": null,
  "Identifier": "74dd4cf9-809b-4067-a6ce-c796dafb015f",
  "Name": "New Authentication Details",
  "AuthenticationCode": "X8KEW43A",
  "AuthenticationType": 1,
  "Uri": "https://www.facebook.com/device"
}
```

CompleteAuthentication

This GET operation adds the specified Social Media Account record to Live Stream. This works in tandem with *GetNewAuthenticationDetails* and should be executed *after* you have authenticated your account with the target platform.

After this operation has executed, you can save the response for use when using the *Social Media Platform Channel Management Operations*.

URL Format

CompleteAuthentication has the following format:

```
http://<host>:<port>/CompleteAuthentication?
identifier={IDENTIFIER}&code={CODE}
```

Operation Sequence

Execute the following operation to obtain the required GUID and code for this operation:

GetNewAuthenticationDetails (Identifier GUID and Code)

Parameters

Parameter	Description
identifier	GUID; identifies a specific social media account record.
code	string; required for processing by the social media platform.

Results

Upon success, *CompleteAuthentication* returns the current Social Media Account record identified by the GUID.

Example

```
http://mylivestreamserver:18000/CompleteAuthentication?
identifier=74dd4cf9-809b-4067-a6ce-c796dafb015f&code=X8KEW43A
```

Typical Response

In this response, this Social Media Account record is returned.

```
{
  "Description":null,
  "Identifier":"71c2ece0-7c64-44c2-8669-622a54e8ff2f",
  "Name":"Authentication",
  "AuthenticationType":1,
  "ExpiresIn":5126097,
  "Username":"Art Anderson"
}
```

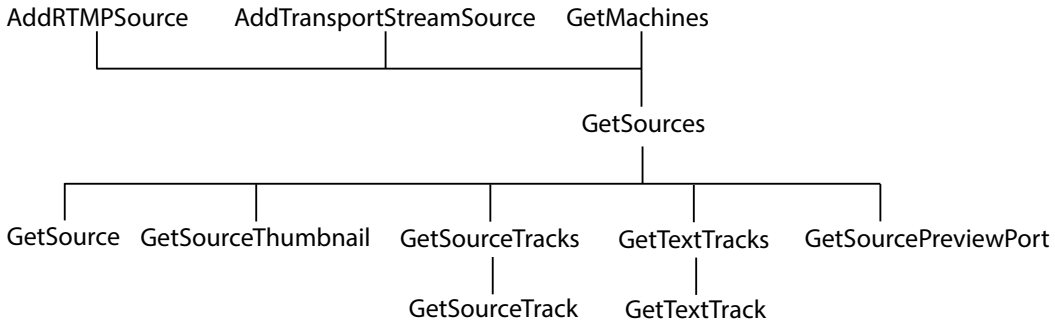
Sources Operations

Use the following operations to obtain a list of servers in a group, add sources, and operate on sources and their components.

- AddRtmpSource
- AddTransportStreamSource
- GetSources
- GetSource
- GetSourcePreviewPort
- GetSourceTrack
- GetTextTracks
- GetTextTrack
- GetSourceThumbnail

Note: All Sources operations start with `http://<host>:<port>/Sources/`.

In this category, operations are organized hierarchically, by machine GUID requirement. The operations to add sources do not require GUIDs. Other source operations require a machine GUID.



Note: To display help for Sources operations, execute `http://<host>:<port>/help` and open the Sources panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

AddRtmpSource

This POST operation adds a new RTMP source to the target Live Stream server.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

URL Format

`http://<host>:<port>/Sources/AddRtmpSource`

Body Format

```
{
  "sourceName":"SOURCE NAME",
  "streamName":"STREAM NAME"
}
```

Operation Sequence

No other operations are required before you can execute this operation.

Parameters

Parameter	Description
sourceName	String; the practical name of the source, which displays in the Sources panel and the Name field of the Configure Source window. For example: `"SourceName":"Boats Of Port Townsend"`
streamName	String; the practical name of the RTMP stream you want associated with this source. This name is specified along with the IP address in the RTMP stream generator program, and the two must match. For example: "`streamName":"BoatsOfPortTownsendStream".`

Results

Upon success, *AddRtmpSource* adds the specified source to the Live Stream server and returns a record with the assigned GUID, indicating the source that was added. If you add an RTMP source twice, the error "Failed to create RTMP source" is returned.

Example

```
http://10.9.9.9:18000/Sources/AddRtmpSource
```

```
{
  "sourceName":"Boats Of Port Townsend"
  "streamName":"BoatsOfPortTownsendStream"
}
```

Typical Response

A new RTMP source named *Boats Of Port Townsend* was added to this server. The server generated a GUID for the new source, and also returns the name you supplied.

```
{
  "Description": "",
  "Identifier": "e5b1f7eb-1ca0-429e-b1e8-4d2bacf4900b",
  "Name": "Boats Of Port Townsend"
}
```

AddTransportStreamSource

This POST operation adds a new Transport Stream source to the target Live Stream server, with the specified name and settings.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

URL Format

```
http://<host>:<port>/Sources/AddTransportStreamSource
```

Body Format

```
{
  "LocalIP":"IP ADDRESS (CIDR FORMAT)",
  "MulticastIP":"MULTICAST IP ADDRESS",
  "Port":PORT NUMBER,
  "ProgramNumber":PROGRAM NUMBER,
  "SourceFilterIP":"SOURCE FILTER IP ADDRESS",
  "SourceName":"SOURCE NAME"
}
```

AddTransportStreamSource adds the specified source to the Live Stream server and returns a component indicating the source that was added.

Operation Sequence

No other operations are required before you can execute this operation.

Parameters

Parameter	Description
sourceName	String; practical name of the source, displayed in the Name field of the Configure Source window. For example:`"sourceName":"Hiking Wind Ridge"`
multicastIP (optional)	IP address string; specifies the optional multicast group IP address being used by the Transport Stream generator program to broadcast.
localIP	IP address string in CIDR format; specifies the IP address of the NIC card in the server where you are listening for the Transport Stream. For example: `10.0.56.0/24`.

Parameter	Description
port	INT; specifies the port number that the Transport Stream generator program is broadcasting this Transport Stream on.
programNumber	INT; specifies the program number in the Transport Stream that you are receiving.
sourceFilterIP (optional)	IP address string; specifies the IP address of a Transport Stream when originating from a system that is broadcasting using multicast.

Results

Upon success, *AddTransportStreamSource* adds the specified source to the Live Stream server and returns a component, indicating the source that was added.

Example

```
http://mylivestreamserver:18000/Sources/AddTransportStreamSource
```

```
{
  "sourceName":"Hiking Wind Ridge",
  "multiCastIP":"239.1.2.3",
  "localIP":"10.0.25.0/24",
  "port":1234,
  "programNumber":1
}
```

Typical Response

In this response, a new Transport Stream source named *Hiking Wind Ridge* was added to this Live Stream server. The Live Stream server generated a GUID for the new source, and also returns the name you supplied.

```
{
  "Description": "",
  "Identifier": "9278c2cd-cc10-4df5-8cb4-018451ef6793",
  "Name": "Hiking Wind Ridge"
}
```

GetSources

This GET operation identifies all of the video sources that are available on the target Live Stream server, and return them in a list for further use.

URL Format

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute this operation to obtain the machine GUID required to execute this operation.

GetMachines (machine GUID)

Parameters

Parameter	Description
machine	GUID; identifies a specific Live Stream server.

Results

On success, *GetSources* returns a set of records; one for each source in the target Live Stream server.

Example

```
http://10.9.9.9:18000/GetSources
?machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server—the four default SDI sources, plus a Transport Stream source—are listed along with their Identifier and Name values, which you can use to query and use them.


```
[
  {
    "Description":null,
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"QA-VL-LIVE-9 - SDI Input 3"
  },
  {
    "Description":null,
    "Identifier":"7bb71ac8-f5ba-496f-826d-558b38721ae2",
    "Name":"QA-VL-LIVE-9 - SDI Input 2"
  },
  {
    "Description":null,
    "Identifier":"8a7bc656-6704-41a2-8161-f4d0d5a5f8de",
    "Name":"Test1"
  },
  {
    "Description":null,
    "Identifier":"b2432c41-5b8f-4fc2-8980-fde99709bef9",
    "Name":"QA-VL-LIVE-9 - SDI Input 1"
  },
  {
    "Description":null,
    "Identifier":"edec5e10-177c-4abb-a274-c9fb82393412",
    "Name":"QA-VL-LIVE-9 - SDI Input 4"
  },
```

```
  {
    "Description":null,
    "Identifier":"9278c2cd-cc10-4df5-8cb4-018451ef6793",
    "Name":"Hiking Wind Ridge"
  }
]
```

GetSource

This GET operation returns the details of the specified source on the target Live Stream server.

URL Format

GetSource has the following format:

`http://<host>:<port>/Sources/GetSource?identifier={SOURCE GUID}`

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetSources (source GUID)

Parameters

Parameter	Description
identifier	GUID; identifies this source.

Results

On success, *GetSource* returns a record with details about the target source. The record has these key/value pairs:

- Name/Value Pair Details—Process, Resolution, Frame Rate, Bit Depth, Audio, Captions, Deck State, Deck Mode, and Time Code
- Text Tracks—One Brief per text track; Identifier and name
- Tracks—One Brief per track; Identifier and name
- Type—SDI, Slate, FileLoop, etc.

If no source exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Sources/GetSource
?identifier=9278c2cd-cc10-4df5-8cb4-018451ef6793
```

Typical Response

This is a typical response from *GetSource*; a record with key/pair values defining the target source, plus the tracks in the source.

Note: The firmware version and process number are not displayed in the web app unless it is operating in Advanced mode.

```
{
  "Description":null,
  "Identifier":"9278c2cd-cc10-4df5-8cb4-018451ef6793",
  "Name":"Test2",
  "Details":[
    {
      "Name":"CPU Usage",
      "Value":"1"
    },
    {
      "Name":"Memory",
      "Value":"444"
    },
    {
      "Name":"Process",
      "Value":"38908"
    },
    {
      "Name":"Resolution",
      "Value":"1920x1080i"
    },
    {
      "Name":"Frame Rate",
      "Value":"29.97"
    },
    {
      "Name":"Bit Depth",
      "Value":"10-Bit"
    },
    {
      "Name":"Audio",
      "Value":"2"
    },
    {
      "Name":"Captions",
      "Value":"No"
    },
    {
      "Name":"Time Code (Free Run)",
      "Value":"00:00:53;01"
    }
  ],
  "TextTracks":{
    "Briefs":[
    ]
  },
```

```
  "Tracks":{
    "Briefs":[
      {
        "Description":null,
        "Identifier":"ca547d60-3963-4127-8045-640b31aec313",
        "Name":"Stereo 1"
      }
    ]
  },
  "Type":4
}
```


GetSourcePreviewPort

This GET operation returns the port number which is streaming JPEG proxy preview frames, referenced by the source GUID.

This operation has no parameters.

URL Format

GetSourcePreviewPort has the following format:

```
http://<host>:<port>/Sources/
GetSourcePreviewPort?identifier={SOURCE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetSources (source GUID)

Parameters

Parameter	Description
identifier	GUID; identifies this source.

Results

On success, *GetSourcePreviewPort* returns the requested port number.

Example

```
http://myliveserver:18000/Sources/GetSourcePreviewPort?
identifier=ca547d60-3963-4127-8045-640b31aec313
```

Typical Response

In this response, the operation returns the port for this source: 58249

GetSourceTrack

This GET operation returns details about a specific audio track.

URL Format

GetSourceTrack has the following format:

```
http://<host>:<port>/Sources/GetSourceTrack
?identifier={TRACK GUID}
```

In addition to Identifier and Name, the track record has these relevant elements:

- ChannelConfiguration—the string identifying the track
- Details—a set of key/value pairs, relevant to the type of audio track.

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetSources (source GUID) *GetSource* (track GUID)
or
GetSources (source GUID) > *GetSourcePreviewPort* (track GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific audio track.

Results

On success, *GetSourceTrack* returns a record with details of key/value pairs about the target track.

If no SourceTrack exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Sources/GetSourceTrack
?identifier=2da934d3-54f9-4880-8e85-b28851355d61
```

Typical Response

In this response, the target track is Stereo 1. Each track has a set of key/value pairs.

```
{
  "Description":null,
  "Identifier":"2da934d3-54f9-4880-8e85-b28851355d61",
  "Name":"Stereo 1",
  "ChannelConfiguration":2,
  "Details":[
    {
```

```
      "Name":"Language",
      "Value":"English"
    },
    {
      "Name":"Audio Service",
      "Value":"Primary"
    },
    {
      "Name":"Default",
      "Value":"False"
    },
    {
      "Name":"Left Channel",
    "Value":"1"
    },
    {
      "Name":"Right Channel",
      "Value":"2"
    }
  ]
}
```


GetTextTracks

This GET operation returns a list of 608/708 text (caption) tracks that are present in the target source.

URL Format

GetTextTracks has the following format:

`http://<host>:<port>/Sources/GetTextTracks?source={SOURCE GUID}`

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetSources (source GUID)

Parameters

Parameter	Description
source	GUID; identifies a specific source.

Results

On success, *GetTextTracks* returns an array; one record for each text track in the source.

If no text tracks are present, the array is returned empty ([]).

Example

```
http://mylivestreamserver:18000/Sources/GetTextTracks
?source=2da934d3-54f9-4880-8e85-b28851355d61
```

Typical Response

In this response, the target source has two text tracks. Each has a GUID in the Identifier, plus a Name.


```
[
  {
    "Description":null,
    "Identifier": "ef1752a6-f4fa-49e3-a779-54373f31cb60",
    "Name": "C608English"
  }
  {
    "Description":null,
    "Identifier": "ef1752a6-f4fa-49e3-a779-54373f31cb60",
    "Name": "C708English"
  }
]
```


GetTextTrack

This GET operation returns details about a specific text (608 or 708 caption) track.

URL Format

GetTextTrack has the following format:

```
http://<host>:<port>/Sources/GetTextTrack?identifier={TRACK GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetSources (source GUID) *GetSource* (track GUID)
or
GetSources (source GUID) > *GetTextTracks* (track GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific text track.

Results

On success, *GetTextTrack* returns a record, with details about the target track.

If no text track exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Sources/GetTextTrack
?identifier=6f79c567-d883-4548-81df-1faeeefe3bf6
```

Typical Response

In this response, the target track is C608English.


```
{
  "Description":null,
  "Identifier":"6f79c567-d883-4548-81df-1faeeefe3bf6",
  "Name":"C608English",
  "Details":[
    {
      "Name":"Language",
      "Value":"English"
    },
    {
      "Name":"CC Index",
      "Value":"1"
    }
  ],
  "TextConfiguration":0
}
```


GetSourceThumbnail

This GET operation returns a thumbnail from the target source. The operation returns a JPEG, which you can render or save as a file.

The size of the thumbnail returned can't be changed.

URL Format

GetSourceThumbnail has the following format:

```
http://<host>:<port>/Sources/GetSourceThumbnail
?source={SOURCE GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetSources (source GUID)

Parameters

Parameter	Description
source	GUID; identifies a specific source.

Results

On success, *GetSourceThumbnail* returns a JPEG. If the source you are targeting does not have a thumbnail, an HTTP 404 error is returned.

Example

```
http://mylivestreamserver:18000/Sources/GetSourceThumbnail
?source=27602854-35a6-4f0c-827c-ebd7ac5d40a9
```

Programs Operations

A program defines what media should be encoded. Programs are comprised of renditions, which are comprised of segments. Segments are comprised of materials.

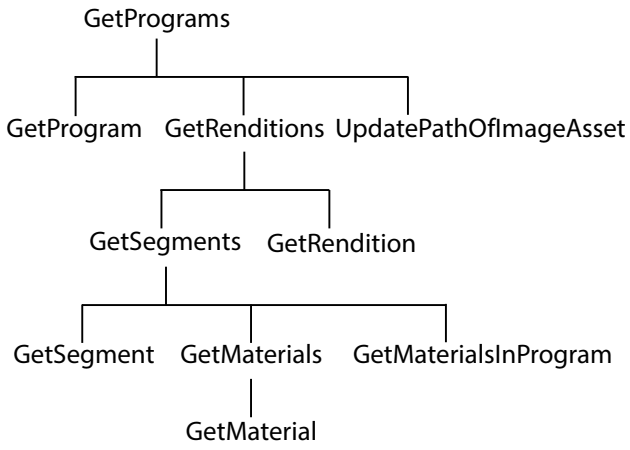
Note: Segments—in the context of Live Stream—are the collection of material that should be encoded at any given time in a rendition. This is not a segment in a package which refers to an atomic unit of ABR content which players might play as a result of dynamically-changing bandwidth.

These operations enable you to identify all of the programs in a Live Stream server and target their components: renditions, segments, and materials, successively.

- GetPrograms
- GetProgram
- GetRenditions
- GetRendition
- GetSegments
- GetSegment
- GetMaterials
- GetMaterialsInProgram
- UpdatePathOfImageAsset

Note: All Programs operations start with `http://<host>:<port>/Programs/`.

The operations are organized hierarchically, by program GUID requirement.



Note: To display help for Programs operations, execute `http://<host>:<port>/help` and open the Programs panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

GetPrograms

This GET operation returns a list of all of the programs that have been added to the target Live Stream system.

This operation has no parameters.

URL Format

GetPrograms has the following format:

`http://<host>:<port>/Programs/GetPrograms`

Operation Sequence

No operations must be executed before you can execute this operation.

Results

On success, *GetPrograms* returns an array with a record for each program in the system.

If no Programs are present, the array is returned empty ([]).

Example

`http://10.9.9.9:18000/Programs/GetPrograms`

Typical Response

In this response, the array of two records indicates there are two programs on this Live Stream server. Each program has a GUID in the Identifier, and a Name.


```
[
  {
    "Description":null,
    "Identifier":"8bd029bb-c5e3-4c20-af98-48df76f59380",
    "Name":"Trigger Program"
  },
  {
    "Description":null,
    "Identifier":"92aa9eb9-a1fb-4aab-b920-c3259895edb4",
    "Name":"Basic Program"
  }
]
```


GetProgram

This GET operation returns detailed information about a specific program.

URL Format

GetProgram has the following format:

```
http://<host>:<port>/Programs/GetProgram?identifier={program GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetPrograms (program GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific program.

Results

On success, *GetProgram* returns a record with details about the program, plus an array of records; one for each rendition in the target program.

If no program exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Programs/GetProgram
?identifier=8bd029bb-c5e3-4c20-af98-48df76f59380
```

Typical Response

In this response, the target program has one Rendition. Each rendition (in the Briefs array) has a GUID Identifier you can use to query it.


```
{
  "Description":null,
  "Identifier":"8bd029bb-c5e3-4c20-af98-48df76f59380",
  "Name":"Trigger Program",
  "Renditions":{
    "Briefs":[
      {
        "Description":null,
        "Identifier":"17b2c3f9-b4ef-401f-bc6d-f01ab6311f84",
        "Name":"Trigger Rendition"
      }
    ]
  },
  "Resolution":"1920 x 1080"
}
```


GetRenditions

This GET operation returns a list of Renditions that comprise a specific program.

URL Format

GetRenditions has the following format:

`http://<host>:<port>/Programs/GetRenditions?program={PROGRAM GUID}`

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetPrograms (program GUID)

Parameters

Parameter	Description
program	GUID; identifies a specific program.

Results

On success, *GetRenditions* returns an array, with one record for each rendition of the program.

If no Renditions are present, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/programs/GetRenditions
?program=7fbb4998-f82a-44da-8d6c-47344b47c10b
```

Typical Response

In this response, there is one rendition, identified by Identifier and Name, which is presented in the web app.


```
[
  {
    "Description":null,
    "Identifier":"17b2c3f9-b4ef-401f-bc6d-f01ab6311f84",
    "Name":"Trigger Rendition"
  }
]
```


GetRendition

This GET operation returns details about a specific rendition.

URL Format

GetRendition has the following format:

```
http://<host>:<port>/Programs/GetRendition
?identifier={RENDITION GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPrograms (program GUID) > GetProgram (rendition GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific rendition.

Results

On success, *GetRendition* returns a record, with details about the target rendition and each of its segments.

If no rendition exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Programs/GetRendition
?identifier=17b2c3f9-b4ef-401f-bc6d-f01ab6311f84
```

Typical Response

In this response, the rendition is English Stereo. It has several settings in the Details, plus a list of Segments and their GUIDs and Name. The rendition Type is also listed.


```
{
  "Description":null,
  "Identifier":"17b2c3f9-b4ef-401f-bc6d-f01ab6311f84",
  "Name":"Trigger Rendition",
  "Details":[
    {
      "Name":"Channel Configuration",
      "Value":"Stereo"
    },
    {
      "Name":"Audio Service",
      "Value":"Primary"
    },
    {
      "Name":"Language",
      "Value":"English"
    }
  ],
  "Segments":{
    "Briefs":[
      {
        "Description":null,
        "Identifier":"6ef8de15-cd72-4184-b36d-dfb9a4cf87e1",
        "Name":"Trigger Backup Segment"
      },
      {
        "Description":null,
        "Identifier":"82dfa6e9-34b0-40bb-a047-d54ae4f32faa",
        "Name":"Trigger Base Segment"
      }
    ]
  },
  "Tracks":{
    "Briefs":[
      {
        "Description":"This track is from the source audio tracks",
        "Identifier":"749d9558-5606-4295-b88a-e35bb0f25e3a",
        "Name":"Rendition Stereo Track"
      },
      {
        "Description":"This track is from the source audio tracks",
        "Identifier":"4385298a-fe7c-40d8-959b-a798ab305e82",
        "Name":"Rendition audio track 4"
      },
      {
        "Description":"This track is from the source audio tracks",
```

```
        "Identifier":"4f889818-2cbc-4403-86d8-741c7b8c1c58",
        "Name":"Rendition audio track 3"
      },
      {
        "Description":"This track is from the source audio tracks",
        "Identifier":"b0e1121d-6275-4acd-bcb5-42326647ef13",
        "Name":"Rendition Stereo Track"
      },
      {
        "Description":"This track is from the source audio tracks",
        "Identifier":"cda1680c-4d58-4938-bd56-9dc88b391ad4",
        "Name":"Rendition audio track 2"
      },
      {
        "Description":"This track is from the source audio tracks",
        "Identifier":"efc212dc-78eb-4a7b-aae8-444067466754",
        "Name":"Rendition Mono Track"
      }
    ]
  },
  "Type":0
}
```


GetSegments

This GET operation returns a list of Segments that comprise a specific rendition of a program.

URL Format

GetSegments has the following format:

```
http://<host>:<port>/Programs/GetSegments?rendition={RENDITION GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPrograms (program GUID) > GetProgram (rendition GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID)

Parameters

Parameter	Description
rendition	GUID; identifies a specific rendition.

Results

On success, *GetSegments* returns an array; one record for each segment in the rendition. If no Segments are present, the array is returned empty ([]). If the rendition you specify does not exist, the Live Stream server returns an error.

Example

```
http://10.9.9.9:18000/Programs/GetSegments
?rendition=1e32e2f1-3702-4d83-8460-e9d2231db3c5
```

Typical Response

In this response, the rendition has two segments, identified by Name and by GUID.

```
[
  {
    "Description":null,
    "Identifier":"6ef8de15-cd72-4184-b36d-dfb9a4cf87e1",
    "Name":"Trigger Backup Segment"
  },
  {
    "Description":null,
    "Identifier":"82dfa6e9-34b0-40bb-a047-d54ae4f32faa",
    "Name":"Trigger Base Segment"
  }
]
```

GetSegment

This GET operation returns details about a specific segment.

URL Format

GetSegment has the following format:

```
http://<host>:<port>/Programs/GetSegment?identifier={SEGMENT GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPrograms (program GUID) > GetProgram (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetRendition* (segment GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific segment.

Results

On success, *GetSegment* returns a record, with details about the target segment and all of its Material.

If no segment exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Programs/GetSegment
?identifier=6ef8de15-cd72-4184-b36d-dfb9a4cf87e1
```

Typical Response

In this response, the target segment is Trigger Backup Segment. It has a StartTrigger and an EndTrigger, and Material consisting of one asset.


```
{
  "Description":null,
  "Identifier":"6ef8de15-cd72-4184-b36d-dfb9a4cf87e1",
  "Name":"Trigger Backup Segment",
  "EndTrigger":{
    "Description":"DurationIdentifier",
    "Identifier":"31afe9dd-9130-42db-b1af-7e0e7e19a1a7",
    "Name":"Duration",
    "Details":[
      {
        "Name":"Duration",
        "Value":"00:02:00"
      }
    ]
  },
  "Materials":{
    "Briefs":[
      {
        "Description":"A placeholder for a source.",
        "Identifier":"e816c0fd-d4df-4f41-a253-8d2b4ce829da",
        "Name":"Backup Source Placeholder"
      }
    ]
  },
  "Priority":0,
  "StartTrigger":{
    "Description":"TimeOfDayIdentifier",
    "Identifier":"5cb3f0dc-d50e-495c-9197-19a72d4e981a",
    "Name":"Time of Day",
    "Details":[
      {
        "Name":"Time",
```

```
        "Value":"12:00:00"
      },
      {
        "Name":"Repeat Interval",
        "Value":"00:04:00"
      }
    ]
  }
}
```


GetMaterials

This GET operation returns a list of material that comprises a specific segment.

URL Format

GetMaterials has the following format:

```
http://<host>:<port>/Programs/GetMaterials?segment={SEGMENT GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPrograms (program GUID) > GetProgram (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetRendition* (segment GUID)

Parameters

Parameter	Description
segment	GUID; identifies a specific segment.

Results

On success, *GetMaterials* returns an array, with one record per material in the segment. If the segment does not contain any material, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/Sources/GetMaterials
?segment=1f3a2a14-4ea6-411d-83a7-59694f4eed7e
```

Typical Response

In this response, the target segment has three material items.


```
[
  {
    "Description":"A placeholder for a source.",
    "Identifier":"262f8038-62bd-448d-b0c3-fe98177557fa",
    "Name":"Basic Source Placeholder"
  },
  {
    "Description":"An asset with both a static visual component.",
    "Identifier":"6ee9b669-2e0f-4810-9fff-290a25601551",
    "Name":"Yoshi"
  },
  {
    "Description":"Asset with both a sound and visual component.",
    "Identifier":"a8b7d795-b309-41c0-83fa-ce5d47ed81aa",
    "Name":"Watchmen"
  }
]
```


GetMaterialsInProgram

This GET operation returns a list of material in the target program. GetMaterialsInProgram is often used in conjunction with UpdatePathOfImageAsset, where a specific material GUID is required, so that you can provide a new image.

URL Format

GetMaterialsInProgram has the following format:

```
http://<host>:<port>/Programs/
GetMaterialsInProgram?program={PROGRAM GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPrograms (program GUID)

Parameters

Parameter	Description
program	GUID; identifies a specific program.

Results

On success, *GetMaterialsInProgram* returns an array, with one record per material in the program. If the program does not contain any material, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/Sources/GetMaterialsInProgram
?program=7fbb4998-f82a-44da-8d6c-47344b47c10b
```

Typical Response

In this response, the target program has three material items.


```
[
  {
    "Description":"A placeholder for a source.",
    "Identifier":"262f8038-62bd-448d-b0c3-fe98177557fa",
    "Name":"Basic Source Placeholder"
  },
  {
    "Description":"An asset with both a static visual component.",
    "Identifier":"6ee9b669-2e0f-4810-9fff-290a25601551",
    "Name":"Yoshi"
  },
  {
    "Description":"Asset with both a sound and visual component.",
    "Identifier":"a8b7d795-b309-41c0-83fa-ce5d47ed81aa",
    "Name":"Watchmen"
  }
]
```


UpdatePathOfImageAsset

The *UpdatePathOfImageAsset* POST operation specifies a new image file to use as a static overlay. You can only change the image when the program is not being used in an active channel.

URL Format

```
http://<host>:<port>/UpdatePathOfImageAsset
```

Body Format

```
{
  "program":"PROGRAM GUID",
  "imageAsset":IMAGE OVERLAY GUID,
  "path":FULLY QUALIFIED PATH AND FILE
}
```

Operation Sequence

Execute the following operation to obtain the required GUIDs for this operation:

GetPrograms (program GUID)

GetMaterialsInProgram (Image GUID)

GetChannels (Channel GUID)

StopChannel (Channel GUID) if the channel is currently active

UpdatePathOfImageAsset

StartChannel (Channel GUID)

Required Post Body

program	GUID; string that identifies a specific program.
`imageAsset`	GUID; string that identifies a specific image asset in the program.
path	String; fully-qualified path to the image to be used as an overlay. For example: `source=01:03:15:00@29.97`

Results

Upon success, *UpdatePathOfImageAsset* returns the string "`true`". If there is a failure, it returns a 404, or a 500 when the Stream service is down.

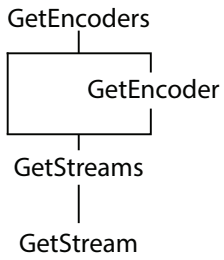
Encoders Operations

The purpose of the Encoders group of operations is to obtain details about specific encoders that have been implemented in the target Live Stream server, and the streams the comprise a given encoder.

- GetEncoders
- GetEncoder
- GetStreams
- GetStream

Note: All Encoders operations start with `http://<host>:<port>/Encoders/`.

The operations are organized hierarchically, by GUID requirement. The *GetEncoders* operation does not require a GUID.



Note: To display help for Encoders operations, execute `http://<host>:<port>/help` and open the Encoders panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

GetEncoders

This GET operation returns a list of encoders that have been created on the target Live Stream system. This operation has no parameters.

URL Format

GetEncoders has the following format:

`http://<host>:<port>/Encoders/GetEncoders`

Operation Sequence

No operations must be executed before you can execute this operation.

Results

On success, *GetEncoders* returns an array, which has one record for each encoder on the Live Stream server. If no encoders are present, the array is returned empty ([]).

Example

`http://10.9.9.9:18000/Encoders/GetEncoders`

Typical Response

In this response, the target Live Stream server has four encoders; each identified by Name and Description, and GUID in the Identifier which you can use to target the encoder for further utilization.


```
[
  {
    "Description":"HEVC",
    "Identifier":"0c54e3a9-dd89-4d43-8dc6-a009ab231ff0",
    "Name":"HEVC"
  },
  {
    "Description":"AVC",
    "Identifier":"0f25fcb4-c412-4761-a96e-9eb5480a7013",
    "Name":"AVC-LIVE"
  },
  {
    "Description":"AVC",
    "Identifier":"1d0c1ae0-10d7-47b4-8f67-6656294df67f",
    "Name":"AVC"
  },
```

```
  {
    "Description":"AAC",
    "Identifier":"36c5006c-f04b-4e3a-9ca3-8ebbfb1cbc8f",
    "Name":"AAC"
  }
]
```


GetEncoder

This GET operation returns details about a specific encoder.

URL Format

GetEncoder has the following format:

```
http://<host>:<port>/Encoders/GetEncoder?identifier={ENCODER GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetEncoders (encoder GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific encoder.

Results

On success, *GetEncoder* returns a record with details about the target encoder and its streams.

If no encoder exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Encoders/GetEncoder
?identifier=0c54e3a9-dd89-4d43-8dc6-a009ab231ff0
```

Typical Response

In this response, the target encoder is identified by its Description, Identifier, and Name. Each stream in the encoder is enumerated—also with a Description, Identifier, and Name.


```
{
  "Description":"HEVC",
  "Identifier":"0c54e3a9-dd89-4d43-8dc6-a009ab231ff0",
  "Name":"HEVC",
  "Details":[
    {
```

```
      "Name":"GOP Duration",
      "Value":"3"
    }
  ],
  "Streams":{
    "Briefs":[
      {
        "Description":null,
        "Identifier":"0ab406ba-3c9d-47b1-8e28-ae94fd17db2a",
        "Name":"UHD HEVC"
      },
      {
        "Description":null,
        "Identifier":"4d75546c-c146-4818-b4f5-f00361190bfa",
        "Name":"1080p-HEVC"
      },
      {
        "Description":null,
        "Identifier":"cbccf0c6-5d12-4f6b-af66-2b53451625cb",
        "Name":"540p-HEVC"
      }
    ]
  }
}
```


GetStreams

This GET operation returns a list of streams that have been added to a specific encoder.

URL Format

GetStreams has the following format:

`http://<host>:<port>/Encoders/GetStreams?encoder={ENCODER GUID}`

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetEncoders (encoder GUID)

Parameters

Parameter	Description
encoder	GUID; identifies a specific encoder.

Results

On success, *GetStreams* returns an array with one record for each stream in the encoder.

If no Streams are present, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/Encoders/GetStreams
?encoder=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

In this response, the target encoder has two streams: one 720p, another for 1080p. Each is identified by Name and by an Identifier GUID. You can use the GUID to target a specific stream for further use.


```
[
  {
    "Description":null,
    "Identifier":"00befa87-3940-4ae0-8652-a66d0e46c0ac",
    "Name":"720p-RTMP"
  },
  {
    "Description":null,
    "Identifier":"e4aa2e66-04c9-4705-aeab-6490c8cec8fc",
    "Name":"1080p-YTL"
  }
]
```


GetStream

This GET operation returns details about a specific stream.

URL Format

GetStream has the following format:

```
http://<host>:<port>/Encoders/GetStream?identifier={STREAM GUID}
```

Operation Sequence

Execute one of the following operations to obtain the required GUID for this operation:

GetStreams or *GetEncoder* (stream GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific stream.

Results

On success, *GetStream* returns a record, with details about the target stream.

If no stream exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Encoders/GetStream
?identifier=00befa87-3940-4ae0-8652-a66d0e46c0ac
```

Typical Response

In this response, the target stream is identified by the Identifier and Name, plus its Details. The key/value pairs provide a set of parameters appropriate for this type of material.


```
{
  "Description":null,
  "Identifier":"00befa87-3940-4ae0-8652-a66d0e46c0ac",
  "Name":"720p-RTMP",
  "Details":[
    {
      "Name":"Width",
      "Value":"1280"
    },
    {
      "Name":"Height",
      "Value":"720"
    },
    {
      "Name":"Bitrate",
      "Value":"3000"
    },
    {
      "Name":"Framerate",
      "Value":"30"
    },
    {
      "Name":"Profile",
      "Value":"High"
    },
    {
      "Name":"GOP length",
      "Value":"90"
    },
    {
      "Name":"Display Aspect Ratio",
      "Value":"16:9"
    },
    {
      "Name":"Burn-in Timecode",
      "Value":"True"
    },
    {
      "Name":"CBR mode",
      "Value":"True"
    },
    {
      "Name":"Low latency",
      "Value":"False"
    },
```

```
    {
      "Name":"IDR Splices",
      "Value":"False"
    },
    {
      "Name":"Compressor",
      "Value":"x264"
    },
    {
      "Name":"Command Line Options",
      "Value":"--bframes 0"
    },
    {
      "Name":"Preset",
      "Value":"veryfast"
    },
    {
      "Name":"Video Processor",
      "Value":"Default"
    }
  ],
  "Essence":1
}
```

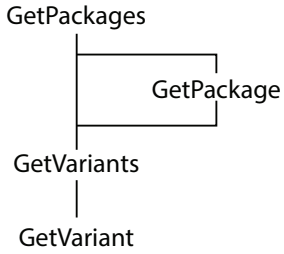

Packages Operations

The Packages operations enable you to query packages and their variants, that have been created on the system (for example, Apple HLS or RTMP).

- GetPackages
- GetPackage
- GetVariants
- GetVariant

Note: All Packages operations start with `http://<host>:<port>/Packages/`.

In this category, operations are organized hierarchically, by GUID requirement. The *GetPackages* operation does not require a GUID.



Note: To display help for Packages operations, execute `http://<host>:<port>/help` and open the Packages panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

GetPackages

This GET operation returns a list of packages that have been created on the target Live Stream system.

This operation has no parameters.

URL Format

GetPackages has the following format:

`http://<host>:<port>/Packages/GetPackages`

Operation Sequence

No operations must be executed before you can execute this operation.

Results

On success, *GetPackages* returns an array, which has one record for each package on the Live Stream server.

If no Packages are present, the array is returned empty ([]).

Example

`http://10.9.9.9:18000/Packages/GetPackages`

Typical Response

In this response, the target Live Stream server has eight packages. Each is identified by a Description, an Identifier GUID, and Name. You can use the GUID to target a specific package for further use.


```
[
  {
    "Description":"DASH",
    "Identifier":"026bb4c3-dd10-4f45-9bdf-66388c20281d",
    "Name":"DASH-SCTE"
  },
  {
    "Description":"RTMP",
    "Identifier":"0b21da42-f881-4e1b-89a4-0413942662f2",
    "Name":"RTMP-Generic"
  },
  {
```

```
    "Description":"RTMP",
    "Identifier":"0fff4aed-d89e-44a6-9674-dd046b276c98",
    "Name":"TS-Akamai-RTMP"
  },
  {
    "Description":"YouTube Live",
    "Identifier":"185b4a7b-126e-4c36-bab7-2626244e9a2f",
    "Name":"YTL-Test"
  },
  {
    "Description":"Apple HLS",
    "Identifier":"19df3b43-470a-4451-b243-51997fe81a93",
    "Name":"HLS AVC Test"
  },
  {
    "Description":"CMAF",
    "Identifier":"2ba22df7-9724-41ff-adc5-00a1dfe4b19f",
    "Name":"CMAF1"
  },
  {
    "Description":"MP4",
    "Identifier":"401310ba-42f2-4ed6-ac8d-2b68b0cbe4ee",
    "Name":"MP4-HEVC"
  },
  {
    "Description":"Transport Stream",
    "Identifier":"e40fcbee-6a2e-4567-9a48-78e32af0d0a8",
    "Name":"TS Package Test"
  }
]
```


GetPackage

This GET operation returns details about a specific package.

URL Format

GetPackage has the following format:

```
http://<host>:<port>/Packages/GetPackage?identifier={PACKAGE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetPackages (package GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific package.

Results

On success, *GetPackage* returns a record, with details about the target package and its variants.

If no package exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Packages/GetPackage
?identifier=026bb4c3-dd10-4f45-9bdf-66388c20281d
```

Typical Response

In this response, the target package is identified by its Description, Identifier, and Name. The package also lists all of its settings, as appropriate by package type, followed by the program and type. Variants in this package, if any, are also listed with their identification and details.


```
{
  "Description":"DASH",
  "Identifier":"026bb4c3-dd10-4f45-9bdf-66388c20281d",
  "Name":"DASH-SCTE",
  "Details":[
    {
      "Name":"Segment Duration",
      "Value":"9"
    },
    {
      "Name":"Playlist Name",
      "Value":"manifest"
    },
    {
      "Name":"Directory Rollover",
      "Value":"False"
    },
    {
      "Name":"UTC Timestamps",
      "Value":"False"
    },
    {
      "Name":"SCTE-35 in Manifest",
      "Value":"True"
    },
    {
      "Name":"SCTE-35 in fMP4",
      "Value":"True"
    },
    {
      "Name":"Playlist Type",
      "Value":"Rolling"
    },
    {
      "Name":"Elements",
      "Value":"25"
```

```
    },
    {
      "Name":"Encryption",
      "Value":"Unencrypted"
    }
  ],
  "Program":"92aa9eb9-a1fb-4aab-b920-c3259895edb4",
  "Type":2,
  "Variants":{
    "Briefs":[
      {
        "Description":null,
        "Identifier":"5003fae7-0bfa-42f9-b7a4-818ede966c12",
        "Name":"540p-X-AVC"
      },
      {
        "Description":null,
        "Identifier":"f4232a60-d80b-48ab-a5c5-eaeb9619e1a8",
        "Name":"720p-X-AVC"
      }
    ]
  }
}
```


GetVariants

This GET operation returns a list of variants that comprise the target package.

URL Format

GetVariants has the following format:

`http://<host>:<port>/Packages/GetVariants?package={PACKAGE GUID}`

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetPackages (package GUID)

Parameters

Parameter	Description
package	GUID; identifies a specific package.

Results

On success, *GetVariants* returns an array, which has one record for each variant in the target package.

If no Variants are present, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/Packages/GetVariants
?package=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

In this response, the target package has two variants; each with a Description, Identifier, and Name, which you can use to target the variant for further utilization.


```
[
  {
    "Description":null,
    "Identifier":"5003fae7-0bfa-42f9-b7a4-818ede966c12",
    "Name":"540p-X-AVC"
  },
  {
    "Description":null,
    "Identifier":"f4232a60-d80b-48ab-a5c5-eaeb9619e1a8",
    "Name":"720p-X-AVC"
  }
]
```


GetVariant

This GET operation returns details about a specific variant.

URL Format

GetVariant has the following format:

```
http://<host>:<port>/Packages/GetVariant?identifier={VARIANT GUID}
```

Operation Sequence

Execute one of the following operations to obtain the required GUID for this operation:

GetPackages (package GUID) > *GetVariants* (variant GUID)
or
GetPackages (package GUID) > *GetPackage* (variant GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific variant.

Results

On success, *GetVariant* returns a record, with details about the target variant.

If no variant exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Packages/GetVariant
?identifier=f4232a60-d80b-48ab-a5c5-eaeb9619e1a8
```

Typical Response

In this response, the target package is identified by its Description, Identifier, and a Name. Specific settings are presented in key/value pairs, plus a rendition and stream.


```
{
  "Description":null,
  "Identifier":"f4232a60-d80b-48ab-a5c5-eaeb9619e1a8",
  "Name":"720p-X-AVC",
  "Details":[
    {
      "Name":"Default",
      "Value":"False"
    }
  ],
  "Rendition":"1b5b2916-0948-4c3d-b726-d92df1ee6ad1",
  "Streams":[
    "b50a47dc-601f-43b7-a11a-0e2864112f75",
    "a0de7639-687c-4ad8-b226-dd2572cb1f7d"
  ]
}
```


Channels Operations

The Channels category of operations enables you to identify and operate on the channels in your Live Stream server. You can use these operations to query and control each channel and work with channel calendar events, and you can set the active segment on a channel.

- GetChannels
- GetChannel
- GetChannelOutputLocations
- SetVariantThumbnailSize
- GetChannelThumbnail
- StartChannel
- StopChannel
- GetSourcePlaceholdersForChannel
- AssignSourcesForChannel
- GetCalendarEvents
- GetCalendarEvent
- AddCalendarEvent
- DeleteCalendarEvent
- GetActiveSegment

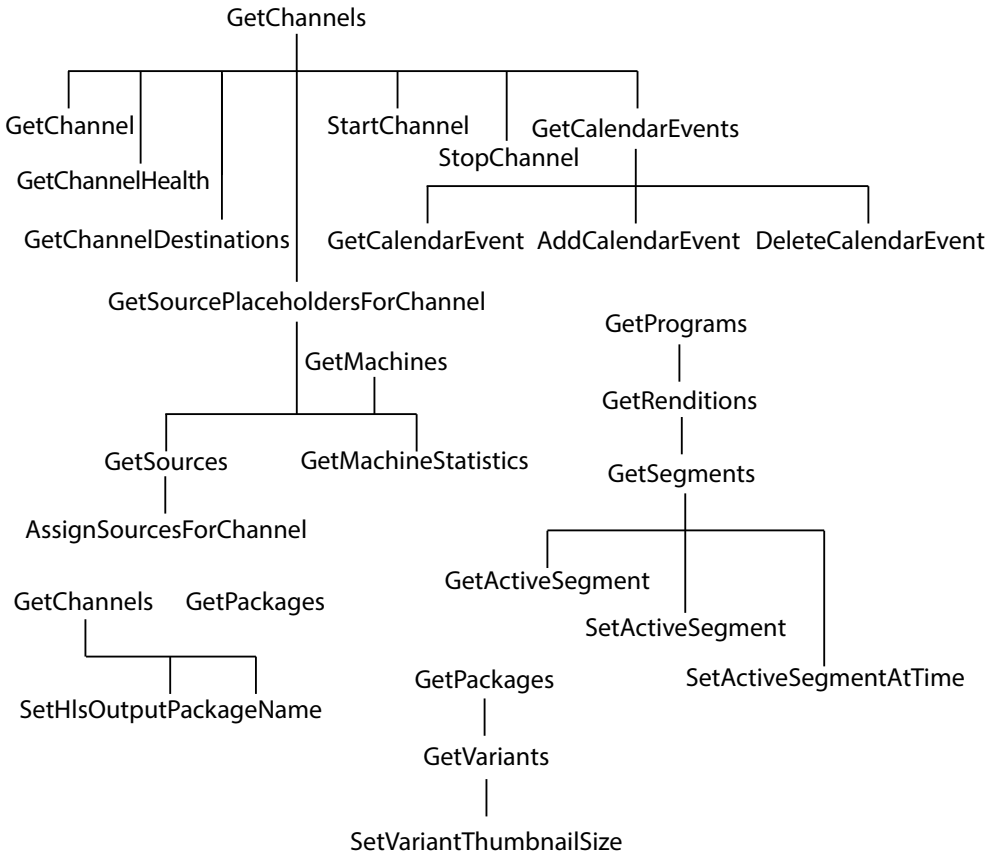
- SetActiveSegment
- SetActiveSegmentAtTime
- GetMachineStatistics
- GetChannelStatistics
- GetChannelHealth
- SetHlsOutputPackageName

For channel commands that operate on specific social media platforms and packages, see *Social Media Platform Channel Management Operations*.

Note: All Channels operations start with `http://<host>:<port>/Channels/`.

Channels are accessible to all the servers in a system and you can access or configure any channel from any server in the system by targeting a single machine.

In this category, channel operations are organized hierarchically in four different structures, based on their GUID requirements. Each operation in this category requires a channel GUID to execute correctly.



Note: To display help for Channels operations, execute `http://<host>:<port>/help` and open the Channels panel. 18000 is the default port—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

GetChannels

This GET operation returns a list of channels on the target Live Stream system.

This operation has no parameters.

URL Format

GetChannels has the following format:

`http://<host>:<port>/Channels/GetChannels`

Operation Sequence

No operations must be executed before you can execute this operation.

Results

On success, *GetChannels* returns an array, with one record for each channel in the Live Stream system.

If there are no channels, the array is returned empty ([]).

Example

`http://10.9.9.9:18000/Channels/GetChannels`

Typical Response

In this response, the Live Stream server has several channels; each with a Description, Identifier, and Name, which you can use to target the channel for further utilization.


```
[
  {
    "Description":null,
    "Identifier":"12133c5b-59dd-48cf-8443-ec7a2501b252",
    "Name":"HLS1"
  },
  {
    "Description":null,
    "Identifier":"40de4c1b-5523-4ed3-a89a-e319a3fc88d2",
    "Name":"RTMPAkamai"
  },
  {
    "Description":null,
    "Identifier":"486caeef-de83-4862-89c7-e79dab1a0564",
```

```
    "Name":"YTL"
  },
  {
    "Description":null,
    "Identifier":"826891c7-d81e-428d-b868-8f0ed58c77fe",
    "Name":"Dash_SCTE"
  }
]
```


GetChannel

This GET operation returns details about a specific channel on the target Live Stream server.

URL Format

GetChannel has the following format:

```
http://<host>:<port>/Channels/GetChannel?identifier={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific channel.

Results

On success, *GetChannel* returns a record with details, including events and packages in the specified channel.

If no channel exists with the specified GUID, an error is returned.

Example

```
http://10.9.9.9:18000/Channels/GetChannel
?identifier=a906f245-c29b-4187-9413-cd6c146e1576
```

Typical Response

In this response, the target channel is *TS1*, with all its details.


```
{
  "Description": null,
  "Identifier": "a906f245-c29b-4187-9413-cd6c146e1576",
  "Name": "TS1",
  "Active": false,
```

```
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "57f4d60b-ad3f-4f28-9864-0a82c7dc6942",
  "Packages": [
    {
      "Description": null,
      "Identifier": "313070b9-6804-4125-a73f-7df450c140da",
      "Name": "Transport Stream",
      "Details": [
        {
          "Name": "CBR padding",
          "Value": "10"
        },
        {
          "Name": "Output Location",
          "Value": "UDP Stream"
        },
        {
          "Name": "IP Address",
          "Value": "239.1.2.90"
        },
        {
          "Name": "Local NIC/IP Address",
          "Value": ""
        },
        {
          "Name": "Port Number - Variant '720p-X-AVC'",
          "Value": "1234"
        },
        {
          "Name": "Source Description Path",
          "Value": ""
        },
        {
          "Name": "Port Number Identifiers",
          "Value": "2000"
        }
      ],
      "Package": "37b70eee-bb7e-4ad2-b926-20244013f0bf"
    }
]
```


GetChannelOutputLocations

This GET operation returns the destinations for the target channel.

URL Format

GetChannelOutputLocations has the following format:

```
http://<host>:<port>/Channels/
GetChannelOutputLocations?channel={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies a specific channel.

Results

On success, *GetChannelOutputLocations* returns an array, with a record for each destination in the target channel (primary, and potentially multiple secondary).

Example

```
http://10.9.9.9:18000/Channels/GetChannelOutputLocations
?channel=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

In this response, the channel has a primary and secondary package; details include Description, Identifier, and Name, the output location, and the package GUID.


```
[
  {
    "Description":null,
    "Identifier":"af06a78f-d467-4112-bd28-671a0eca5bcb",
    "Name":"General outputs locations",
    "Details":[
      {
        "Name":"Output Location",
        "Value":"Push to CDN"
      },
      {
        "Name":"Publishing Point",
        "Value":"http:\/\/post.nwstudio.akamaihd.net\/554433"
      },
      {
        "Name":"HTTP Method",
        "Value":"POST"
      },
      {
        "Name":"Remove Local Copy",
        "Value":"True"
      },
      {
        "Name":"Output Package Name",
        "Value":"Default"
      }
    ],
    "Package":"19df3b43-470a-4451-b243-51997fe81a93"
```

```
  },
  {
    "Description":null,
    "Identifier":"af06a78f-d467-4112-bd28-671a0eca5bcb",
    "Name":"General outputs locations",
    "Details":[
      {
        "Name":"Output Location",
        "Value":"Push to CDN"
      },
      {
        "Name":"Publishing Point",
        "Value":"http:\/\/post.nwstudio.akamaihd.net\/554433"
      },
      {
        "Name":"HTTP Method",
        "Value":"POST"
      },
      {
        "Name":"Remove Local Copy",
        "Value":"True"
      },
      {
        "Name":"Output Package Name",
        "Value":"Default"
      }
    ],
    "Package":"19df3b43-470a-4451-b243-51997fe81a93"
  }
]
```


SetVariantThumbnailSize

This POST operation sets the size of thumbnails obtained from the target variant.

Note: Variants can be shared across channels, so *SetVariantThumbnailSize* affects all instances of this variant in every channel where it is used.
The variant must be inactive before you execute this operation. After execution, you must restart the channel of the variant, then execute *GetChannelThumbnail*.

Use this operation to specify the size before calling *GetChannelThumbnail*. This operation has no effect on the size of thumbnails presented in the Channels panel of the web app.

Output sizes up to 4096 x 2160 are supported. Use 0 for height/width to revert the size to their default values (160 x 90).

URL Format

`http://<host>:<port>/Channels/SetVariantThumbnailSize`

Body Format

```
{
  "variant":"VARIANT GUID",
  "width":FRAME WIDTH,
  "height":FRAME HEIGHT
}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetPackages (package GUID) > *GetVariants* (variant GUID)
or
GetPackages (package GUID) > *GetPackage* (variant GUID)

Parameters

Parameter	Description
variant	GUID; identifies a specific variant.
width	INT; specifies the width of the thumbnail in pixels. Max: 4096.
height	INT; specifies the height of the thumbnail in pixels. Max: 2160.

Results

On success, *SetVariantThumbnailSize* returns a string indicating that the thumbnail size has been set.

Example

```
http://10.0.25.162:18000/Channels/SetVariantThumbnailSize
```

```
{
  "variant":"b0e7a6c0-a920-4702-b4f7-5fafe599a26c",
  "width":1280,
  "height":720
}
```

Typical Response

The operation succeeded: `"Set thumbnail size for variant 720p-X-AVC".`

GetChannelThumbnail

This GET operation returns a thumbnail of a specific variant of the target active channel (inactive channels do not have thumbnails). The operation returns a JPEG, which you can display (render) or save as a file.

The *GetChannelThumbnail* command requires both a channel and variant GUID. So to get the thumbnail of variant A, you provide the variant A GUID; to get the thumbnail of variant B, you provide the variant B GUID.

Variants are source-agnostic, while channels are not. Variant A may be used in Channel A and Channel B at the same time, which may be using different sources, in which case the thumbnails will be the same size but different content.

Note: If you have modified the thumbnail size (using *SetVariantThumbnailSize),* you must restart the target channel for the size modification to take effect, then execute *GetChannelThumbnail*.

URL Format

GetChannelThumbnail has the following format:

```
http://<host>:<port>/Channels/GetChannelThumbnail
?channel={CHANNEL GUID}&variant={VARIANT GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetChannels (channel GUID)
and
GetPackages (package GUID) > *GetVariants* (variant GUID)
or
GetPackages (package GUID) > *GetPackage* (variant GUID)

Parameters

Parameter	Description
channel	GUID; identifies a specific channel.
variant	GUID; identifies a specific variant.

Results

On success, *GetChannelThumbnail* returns a JPEG.

If the channel is inactive, an HTTP 404 error is returned—there is no thumbnail.

Example

```
http://10.9.9.9:18000/Channels/GetChannelThumbnail
?channel=27602854-35a6-4f0c-827c-ebd7ac5d40a9
&variant=b0e7a6c0-a920-4702-b4f7-5fafe599a26c
```

StartChannel

The purpose of this POST operation is to begin streaming a specified channel. When you start the channel, the active segment is encoded according to the package variant definitions and delivered to the locations you have specified.

Note: The host that you specify must be the server where the channel was added.

URL Format

```
http://<host>:<port>/Channels/StartChannel
```

Body Format

```
"CHANNEL GUID"
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
`identifier`	GUID; identifies a specific channel.

Results

On success, *StartChannel* returns a status message indicating success: `"Successfully started channel"`.

Example

```
http://10.9.9.9:18000/Channels/StartChannel
"d5d2a977-68c9-4fa3-a687-5d60535d4958"
```

Typical Response

In this response, the operation returned: `"Successfully started channel."`.

StopChannel

The purpose of this POST operation is to terminate the streaming of a specified channel.

Note: The host that you specify must be the server where the channel was added.

URL Format

```
http://<host>:<port>/Channels/StopChannel
```

Body Format

```
"CHANNEL GUID"
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
Identifier	GUID; identifies a specific channel.

Results

On success, *StopChannel* returns a status message indicating success: `"Successfully stopped channel"`.

Example

```
http://10.9.9.9:18000/Channels/StopChannel
"d5d2a977-68c9-4fa3-a687-5d60535d4958"
```

Typical Response

In this response, the operation returned: `"Successfully stopped channel."`.

GetSourcePlaceholdersForChannel

This GET operation returns a list of all source placeholders for the specified channel. This operation is used to operate on a specific source.

URL Format

GetSourcePlaceholdersForChannel has the following format:

```
http://<host>:<port>/Channels/GetSourcePlaceholdersForChannel
?channel={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies a specific channel.

Results

On success, *GetSourcePlaceholdersForChannel* returns an array, with a record for each source placeholder in the target channel.

Example

```
http://10.9.9.9:18000/Channels/GetSourcePlaceholdersForChannel
?channel=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

In this response, the channel has two source placeholders, uniquely identified by GUID.

```
{
  "Briefs":
  [
    {
      "Description":null,
      "Identifier":"d98672bb-909a-4f19-8c5d-a3d5b3fbad42",
      "Name":"Source Placeholder"
    }
  ,
    {
      "Description":null,
      "Identifier":"27af3c44-327f-2789-e313-32787ab81a37",
      "Name":"Source Placeholder"
    }
  ]
}
```

AssignSourcesForChannel

The purpose of this POST operation is to attach a source to a specific source placeholder (identified as the *Live* key pair value) on a given channel. Each source placeholder is identified in the array of the request body; you can specify one or more Live-Source pairs per operation.

URL Format

```
http://<host>:<port>/Channels/AssignSourcesForChannel
```

Body Format

```
{
  "Assignments":
  [
    {
```

```
      "Live":"SOURCE PLACEHOLDER GUID",
      "Source":"SOURCE GUID"
    }
  ],
  "Channel":"CHANNEL GUID"
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetChannels (channel GUID) > *GetSourcePlaceholdersForChannel* (Source Placeholder GUID)
and
GetSources (source GUID)

Parameters

Parameter	Description
live	GUID; identifies a specific source placeholder.
source	GUID; identifies a specific source.
channel	GUID; identifies a specific channel.

Results

On success, *AssignSourcesForChannel* returns a status message: `"Successfully set source assignments"`.

Example

```
http://10.9.9.9:18000/Channels/AssignSourcesForChannel
```

```
{
  "Assignments":
    [
      {
       "live":"d98672bb-909a-4f19-8c5d-a3d5b3fbad42",
       "source":"b60038aa-ac76-4301-83f6-3615529a88e7"
      }
    ],
  "channel":"d0547e27-ab43-4a70-9701-a8eb038b6203"
}
```

Typical Response

In this response, the operation was successful: `"Successfully set source assignments"`.

GetCalendarEvents

This GET operation returns a list of CalendarEvents from the target channel.

URL Format

GetCalendarEvents has the following format:

```
http://<host>:<port>/Channels/GetCalendarEvents?channel={CHANNEL
GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies a specific channel.

Results

On success, *GetCalendarEvents* returns an array, with one record for each calendar event in the channel. If no calendar events are present, the array is returned empty ([]).

Example

```
http://10.9.9.9:18000/Channels/GetCalendarEvents
?channel=9378dd79-a374-46f8-9f23-dc68d8d52d07
```

Typical Response

In this response, the target channel has two calendar events; with a Description, Identifier, and Name, which you can use to target the calendar event for further utilization.

```
[
  {
    "Description": null,
    "Identifier": "4ac69260-b10e-4da9-955e-831ea02a814b",
    "Name": "Event1"
  },
  {
    "Description": null,
    "Identifier": "290c6ad3-3b2a-42cd-9e31-aac9fd069d93",
    "Name": "Event2"
  }
]
```

GetCalendarEvent

This GET operation returns details about a specific calendar event in the target channel.

URL Format

GetCalendarEvent has the following format:

```
http://<host>:<port>/Channels/
GetCalendarEvent?identifier={CALENDAREVENT GUID}
```

Operation Sequence

Execute the following operations to obtain the required GUID for this operation:

GetChannels (channel GUID) > *GetCalendarEvents* (calendar-event GUID)
or
GetChannels (channel GUID) > *GetChannel* (calendar-event GUID)

Parameters

Parameter	Description
identifier	GUID; identifies a specific calendar event.

Results

On success, *GetCalendarEvent* returns a record, with details about the event, including frequency, and start and end times.

If no calendar event exists with the specified GUID, an error is returned. For example, `"Could not find calendar event with identifier 4ac69260-b10e-5da9-955e-831ea02a814b"`.

Example

```
http://10.9.9.9:18000/Channels/GetCalendarEvent
?identifier=2f5df131-deaf-4789-87dd-7ff850bcf1dc
```

Typical Response

In this response, the target CalendarEvent has a Description, Identifier, and Name, plus details about the event, including the start and end timestamps.


```
{
    "Description": null,
    "Identifier": "4ac69260-b10e-4da9-955e-831ea02a814b",
    "Name": "Event1",
    "Color": {
        "primary": "green",
        "secondary": null
    },
    "Details": [
```

```
            {
                "Name": "Frequency",
                "Value": "Does Not Repeat"
            },
            {
                "Name": "By day",
                "Value": ""
            }
        ],
        "End": "2017-11-15T10:00:00.0000000-08:00",
        "Start": "2017-11-15T09:00:00.0000000-08:00",
        "Title": null
}
```

AddCalendarEvent

The purpose of this POST operation is to add a calendar event to the specified channel. A calendar event is the date and time you want to start and stop broadcasting a channel. You can add as many calendar events as you require. You can optionally specify an exact source frame at which to start and stop the channel.

Note: If you add multiple events with the same name, the name is appended with (<999>) where 999 is an incremental integer starting at 1.

URL Format

`http://<host>:<port>/Channels/AddCalendarEvent`

Body Format

```
{
  "channel":"CHANNEL GUID",
  "name":"CALENDAR EVENT NAME",
  "start":"START DATE TIMESTAMP",
  "startTimeCode":"START TIMECODE",
  "end":"END DATE TIMESTAMP"
  "endTimeCode":"END TIMECODE",
}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (Channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.
name (optional)	String; practical name that identifies this calendar event. If not supplied, the default string `"Scheduled Stream Activation"` is placed in the Name field.
start	Date/Time stamp string; identifies the computer-based date and time to start the channel. If GMT is not specified, the time is local to the server. Support is provided for date/time strings that follow a recognized pattern, as described in the Microsoft .net DateTime.Parse method (for details, see https://msdn.microsoft.com/en-us/library/system.datetime.parse.aspx#StringToParse). For example, MM-DD-YYYY HH:MM:SS or Thu, 01 May 2017 07:34:42 GMT. If you include the *StartTimeCode* parameter, the Start SS value should be approximately 10 seconds prior to the *StartTimeCode* SS value (taking into account that one is computer time and the other is source time, of course) to make sure that its start is frame-accurate.
startTimeCode (optional)	Timecode string; identifies the source-based frame in HH:MM:SS:FF timecode format on which to start the channel. The *StartTimeCode* value is inclusive, and the SS value should be approximately 10 seconds after the SS value of the *Start* parameter. All values must be two digits, and the HH value represents a 24-hour clock. For example: 10:30:00:00. Invalid frame values (for example, specifying frame 25 or higher in 25FPS video) causes unexpected behavior.
end	Date/Time stamp string; identifies the computer-based date and time to stop the channel, as described above in *Start*. If you include the *EndTimeCode* parameter, the End SS value should be approximately 10 seconds after the *EndTimeCode* SS value (taking into account that one is computer time and the other is source time, of course) to make sure that it its end is frame-accurate.
endTimeCode (optional)	Timecode string; identifies the source-based frame in HH:MM:SS:FF timecode format on which to end the channel. The *EndTimeCode* is exclusive, and the SS value should be approx. 10 seconds before the SS value of the End parameter. All values must be two digits, and the HH value represents a 24-hour clock. Invalid frame values (for example, specifying frame 25 in 25FFS video) causes unexpected behavior.

Results

On success, *AddCalendarEvent* returns a record with the GUID of the new event.

Examples

Two examples are presented: the first, with just a date/time stamp. The second (using the same start and stop times) is functionally equivalent, but includes a frame-accurate starting and ending TimeCode.

Date/Time Only Example

In this example, only the Start and End parameters are provided.

```
http://10.0.2.258:18000/Channels/AddCalendarEvent
```

```
{
  "channel":"9378dd79-a374-46f8-9f23-dc68d8d52d07",
  "name":"FirstEvent",
  "start":"03-15-2017 00:00:00",
  "end":"03-15-2017 10:00:00"
}
```

TimeCode Example

In this example, the channel will start on about 9AM local time, and start streaming the media exactly at the 01:00:00:00 timecode. It will end about an hour later at about 10AM local, exactly at the 02:00:00:00 timecode (exclusive of course).

```
http://10.0.2.258:18000/Channels/AddCalendarEvent
```

```
{
  "channel":"9378dd79-a374-46f8-9f23-dc68d8d52d07",
  name":"FirstEvent",
  "start":"03-15-2017 08:59:50",
  "startTimeCode":"01:00:00:00",
  "end":"03-15-2017 10:00:10",
  "endTimeCode":"02:00:00:00"
}
```

Typical Response

Both operations generate an identical response if successful. The new event was successfully added to the channel; its Identifier GUID is returned, which you can use to target the event for further utilization.

```
{
    "Description": null,
    "Identifier": "4ac69260-b10e-4da9-955e-831ea02a814b",
    "Name": "Event1"
}
```

DeleteCalendarEvent

The purpose of this POST operation is to delete a calendar event from a channel.

URL Format

```
http://<host>:<port>/Channels/DeleteCalendarEvent
```

Body Format

```
{
  "channel":"CHANNEL GUID",
```

```
  "calendarEvent":"CALENDAR EVENT GUID"
}
```

Operation Sequence

Execute one of these operation sequences to obtain the GUID for this operation:

GetChannels (channel GUID)
and
GetCalendarEvents(calendar-event GUID)
or
AddCalendarEvent (calendar-event GUID)
or
GetChannels (channel GUID) > *GetChannel* (calendar-event GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.
calendarEvent	GUID; identifies this calendar event.

Results

On success, *DeleteCalendarEvent* returns a status message: “Successfully deleted calendar event”.

Example

```
http://10.0.2.258:18000/Channels/DeleteCalendarEvent
```

```
{
  "channel":"9378dd79-a374-46f8-9f23-dc68d8d52d07",
  "calendarEvent":"a82a7f8e-d3c1-4bab-8644-67a4f5917a52"
}
```

Typical Response

In this response, the new event was successfully deleted. A successful Status was returned: “Successfully deleted calendar event”.

GetActiveSegment

This GET operation returns details about the active segment on the target channel, which must be active.

URL Format

GetActiveSegment has the following format:

```
http://<host>:<port>/Channels/GetActiveSegment?
channel={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.

Results

On success, *GetActiveSegment* returns the active segment's record.

If the channel isn't active, it returns an error: `"Channel is not active"`.

If no channel exists with the specified GUID, an error is returned: `"Could not find channel with identifier <channel GUID>"`.

Example

```
http://10.9.9.9:18000/Channels/GetActiveSegment?
channel=68db056e-6ab7-4898-a6cd-5c888422606d
```

Typical Response

In this response, the active segment is returned.

```
{
  "Description":null,
  "Identifier":"f269a2b3-c8e5-41a6-8855-118de2c3b94e",
  "Name":"SCTE Ad Break Segment"
}
```

SetActiveSegment

The purpose of this POST operation is to immediately start streaming the specified segment, stopping streaming of the current active segment. The channel must be running—that is, you must have started the channel. The specified segment may be the current segment being streamed—no error is returned.

Note: The segment must be activated directly on the target server.

Typical use cases for manually changing the active segmentPrograms Operations include correcting a state that is determined to be incorrect, and manually (or with server-side logic) changing the segment to the correct segment.

Another use case: If you or your system determines that the next media segment is one you don't want to broadcast, you can switch from the segment you want to avoid to a different segment (even a “We'll be right back” segment, with color bars).

URL Format

```
http://<host>:<port>/Channels/SetActiveSegment
```

Body Format

```
{
  "channel":"CHANNEL GUID",
  "segment":"SEGMENT GUID"
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetChannels (channel GUID)
and
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > GetProgram (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > GetProgram (segment GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.
segment	GUID; identifies this segment.

Results

On success, *SetActiveSegment* returns the success message: "Segment successfully triggered".

If the channel is not active, an error is returned: "Channel is not active".

Example

```
http://10.0.25.158:18000/Channels/SetActiveSegment
```

```
{
  "channel":"ad1c45b7-67fb-419d-8c5b-8ba474bd6dfd",
  "segment":"1d20e392-8876-411a-9681-70e08e7baca9"
}
```

Typical Response

In this response, the Live Stream system reports the successful operation: "Segment successfully triggered".

SetActiveSegmentAtTime

This POST operation schedules a segment to start broadcasting at a specified time for the current date. The channel must be running—you must have started the channel. This operation does not remove configured triggers for any segments. If the channel is stopped and started again, it will start at the base segment of the source again.

The specified source may be the current source being streamed—no error is returned.

Note: The segment must be activated directly on the target server.

URL Format

```
http://<host>:<port>/Channels/SetActiveSegment
```

Body Format

```
{
  "channel":"CHANNEL GUID",
  "degment":"SEGMENT GUID",
  "time":"HH:MM:SS"
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetChannels (channel GUID)
and
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > GetProgram (rendition GUID) > *GetSegments* (segment GUID)
or
GetPrograms (program GUID) > *GetRenditions* (rendition GUID) > GetProgram (segment GUID)

Parameters

Parameter	Description
vhannel	GUID; identifies this channel.
degment	GUID; identifies this segment.
time	Time code (in format HH:MM:SS 24-hour time) at which the segment should begin.

Results

On success, *SetActiveSegmentAtTime* returns a status message: "Successfully added trigger". If the channel is not active, an error is returned: "Channel is not active".

Example

```
http://10.0.25.158:18000/Channels/SetActiveSegmentAtTime
```

```
{
  "channel":"ad1c45b7-67fb-419d-8c5b-8ba474bd6dfd",
  "segment":"1d20e392-8876-411a-9681-70e08e7baca9",
  "time":"10:40:00"
}
```

Typical Response

In this response, the returned message reports the successful operation: "Successfully added trigger".

GetMachineStatistics

This GET operation returns statistical information about the target server. Statistics are only available when at least one channel on the target server is streaming.

URL Format

GetMachineStatistics has the following format:

```
http://<host>:<port>/Channels/GetMachineStatistics?
machine={MACHINE GUID}
```

Operation Sequence

Execute this operation to obtain the machine GUID required to execute this operation.

GetMachines (machine GUID)

Parameters

Parameter	Description
machine	GUID; identifies the target Live Stream server.

Results

On success, *GetMachineStatistics* returns a record, with a variety of statistical values.

If the machine GUID does not exist or no channels are active, an error is returned: `"Failed to generate statistics. No channels belonging to machine b6347bb5-c8c8-4bb1-8cec-53342ca6225d are currently active"`, for example.

Example

```
http://10.0.25.158:18000/Channels/Getmachinestatistics?
machine=af0f694b-5f17-4b8f-af13-34486d488012
```

Typical Response

```
{
  "Description":null,
  "Identifier":"da716ac0-d5e2-4d18-a718-533b42f6457a",
  "Name":"QA-VL-LIVE-9",
  "CpuUsage":15,
  "DiskSpaceUsed":"88",
  "GpuComputeUtilization":34,
  "GpuEncoderUtilization":0,
  "GpuMemory":13,
  "Memory":15
}
```

GetChannelStatistics

This GET operation returns statistical information about the target active channel.

URL Format

GetChannelStatistics has the following format:

```
http://<host>:<port>/Channels/GetChannelStatistics
?channel={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.

Results

On success, *GetChannelStatistics* returns a record with a variety of statistical values.

If the channel GUID does not exist, an error is returned: `"Could not find channel with identifier 68db056e-6ab7-4898-a6cd-5c888422606d"`, for example.

If the channel is not active, an error is returned: “Channel is not active”.

Example

```
http://10.0.25.158:18000/Channels/GetChannelStatistics?
channel=68db056e-6ab7-4898-a6cd-5c888422606d
```

Typical Response

```
{
  "Description": null,
  "Identifier": "808f9b52-5ad4-4520-82ce-c3df5175c48f",
  "Name": "HLS_Test",
  "CpuUsage": 2,
  "GpuMemory": 0,
  "Memory": 1092,
  "OutputLocation": null,
  "Uptime": "0:00:24"
}
```

GetChannelHealth

This GET operation returns details regarding the current operational status of the target active channel, including dropping frames or recovered, when queues are full or recovered, and if the channel restarted because of a program or encoder failed.

URL Format

GetChannelStatistics has the following format:

```
http://<host>:<port>/Channels/GetChannelHealth
?channel={CHANNEL GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.

Results

On success, *GetChannelHealth* returns a document with operational metrics:

Possible statuses (in the *value* member) are:

- OK
- Input queue is full
- Recovered from full input queue
- Dropping frames
- Recovered from dropping frames
- Non-divisible aspect ratio
- Frame rate up-conversion
- Resolution up-conversion
- Encoder restarted
- Program restarted
- Encoder failed
- Inactive

If the channel GUID does not exist, a 404 error is returned: *"Could not find channel with identifier 68db056e-6ab7-4898-a6cd-5c888422606d"*, for example.

If the channel is not active, an error is returned: "*Channel is not active*".

Example

```
http://10.0.25.158:18000/Channels/GetChannelHealth?
channel=68db056e-6ab7-4898-a6cd-5c888422606d
```

Typical Response


```
{
  "Identifier": "68db056e-6ab7-4898-a6cd-5c888422606d",
  "Name": "HLS1",
  "Description": "Channel is healthy.",
  "Details": [
    {
      "name": "Status",
      "value": "OK"
    }
  ]
```

```
}
```

SetHlsOutputPackageName

Use this POST operation to specify or modify the name of an HLS output package.

Note: In the event that the same package has been added to a channel multiple times, only the first matching package found will be affected.

URL Format

```
http://<host>:<port>/Channels/SetHlsOutputPackageName
```

Body Format

```
{
  "channel":"CHANNEL GUID",
  "package":"PACKAGE GUID",
  "baseName":"Base Name String",
  "attachDateTime":BOOLEAN
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetChannels (channel GUID)
and
GetPackages (package GUID)
or
GetChannel (package GUID)

Parameters

Parameter	Description
channel	GUID; identifies this channel.
package	GUID; identifies this package.
baseName	String; the name to specify for this package.
attachDateTime (optional)	Boolean; true | false specifies whether to attach the current date and time to the output package name.

Results

On success, *SetHlsOutputPackageName* returns a status message: `"Successfully changed output package name to <NewOutputPackageName>"`.

Example

```
http://10.0.25.158:18000/Channels/SetHlsOutputPackageName
```

```
{
  "channel":"ad1c45b7-67fb-419d-8c5b-8ba474bd6dfd",
  "segment":"1d20e392-8876-411a-9681-70e08e7baca9",
  "baseName":"NewOutputPackageName",
  "attachDateTime":true
}
```

Typical Response

In this response, the returned message reports the successful operation:
`"Successfully changed output package name to NewOutputPackageName".`

Social Media Platform Channel Management Operations

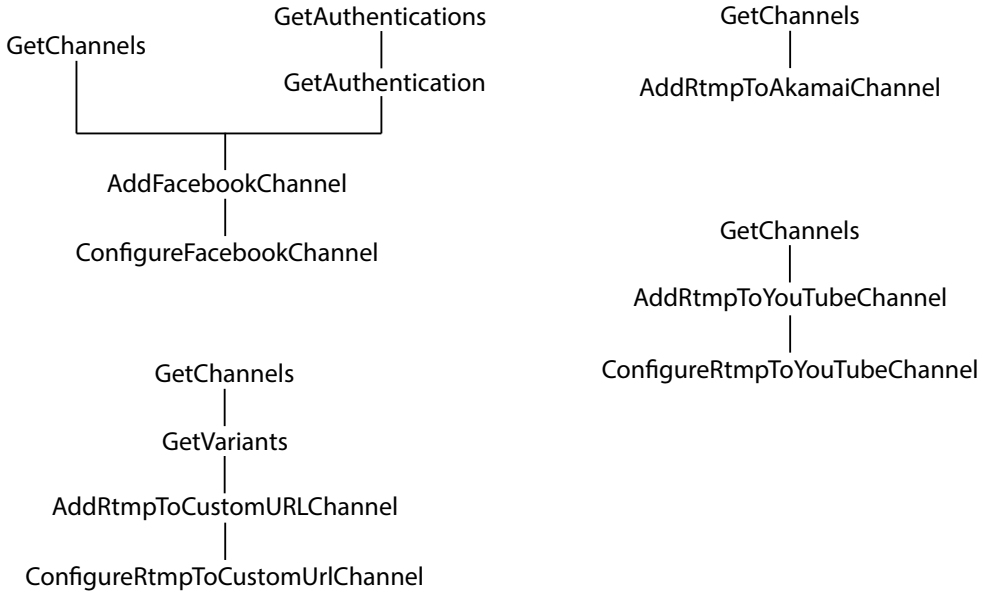
Use these Channels operations to create and configure channels for specific social media platforms and packages on your Live Stream server.

Note: Some operations require an authenticated social media account. For adding and obtaining these records, see *GetNewAuthenticationDetails* and *GetAuthentication*.

- AddFacebookChannel
- ConfigureFacebookChannel
- AddRtmpToAkamaiChannel
- AddRtmpToCustomUrlChannel
- ConfigureRtmpToCustomUrlChannel
- AddRtmpToYouTubeChannel
- ConfigureRtmpToYouTubeChannel

Note: All Channels operations start with `http://<host>:<port>/Channels/`.

In this category, channel operations are organized by social platform:



Note: To display help for Channels operations, execute `http://<host>:18000/help` and open the Channels panel. The default port is 18000—see *Ports for Lightspeed Live Server Access* for displaying or changing this port. See *Obtaining Help for Stream & Source Operations* for more info.

AddFacebookChannel

The purpose of this POST operation is to add a new Facebook channel to Live Stream, using a social media account record.

URL Format

```
http://<host>:<port>/Channels/AddFacebookChannel
```

Body Format

```
{
  "authentication":"AUTHENTICATION GUID",
  "name":"CHANNEL NAME",
  "package":"PACKAGE GUID"
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetAuthentication (Social Media Account record GUID)
and
GetPackages (package GUID).

Parameters

Parameter	Description
authentication	GUID; identifies the social media account record used for accessing this social media platform. Obtained from *GetAuthentication*.
name	String; specifies the name for the new channel.
package	GUID; identifies the primary package; obtained from *GetPackages*.

Results

On success, *AddFacebookChannel* returns a channel record.

If the device code you use is not valid, this error is returned: "An error occurred while creating channel: Invalid Facebook verification device code".

If the user you supply is not authenticated, this error is returned: "Penelope_N45PLJ is not authenticated for Facebook Live", for example.

If your channel type is incorrect, or not constructed correctly, or there are other configuration errors, Live Stream provides contextual error messages.

Example

```
http://LS-SVR:18000/Channels/AddFacebookChannel
```

```
{
  "authentication":"1af0ba42-de0e-48d0-8fa4-db14401e5bda",
  "name":"BoatsOfPortTownsend",
  "package":"90ad1d8f-d10b-425e-b39d-a7e447cc766b"
}
```

Typical Response

In this response, the Channel record reports the GUID details of the new *FBChannel007* channel added to the Live Stream system.


```
{
  "Description": null,
  "Identifier": "f2d69e96-7b61-407b-8406-44757d833e90",
  "Name": "BoatsOfPortTownsend",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "57f4d60b-ad3f-4f28-9864-0a82c7dc6942",
  "Packages": [
    {
      "Description": null,
      "Identifier": "90ad1d8f-d10b-425e-b39d-a7e447cc766b",
      "Name": "FacebookLive Outputs",
      "Details": [
        {
          "Name": "User Name",
          "Value": "Larry Wood"
        },
        {
          "Name": "Authentication Identifier",
          "Value": "f0533a5f-8738-4d23-ac7a-84441ef5ad7e"
        }
      ],
      "Package": "e95b7006-c8b1-4af8-80fd-55ba16422d90"
    }
  ]
}
```


ConfigureFacebookChannel

The purpose of this POST operation is to update the settings of an existing Facebook channel.

URL Format

```
http://<host>:<port>/Channels/ConfigureFacebookChannel
```

Body Format

```
{
  "Channel":"CHANNELGUID",
  "DeleteVideo":BOOLEAN,
  "Description":"DESCRIPTION",
  "PostToEvent":"EVENT",
  "PostToGroup":"GROUP",
  "PostToPage":"PAGE",
  "Privacy":"PRIVACY KEYWORD",
  "SecureConnection":BOOLEAN,
  "StartDate":"START DATE TIMESTAMP",
  "Title":"TITLE"
}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channel	GUID; identifies the target Facebook channel, obtained from *GetChannels*.
startDate (optional)	Date/time stamp string; identifies the date and time to start the channel. If GMT is not specified, the time is local to the server. Support is provided for date/time strings that follow a recognized pattern, as described in the Microsoft .net DateTime.Parse method (for details, see https://msdn.microsoft.com/en-us/library/system.datetime.parse.aspx#StringToParse). For example: `MM-DD-YYYY HH:MM:SS or Thu, 01 May 2017 07:34:42 GMT`. If a startDate is not included, the schedule is set to *Go Live Now*. If the startDate is included, the schedule is set to *Schedule for Later*.
title (optional)	String; displayed as the title of the Facebook Live event.
description (optional)	String; description displayed as the description of the event.
postToPage postToGroup postToEvent (optional)	String; the name of the target page, group, or event. The *postToPage*, *postToGroup*, and *postToEvent* parameters are mutually exclusive. Use only one (or none) to select where to post the video. If none of these parameters are present, *postTo* is set to "User".

Parameter	Description
deleteVideo (optional)	Boolean; true | false to specify whether the video should be deleted after broadcast.
secureConnection (optional)	Boolean; true | false to specify whether to use a secure connection for the stream over SSL (RTMPS).
privacy (optional)	Keyword string: Public | Friends | Only Me to specify the privacy level for this broadcast. This is only relevant if the *postTo* parameter is set to *User*. When posting to Pages, Groups, and Events, privacy is *Public*.

Results

On success, *ConfigureFacebookChannel* returns a record with the channel details.

Example

```
http://10.0.25.158:18000/Channels/ConfigureFacebookChannel
```

```
{
  "channel":"90e90139-cace-4d10-8024-6533b1b74edd",
  "deleteVideo":true
}
```

Typical Response


```
{
  "Identifier": "f2d69e96-7b61-407b-8406-44757d833e90",
  "Name": "BoatsOfPortTownsend",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "57f4d60b-ad3f-4f28-9864-0a82c7dc6942",
  "Packages": [
    {
      "Description": null,
      "Identifier": "90ad1d8f-d10b-425e-b39d-a7e447cc766b",
      "Name": "FacebookLive Outputs",
      "Details": [
        {
        "Name": "User Name",
        "Value": "Larry Wood"
      },
      {
        "Name": "Authentication Identifier",
        "Value": "f0533a5f-8738-4d23-ac7a-84441ef5ad7e"
      },
      {
```

```
      "Name": "Schedule",
      "Value": null
    },
    {
      "Name": "Event Title",
      "Value": ""
    },
    {
      "Name": "Broadcast Description (Optional)",
      "Value": ""
    },
    {
      "Name": "Secure Connection (SSL)",
      "Value": "False"
    },
    {
      "Name": "Delete after Broadcast",
      "Value": "True"
    },
    {
      "Name": "Stream Id",
      "Value": "10218125435513155"
    },
    {
      "Name": "Status",
      "Value": "UNPUBLISHED"
    },
    {
      "Name": "Stream URL",
      "Value": "rtmp://live-api-s.facebook.com:80/rtmp"
    },
    {
      "Name": "Stream Name",
      "Value": "10218125435513155?s_sw=0&s_vt=api
          -s&a=AbxDgMgCSZOAOJlw"
    },
    {
      "Name": "Secure Stream URL",
      "Value": "rtmps://live-api-s.facebook.com:443/rtmp"
    },
    {
      "Name": "Secure Stream Name",
      "Value": "10218125435513155?s_sw=0&s_vt=api
          -s&a=AbxDgMgCSZOAOJlw"
    },
    {
      "Name": "Post To",
      "Value": "User"
    },
    {
      "Name": "Privacy",
      "Value": "Public"
    }
  ],
    "Package": "e95b7006-c8b1-4af8-80fd-55ba16422d90"
  }
]
```

```
}
```

AddRtmpToAkamaiChannel

The purpose of this POST operation is to add a new RTMP channel to Akamai using the stream/streamkey RTMP publishing method via Live Stream.

URL Format

```
http://<host>:<port>/Channels/AddRtmpToAkamaiChannel
```

Body Format

```
{
  "akamaiUri":"AKAMAI URI",
  "angle":"ANGLE KEYWORD",
  "attachDate":BOOLEAN,
  "channelName":"CHANNEL NAME",
  "outputPackageName":"OUTPUT PACKAGE NAME",
  "package":"PACKAGE",
  "password":"PASSWORD",
  "streamId":"STREAM ID",
  "userName":"USER NAME"
}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetPackages (Package GUID)

Parameters

Parameter	Description
akamaiUri (optional)	URI string; URI of the Akamai end point for this stream. For example: `RTMP://p.ep500211.i.akamaientrypoint.net/EntryPoint.`
angle (optional)	Keyword string; user-specified value used in the playback URL from Akamai. For example: `HTTP://rtmp1-lh.akamaihd.net/i/RTMPAkamai_[angle value]@500211/master.m3u8.` If not supplied, defaults to *live*.
attachDate (optional)	Boolean; true | false specifies whether to attach the current date and time to the output package name.
channelName	String; the name of the RTMP channel being created.
outputPackageName (optional)	String; The base name of the output package. If not supplied, defaults to the value you supply in ChannelName.

Parameter	Description
package	GUID; identifies the package to use in the channel.
userName	String; registered Akamai user name.
password	String; password associated with Akamai user name.
streamId	GUID; identifies the stream to use in the channel.

Results

On success, *AddRtmpToAkamaiChannel* returns a record with the channel details.

Example

```
http://10.0.25.158:18000/Channels/AddRtmpToAkamaiChannel
```

```
{
   "akamaiUri":"rtmp://p.ep500211.i.akamaientrypoint.net/
EntryPoint",
   "angle":"live",
   "attachDate":false,
   "channelName":"Akamai_RTMP",
   "outputPackageName":"PortTownsend",
   "package":"3482ea4d-5215-4357-ac98-a77aae32dfad",
   "password":"test1",
   "streamId":"500000",
   "userName":"test"
}
```

Typical Response

In the example, the operation was successful and returned this response body with the channel details:

```
{
  "Description": null,
  "Identifier": "728bf13d-47b5-43a2-a8d8-6f3d08eeeb67",
  "Name": "Akamai_RTMP",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "c035c768-50da-41d6-ae61-49b3f129326c",
  "Packages": [
    {
      "Description": null,
      "Identifier": "35d75893-5e97-499a-a5fe-e510dfd22dab",
      "Name": "Rtmp Outputs",
      "Details": [
        {
```

```
            "Name": "Destination",
            "Value": "Akamai"
          },
          {
            "Name": "Akamai URL",
            "Value": "rtmp://p.ep500211.i.akamaientrypoint.net/
EntryPoint"
          },
          {
            "Name": "Stream Id",
            "Value": "500000"
          },
          {
            "Name": "Username",
            "Value": "******"
          },
          {
            "Name": "Password",
            "Value": "******"
          },
          {
            "Name": "Angle",
            "Value": "live"
          },
          {
            "Name": "Output Package Name",
            "Value": "Custom Name"
          },
          {
            "Name": "Base Name",
            "Value": "PortTownsend"
          },
          {
            "Name": "Attach Date/Time",
            "Value": false
          }
        ],
        "Package": "3482ea4d-5215-4357-ac98-a77aae32dfad"
      }
    ]
}
```


AddRtmpToCustomUrlChannel

The purpose of this POST operation is to connect a Live Stream Channel (including an RTMP Package) to an endpoint of a user-specified URL, using the stream/streamkey RTMP publishing method.

URL Format

`http://<host>:<port>/Channels/AddRtmpToCustomUrlChannel`

Body Format

```
{
  "channelName":"CHANNEL NAME",
  "package":"PACKAGE GUID",
  "userName":"USER NAME",
  "password":"PASSWORD",
  "variantUris":[{
    "streamName":"STREAM NAME",
    "uri":"URI",
    "variant":"VARIANT GUID"
  }]
}
```

Operation Sequence

Execute the following operations to obtain the required GUIDs for this operation:

GetPackages (Package GUID)
and
GetVariants (Variant GUID)

Parameters

Parameter	Description
channelName	String that identifies this new channel.
package	GUID; identifies the package to use in the channel.
userName	String; registered user name.
password	String; password associated with the user name.
variantUris	Array; One or more permitted to specify each variant in the package.
streamName (optional)	String; name of the stream.
uri (optional)	String; URI of the stream for this variant.
variant	GUID; identifies the variant to use in the channel.

Results

On success, *AddRtmpToCustomUrlChannel* returns a JSON response body with the channel details.

Example

```
http://10.0.25.158:18000/Channels/AddRtmpToCustomUrlChannel
{
  "channelName":"RTMP_Test",
  "package":"3482ea4d-5215-4357-ac98-a77aae32dfad",
```

```
  "password":"telestream",
  "userName":"wirecast",
  "variantUris":[{
    "streamName":"Test1",
    "uri":"rtmp://qa-xbuster.telestream.net:1935/live",
    "variant":"a02ab97b-a318-4061-bf33-ccbf0c3a1414"
  }, {
    "streamName":"Test2",
    "uri":"rtmp://qa-xbuster.telestream.net:1935/live",
    "variant":"1d704fde-6dea-4633-ae04-bf40dad74e95"
  }]
}
```

Typical Response

In the example, the operation was successful and returned this response body with the channel details:

```
{
  "Description": null,
  "Identifier": "88ea9c60-8ccf-4908-929b-6993b6680c94",
  "Name": "RTMP_Test",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "c035c768-50da-41d6-ae61-49b3f129326c",
  "Packages": [
    {
      "Description": null,
      "Identifier": "35d75893-5e97-499a-a5fe-e510dfd22dab",
      "Name": "Rtmp Outputs",
      "Details": [
        {
          "Name": "Destination",
          "Value": "Custom URL(s)"
        },
        {
          "Name": "Destination Username",
          "Value": "******"
        },
        {
          "Name": "Destination Password",
          "Value": "******"
        },
        {
          "Name": "URL - Variant '540p-X-AVC'",
          "Value": "rtmp://qa-xbuster.telestream.net:1935/live"
        },
        {
          "Name": "URL - Variant '720p-X-AVC'",
          "Value": "rtmp://qa-xbuster.telestream.net:1935/live"
        },
        {
```

```
          "Name": "Stream Name - Variant '540p-X-AVC'",
          "Value": "Test2"
        },
        {
          "Name": "Stream Name - Variant '720p-X-AVC'",
          "Value": "Test1"
        },
        {
          "Name": "URL Identifiers",
          "Value": "1d704fde-6dea-4633-ae04-bf40dad74e95;a02ab97b-
a318-4061-bf33-ccbf0c3a1414"
        },
        {
          "Name": "Stream Name Identifiers",
          "Value": "1a858716-4415-4fed-a5a3-b0fdaa69e444;f005275d-
1787-47fb-9521-a30cced7de91"
        }
      ],
      "Package": "3482ea4d-5215-4357-ac98-a77aae32dfad"
    }
  ]
}
```


ConfigureRtmpToCustomUrlChannel

The purpose of this POST operation is to update the configuration of an RTMP channel you've created with a custom URL, after the channel has been added using *AddRtmpToCustomUrlChannel*. This operation also can be used to update any of the attached Variant URI configurations (the stream name and the URI).

The GUIDs in this operation identify the component you are updating; the remaining parameters modify the value of the component's key pair.

Note: This operation does not add or delete Variants—it only modifies existing Variants.

URL Format

```
http://<host>:<port>/Channels/ConfigureRtmpToCustomUrlChannel
```

Body Format

```
{
  "channelIdentifier":"CHANNEL GUID",
  "channelName":"CHANNEL NAME",
  "userName":"USER NAME",
  "password":"PASSWORD",
  "variantUris":[{
    "variant":"VARIANT GUID",
    "streamName":"STREAM NAME",
    "uri":"URI"
  }]
}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)
and
GetVariants (variant GUID)

Parameters

Parameter	Description
channelIdentifier	GUID; identifies the target Live Stream channel you are updating.
channelName (optional)	String; the name of the Live Stream RTMP channel.
userName	String; registered user name.
password	String; password associated with the user name.
variantUris	Array; One or more permitted to specify each variant you are updating.
variant	GUID; identifies the variant you are updating.
streamName (optional)	String; name of the stream.
uri (optional)	String; URI of the stream for this variant.

Results

On success, *ConfigureRtmpToCustomUrlChannel* returns a record with the channel details.

Example

In this example, the channel name is being updated:

```
http://myliveserver:18000/Channels/ConfigureRtmpToCustomUrlChannel
```

```
{
  "channel":d1657fad-80b6-06c0-8558-4c73d06711e8,
  "channelName":"UpdatedRTMPChannel",
}
```

Typical Response

In the example, the operation was successful and returned this response body with the updated details:

```
{
  "Description": null,
  "Identifier": "cfb06763-c027-4fc0-9a6a-c786c3f414ff",
  "Name": "UpdatedRTMPChannel",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "c035c768-50da-41d6-ae61-49b3f129326c",
  "Packages": [
    {
      "Description": null,
      "Identifier": "35d75893-5e97-499a-a5fe-e510dfd22dab",
      "Name": "Rtmp Outputs",
      "Details": [
        {
          "Name": "Destination",
          "Value": "YouTube Live RTMP"
        },
        {
          "Name": "Server URL",
          "Value": "rtmp://test1"
        },
        {
          "Name": "Stream name/key",
          "Value": "******"
        }
      ],
      "Package": "3482ea4d-5215-4357-ac98-a77aae32dfad"
    }
  ]
}
```


AddRtmpToYouTubeChannel

The purpose of this POST operation is to connect a Live Stream Channel (including an RTMP Package) to an endpoint in a YouTube Live account, using the stream name via the Live Stream's RTMP publishing method.

URL Format

`http://<host>:<port>/Channels/AddRtmpToYouTubeChannel`

Body Format

```
{
  "channelName":"CHANNEL NAME",
  "package":"PACKAGE GUID",
  "serverUri":"SERVER URI",
  "streamName":"STREAM NAME"
}
```

Operation Sequence

Execute the following operations to obtain the required values for this operation:

GetChannels (channel name)
and
GetPackages (Package GUID)

Parameters

Parameter	Description
channelName	String; identifies the Live Stream RTMP channel you are connecting to YouTube Live.
package	GUID; identifies the Live Stream Package associated with this channel to stream.
serverUri (optional)	URI string; URI of the end point for this stream, obtained from the YouTube Live dashboard Primary Server URL field. For example: `rtmp://a.rtmp.youtube.com/live1`.
streamName (optional)	String; the Stream name, obtained from the YouTube Live dashboard Stream Name field.

Results

On success, *AddRTMPtoYouTubeChannel* returns a response body with the details.

Example

http://10.0.25.158:18000/Channels/`AddRtmpToYouTubeChannel`

```
{
  "channelName":"YTL1",
  "package":"3482ea4d-5215-4357-ac98-a77aae32dfad",
  "serverUrl":"rtmp://a.rtmp.youtube.com/live1",
  "streamName":"w81b-5x37-1f8p-ck11"
}
```

Typical Response

In the example, the operation was successful and returned this response body with the channel details:

```
{
  "Description": null,
  "Identifier": "cfb06763-c027-4fc0-9a6a-c786c3f414ff",
  "Name": "YTL1",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "c035c768-50da-41d6-ae61-49b3f129326c",
```

```
  "Packages": [
    {
      "Description": null,
      "Identifier": "35d75893-5e97-499a-a5fe-e510dfd22dab",
      "Name": "Rtmp Outputs",
      "Details": [
        {
          "Name": "Destination",
          "Value": "YouTube Live RTMP"
        },
        {
          "Name": "Server URL",
          "Value": "rtmp://a.rtmp.youtube.com/live1"
        },
        {
          "Name": "Stream name/key",
          "Value": "******"
        }
      ],
      "Package": "3482ea4d-5215-4357-ac98-a77aae32dfad"
    }
  ]
}
```


ConfigureRtmpToYouTubeChannel

The purpose of this POST operation is to update the settings of an RTMP channel you've set up for YouTube, after it has been added using *AddRtmpToYouTubeChannel*.

URL Format

```
http://<host>:<port>/Channels/ConfigureRtmpToYouTubeChannel
```

Body Format

```
{
  "channel":"CHANNEL GUID",
  "channelName":CHANNEL NAME,
  "serverUri":"SERVER URL",
  "streamName":"STREAM NAME"
}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetChannels (channel GUID)

Parameters

Parameter	Description
channelIdentifier	GUID; identifies the target Live Stream channel you are updating.
channelName (optional)	String; the name of the Live Stream RTMP channel you are updating.
serverUri (optional)	URI string; URL of the end point for this stream that you are updating—the YouTube server URI from the YouTube Live dashboard. For example: `rtmp://endpoint.youtube.com.`
streamName (optional)	String; the Stream name/key from the YouTube Live dashboard.

Results

On success, *ConfigureRtmpToYouTubeChannel* returns a record with the channel details.

Example

In this example, the channel name is being updated:

```
http://10.0.25.153:18000/Channels/ConfigureRtmpToYouTubeChannel
```

```
{
  "channel":d1657fad-80b6-06c0-8558-4c73d06711e8,
  "channelName":"UpdatedRTMPChannel",
}
```

Typical Response

In the example, the operation was successful and returned this response body with the updated details:


```
{
  "Description": null,
  "Identifier": "cfb06763-c027-4fc0-9a6a-c786c3f414ff",
  "Name": "UpdatedRTMPChannel",
  "Active": false,
  "ActiveSegments": [],
  "Assignments": [],
  "CalendarEvents": {
    "Briefs": []
  },
  "Machine": "c035c768-50da-41d6-ae61-49b3f129326c",
  "Packages": [
    {
      "Description": null,
      "Identifier": "35d75893-5e97-499a-a5fe-e510dfd22dab",
      "Name": "Rtmp Outputs",
      "Details": [
```

```
        {
          "Name": "Destination",
          "Value": "YouTube Live RTMP"
        },
        {
          "Name": "Server URL",
          "Value": "rtmp://test1"
        },
        {
          "Name": "Stream name/key",
          "Value": "******"
        }
      ],
      "Package": "3482ea4d-5215-4357-ac98-a77aae32dfad"
    }
  ]
}
```

Source Operations

You can use the Lightspeed Source web service operations in a program to identify Live servers by GUID, obtain a list of sources to operate on, and insert various SCTE-35 triggers and ID3 frame tags into a source. Source operations can be used in both Capture and Stream programs.

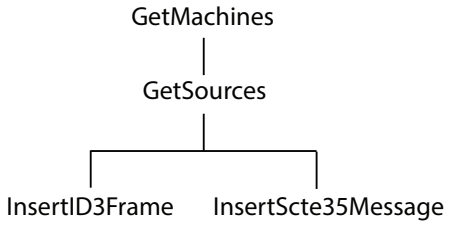
- GetMachines
- GetSources
- GetFileLoopSource
- GetMpeg2TransportStreamSource
- GetRtmpSource
- GetSdiSource
- GetSlateSource
- GetST2110Source
- GetSourceAffinities
- GetSourceDetails
- GetSourceTimecode
- GetSystemAffinity
- CreateFileLoopSource
- CreateMpeg2TransportStreamSource
- CreateRtmpSource
- CreateSlateSource
- InsertID3Frame
- InsertScte35Message
- SetSingleLinkToLoopThru
- SetSingleLinkToNoLoopThru
- SetQuadLinkToLoopThru
- SetQuadLinkToNoLoopThru
- UpdateFileLoopSource

- UpdateMpeg2TransportStreamSource
- UpdateRtmpSource
- UpdateSdiSource
- UpdateSlateSource
- UpdateSt2110Source

Note: All Source operations at located at the root level: `http://<host>:<port>/`. Sources operations do not require authentication. The default port to accessing the Source API operations is 15000.

Live Source Component Hierarchy

When operating on sources, you must first obtain a list of machines and then identify the machine you want to access by GUID: Machine > Source. Next, obtain a list of Sources and then identify the source you want to access by GUID.



Note: To display help for Source operations, enter `http://<host>:<port>/help`.

GetMachines

The purpose of this GET operation is to identify the Live server (or in the case of a group, all of the Live servers in the group) by GUID.

This operation is a prerequisite for other operations which require a machine GUID.

To execute this operation, use the Windows domain name or the IP address of the Live server or any Live server in the group.

GetMachines has the following format:

```
http://<host>:<port>/GetMachines
```

Operation Sequence

No other operations are required before you can execute this operation.

Results

Upon success, *GetMachines* returns an array, with a record of the Live server (or, in the case of a group, each Live server) GUID and Name.

Example

```
http://10.9.9.9:15000/GetMachines
```

Typical Response

In this response, three server records are listed. You can extract each machine's GUID from the Identifier and use it to connect, and monitor or control its resources. Of course, in a standalone system, only one record is returned.


```
[
  {
    "Identifier":"89d2be4b-be21-4838-a551-522cce299fbe",
    "Name":"LL-PM-1",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"4bd2be89-2c24-3947-a432-484bca2387fba",
    "Name":"LL-PM-2",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"43b2ff5b-ac29-48573-c443-567cad734efa",
    "Name":"LL-PM-3",
    "Description":null,
    "Details":[]
  }
]
```

GetSources

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.


```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetFileLoopSource

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetMpeg2TransportStreamSource

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetRtmpSource

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetSdiSource

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetSlateSource

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetST2110Source

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetSourceAffinities

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.


```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

GetSourceDetails

The purpose of this GET operation is to obtain ednote what? of the specified source.

GetSourceDetails has the following format:

```
http://<host>:<port>/GetSourceDetails?sourceidentifier={SOURCE
GUID}
```

GetSourceDetails returns a response that includes the status of the source process and corresponding data for the video and audio signals and captions.

GetSourceDetails has the following format:

```
http://<host>:<port>/record/
GetSourceDetails?sourceIdentifier=(Source GUID)
```

Operation Sequence

Execute the following operations to obtain the required Source GUID for this operation:

GetMachines > machine GUID

GetSources (machine GUID) > Source GUID

Required Parameter

Parameter	Description
source Identifier	GUID; string that identifies a specific source on a specific server.

Results

On success, *GetSourceDetails* returns a set of elements by name that describes each metadata item's current state, which you can parse to utilize in your program.

Example

Here is an example of a *GetSourceDetails* operation:

```
http://LS-SVR:17001/GetSourceDetails?sourceIdentifier=6741b1bd-
9461-4e18-9002-c133xeb72acc
```

Typical Response

In this response, the Source on the target server returns details about its current state .

```
<[
  {
    "name": "ValidSignal",
    "id": "512f4eaa-2384-4816-bab6-739272072f9e",
    "value": "True"},
  {
    "name": "SDI #",
```

```
    "id": "47156db9-f434-4664-adbb-3c339cb4198f",
    "value": "1"},
  {
    "name": "Resolution",
    "id": "e9e57e96-d46a-4c24-8a52-5b8fede3f4b9",
    "value": "1920x1080i"},
  {
    "name": "Frame Rate",
    "id": "6c2dcf59-1955-463b-9b45-f2250246ee7d",
    "value": "25"},
  {
    "name": "Bit Depth",
    "id": "8334d111-6776-46b7-ab11-ddd4f8d73d1e",
    "value": "10"},
  {
    "name": "Audio Channels",
    "id": "45f4a40e-d4fc-411e-81e7-f1ebfbaf2ab6",
    "value": "16"},
  {
    "name": "Captions",
    "id": "755b3a5f-3c9a-41df-9311-02c69f421233",
    "value": "No"},
]>
```

GetSourceTimecode

The purpose of this GET operation is to obtain the timecode of a source with timecode.

GetSourceTimecode has the following format:

```
http://<host>:<port>/GetSourceTimecode?sourceidentifier={SOURCE
GUID}
```

Operation Sequence

Execute the following operations to obtain the required machine GUID and source GUID for this operation:

GetMachines > (machine GUID)

GetSources > (source GUID)

Required Parameter

Parameter	Description
sourceidentifier	GUID; string that identifies a specific SDI source.

Results

On success, *GetSourceTimecode* returns the timecode for the specified source on the Live server.

Example

```
http://10.9.9.9:15000/GetSourceTimecode?
sourceidentifier=4ad20640-a5d8-45d1-a035-3a38227f21d5
```

Typical Response

In this response, the source timecode is returned for your use.

```
"00:46:36;07"
```

GetSystemAffinity

The purpose of this GET operation is to identify all of the video sources that are available on the target Live server, and return them in a list for further use.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

GetSources has the following format:

```
http://<host>:<port>/GetSources?machine={MACHINE GUID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID)

Required Parameter

Parameter	Description
machine	GUID; string that identifies a specific Live server.

Results

On success, *GetSources* returns a set of records; one for each source in the Live server.

Example

```
http://10.9.9.9:15000/GetSources?
machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

In this response, all of the Sources that are operational on the target server are listed along with their Identifier and Name values, which you can use to query and use them.

```
[
  {
    "Identifier":"4ad20640-a5d8-45d1-a035-3a38227f21d5",
    "Name":"LL-PM - SDI Input 1 to Output 3",
    "Description":null,
    "Details":[]
  },
  {
    "Identifier":"f6b89737-5c4a-44b5-98df-c8b7b2ffc8a3",
    "Name":"LL-PM - SDI Input 2",
    "Description":null,
    "Details":[]
  },
]
```

CreateFileLoopSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

CreateMpeg2TransportStreamSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

CreateRtmpSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

CreateSlateSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

InsertID3Frame

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

Body (text/plain MIME type with base-64-encoded binary data)

PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxNYXJrZXI+DQ
o8TWFya2VySUQ+MHgxNWQxZTg8L01hcmtlcklEPg0KPFByZXJvbGxITlM+MDwvUHJl
cm9sbEhOUz4NCjxTcGxpY2U+DQo8RXZlbnRJRD4xNDI5OTkyPC9FdmVudElEPg0KPE
91dE9mTmV0d29yaz4wPC9PdXRPZk5ldHdvcms+DQo8UHJvZ3JhbVNwbGljZT4xPC9Q
cm9ncmFtU3BsaWNlPg0KPER1cmF0aW9uPjA8L0R1cmF0aW9uPg0KPFNwbGljZUltbW
VkaWF0ZT4wPC9TcGxpY2VJbW1lZGlhdGU+DQo8VGltZVNwZWNpZmllZD4xPC9UaW1l
U3BlY2lmaWVkPg0KPEF1dG9SZXR1cm4+MDwvQXV0b1JldHVybj4NCjwvU3BsaWNlPg
0KPFByb2dyYW0+DQo8UHJvZ3JhbUlEPjEyNDwvUHJvZ3JhbUlEPg0KPEF2YWlsTnVt
PjA8L0F2YWlsTnVtPg0KPEF2YWlsRXhwZWN0ZWQ+MDwvQXZhaWxFeHBlY3RlZD4NCj
wvUHJvZ3JhbT4NCjxUaW1lPg0KPFNwbGljZVRpbWVITlM+MTAxOTE4MjEzMDAwMDA8
L1NwbGljZVRpbWVITlM+DQo8U3BsaWNlRGF0ZVRpbWU+MjAxNy0xMC0yN1QwMDoxNz
oyM1o8L1NwbGljZURhdGVUaW1lPg0KPER1cmF0aW9uSE5TPjA8L0R1cmF0aW9uSE5T
Pg0KPC9UaW1lPg0KPC9NYXJrZXI+

InsertScte35Message

The purpose of this GET operation is to insert a SCTE-35 message in the source. If a time is not specified, the message will be inserted immediately.

Note: Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertScte35Message has the following format:

```
http://<host>:<port>/
InsertScte35Message?source={SOURCE}&scte35Message={SCTE35MESSAGE}&
time={TIME}&eventID={EVENTID}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
scte35 Message	String; keyword that defines the SCTE-35 message. You can enter a string representing the SCTE segmentation type ID in plain text or as a string representation of the hex value. Capitalization is ignored, and with or without spaces. For example: "Program Start", "programstart", and "0x10" all refer to the same message; creating a SCTE trigger and setting its segmentation type ID to 0x10 (16 in base 10). For example: scte35Message=`"Program Start"` See the SCTE Commands table below.
timeCode (optional)	Timecode; time code value in the source at which to insert the SCTE-35 message. If a time code is not specified, the message is inserted immediately. For example: `timeCode=01:03:15:00@29.97`
eventID	Integer value; a 32-bit unsigned integer value which specifies the ID of the message.

SCTE-35 Commands

Command	Hex Value	Text Value
Program Start	0x10	ProgramStart
Program End	0x11	ProgramEnd
National Break Start	0x30	ProviderAdStart
National Break End	0x31	ProviderAdEnd
Local Break Start	0x32	DistributorAdStart
Local Break End	0x33	DistributorAdEnd
Ad Break Start	0x34	AdBreakStart
Ad Break End	0x35	AdBreakEnd

Results

Upon success, *InsertScte35Message* adds the specified message and its type value in the body, to the source at the indicated time frame (or immediately) and returns a record with the string “Success”:

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertScte35Message?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
scte35Message="Program Start"&eventID=4967295
```

SetSingleLinkToLoopThru

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
parm1	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

SetSingleLinkToNoLoopThru

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
parm1	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

SetQuadLinkToLoopThru

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
parm1	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

SetQuadLinkToNoLoopThru

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
parm1	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateFileLoopSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateMpeg2TransportStreamSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateRtmpSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateSdiSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateSlateSource

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```

UpdateSt2110Source

This POST operation inserts an ID3 tag with a single PRIV frame and the specified type and value in the source at the specified timecode. If a time code is not specified, the tag is inserted immediately. Sources are defined for a specified hardware port on a specific server. Thus, the host that you specify must be the server where the Source was added.

InsertID3Frame has the following format:

```
http://<host>:<port>/InsertID3Frame?
source={SOURCE GUID}&type={FRAME TYPE}&timeCode={TIMECODE VALUE}

Body <MIME TYPE>: {type value>}
```

Operation Sequence

Execute the following operation to obtain the required GUID for this operation:

GetMachines (machine GUID) > *GetSources* (Source GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific source on the Live server. For example: `source=442082bf-bde8-44b5-9bce-832b6d0fd885`
type	String; the type of PRIV frame. For example: `type=com.cisco.streaming.SplicePoint.0`
timeCode (optional)	Timecode; time code value in the source at which to insert the tag. If a time code is not specified, the tag is inserted immediately. For example: `source=01:03:15:00@29.97`

Required Post Body

Type Value	String; appropriately-encoded string that is the value for the type key-pair value inserted into the video. This operation supports these MIME types: - text/plain (base 64-encoded binary data) - application/xml or text/xml (un-encoded XML) - application/octet-stream (raw binary data).

Results

Upon success, *InsertID3Frame* adds an ID3 tag with a PRIV frame of the specified type and value to the source at the indicated time frame (or immediately) and returns a record with the string "`Success`":

```
[
  {
  "Success"
  }
]
```

Example

```
http://10.9.9.9:15000/InsertID3Frame?
source=442082bf-bde8-44b5-9bce-832b6d0fd885&
type="com.cisco.streaming.SplicePoint.0"
```