

Lightspeed Live Web API Reference



Live Capture V2.0.3 ComponentPac 7.0.5

Live Stream V1.2

Copyrights and Trademark Notices

Copyright © 2017 by Telestream, LLC. All rights reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, altered, or translated into any languages without the written permission of Telestream. Information and specifications in this document are subject to change without notice and do not represent a commitment on the part of Telestream.

Telestream. Telestream, CaptionMaker, Episode, Flip4Mac, FlipFactory, Flip Player, Lightspeed, ScreenFlow, Switch, Vantage, Wirecast, Gameshow, GraphicsFactory, MetaFlip, and Split-and-Stitch are registered trademarks and MacCaption, e-Captioning, Pipeline, Post Producer, Tempo, TrafficManager, VidChecker, and VOD Producer are trademarks of Telestream, LLC. All other trademarks are the property of their respective owners.

Adobe. Adobe® HTTP Dynamic Streaming Copyright © 2014 of Adobe Systems All right reserved.

Apple. QuickTime, MacOS X, and Safari are trademarks of Apple, Inc. Bonjour, the Bonjour logo, and the Bonjour symbol are trademarks of Apple, Inc.

Avid. Portions of this product Copyright 2012 Avid Technology, Inc.

Dolby. Dolby and the double-D symbol are registered trademarks of Dolby Laboratories.

Fraunhofer IIS and Thomson Multimedia. MPEG Layer-3 audio coding technology licensed from Fraunhofer IIS and Thomson Multimedia.

Google. VP6 and VP8 Copyright Google Inc. 2014 All rights Reserved.

MainConcept. MainConcept is a registered trademark of MainConcept LLC and MainConcept AG. Copyright 2004 MainConcept Multimedia Technologies.

Manzanita. Manzanita is a registered trademark of Manzanita Systems, Inc.

MCW. HEVC Decoding software licensed from MCW.

MedialInfo. Copyright © 2002-2013 MediaArea.net SARL. All rights reserved.

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 HOLDER 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.

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.

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.

ZLIB. Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler.



Other brands, product names, and company names are trademarks of their respective holders, and are used for identification purpose only.

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](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](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](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: 226202

Date: August 02, 2017

Contents

Overview 15

Lightspeed Live Capture Web API	16
Lightspeed Live Stream Web API	17
Web Service Operation & Response Formats	18
Operation Keyword Terms	19
Operation Case Sensitivity	19
Use of GUIDs in Operations	20
Live Web Service Operation Responses	21
Topics	21
Required Capture Web Service Response Elements	21
Optional Capture Web Service Response Elements	22
Web Service Error Responses	23
Avoid Using Reserved Characters in Value Strings	24
Adjusting Chrome Browser Prediction Settings	25

Capture Operations 27

Start	28
Optional Parameters	28
Example	29
Typical Start Operation Response	30
Modify	31
Parameters	31
Example	31
Typical Modify Operation Response	32
MarkOut MarkIn	33
Optional Parameters	33
Example	33
Typical MarkIn/MarkOut Operation Response	34
EditOut EditIn	35
Optional Parameters	35
Example	36
Typical EditIn/EditOut Operation Response	36

Message	37
Parameters	37
Example	37
Typical Message Operation Response	38
Stop	39
Optional Parameters	39
Example	39
Status	41
Optional Parameters	41
Example	41
Typical Status Operation Response	41

Stream Operations 43

Introduction	44
Limits of the API	44
Using Live Stream Groups via the API	44
Using Shared vs. Dedicated Components	44
Brevity in Examples	44
Live Stream Component Hierarchy	44
Obtaining Help for Streaming Operations	46
Displaying a List of Live Stream Services	46
Displaying the Operations of a Specific Live Stream Service	46
Displaying Operation Details	46
System Operations	48
GetMachines	49
Operation Chain	49
Results	49
Example	49
Typical Response	49
GetServiceEndpoints	51
Operation Chain	51
Results	51
Example	51
Typical Response	51
GetSystemSettings	53
Results	53
Example	53
Typical Response	53
GetSystemLogs.zip	54
Results	54
Sources Operations	55
AddRtmpSource	56
Operation Chain	56
Required Parameters	56
Results	56
Example	56
Typical Response	57

AddTransportStreamSource	58
Operation Chain	58
Parameters	58
Results	58
Example	59
Typical Response	59
GetSources	60
Operation Chain	60
Parameters	60
Results	60
Example	60
Typical Response	60
GetSource	62
Operation Chain	62
Parameters	62
Results	62
Example	62
Typical Response	62
GetSourceTracks	65
Operation Chain	65
Parameters	65
Results	65
Example	65
Typical Response	65
GetSourceTrack	66
Operation Chain	66
Parameters	66
Results	66
Example	66
Typical Response	66
GetTextTracks	68
Operation Chain	68
Parameters	68
Results	68
Example	68
Typical Response	68
GetTextTrack	69
Operation Chain	69
Parameters	69
Results	69
Example	69
Typical Response	69
GetSourceThumbnail	70
Operation Chain	70
Parameters	70
Results	70
Example	70

Programs Operations	71
GetPrograms	72
Operation Chain	72
Results	72
Example	72
Typical Response	72
GetProgram	73
Operation Chain	73
Parameters	73
Results	73
Example	73
Typical Response	73
GetRenditions	75
Operation Chain	75
Parameters	75
Results	75
Example	75
Typical Response	75
GetRendition	76
Operation Chain	76
Parameters	76
Results	76
Example	76
Typical Response	76
GetSegments	78
Operation Chain	78
Parameters	78
Results	78
Example	78
Typical Response	78
GetSegment	80
Operation Chain	80
Parameters	80
Results	80
Example	80
Typical Response	80
GetMaterials	82
Operation Chain	82
Parameters	82
Results	82
Example	82
Typical Response	82
GetMaterial	84
Operation Chain	84
Parameters	84
Results	84
Example	84
Typical Response	84

Encoders Operations	86
GetEncoders	87
Operation Chain	87
Results	87
Example	87
Typical Response	87
GetEncoder	89
Operation Chain	89
Parameters	89
Results	89
Example	89
Typical Response	89
GetStreams	91
Operation Chain	91
Parameters	91
Results	91
Example	91
Typical Response	91
GetStream	93
Operation Chain	93
Parameters	93
Results	93
Example	93
Typical Response	93
Packages Operations	95
GetPackages	96
Operation Chain	96
Results	96
Example	96
Typical Response	96
GetPackage	97
Operation Chain	97
Parameters	97
Results	97
Example	97
Typical Response	97
GetVariants	99
Operation Chain	99
Parameters	99
Results	99
Example	99
Typical Response	99
GetVariant	101
Operation Chain	101
Parameters	101
Results	101
Example	101
Typical Response	101

Channels Operations	103
GetChannels	105
Operation Chain	105
Results	105
Example	105
Typical Response	105
GetChannel	106
Operation Chain	106
Parameters	106
Results	106
Example	106
Typical Response	106
GetChannelOutputLocations	108
Operation Chain	108
Parameters	108
Results	108
Example	108
Typical Response	108
GetChannelThumbnail	110
Operation Chain	110
Parameters	110
Results	110
Example	110
StartChannel	111
Operation Chain	111
Parameters	111
Results	111
Example	111
Typical Response	111
StopChannel	112
Operation Chain	112
Parameters	112
Results	112
Example	112
Typical Response	112
GetCalendarEvents	113
Operation Chain	113
Parameters	113
Results	113
Example	113
Typical Response	113
GetCalendarEvent	114
Operation Chain	114
Parameters	114
Results	114
Example	114
Typical Response	114

AddCalendarEvent	116
Operation Chain	116
Parameters	116
Results	116
Example	116
Typical Response	117
DeleteCalendarEvent	118
Operation Chain	118
Parameters	118
Results	118
Example	118
Typical Response	118
GetActiveSegment	119
Operation Chain	119
Parameters	119
Results	119
Example	119
Typical Response	119
SetActiveSegment	120
Operation Chain	120
Parameters	120
Results	120
Example	120
Typical Response	121
SetActiveSegmentAtTime	122
Operation Chain	122
Parameters	122
Results	122
Example	122
Typical Response	123
GetMachineStatistics	124
Operation Chain	124
Parameters	124
Results	124
Example	124
Typical Response	124
GetChannelStatistics	125
Operation Chain	125
Parameters	125
Results	125
Example	125
Typical Response	125
GetNewFacebookVerificationDeviceCode	127
Results	127
Example	127
Typical Response	127

AddFacebookChannel	128
Operation Chain	128
Parameters	128
Results	128
Example	129
Typical Response	129
ConfigureFacebookChannel	130
Operation Chain	130
Parameters	130
Results	131
Example	131
Typical Response	131

Overview

The two Lightspeed Live Web APIs—Capture and Stream —enable you to monitor your Lightspeed Live system within a broader, Web services-based system or create your own customized monitoring system. You can also create a Web services-based system to control streaming and capture beyond the functionality or capability of the general-purpose Vantage Capture and Stream web applications provided to meet your organization's operational requirements.

- [Lightspeed Live Capture Web API](#)
- [Lightspeed Live Stream Web API](#)
- [Web Service Operation & Response Formats](#)
- [Avoid Using Reserved Characters in Value Strings](#)
- [Adjusting Chrome Browser Prediction Settings](#)

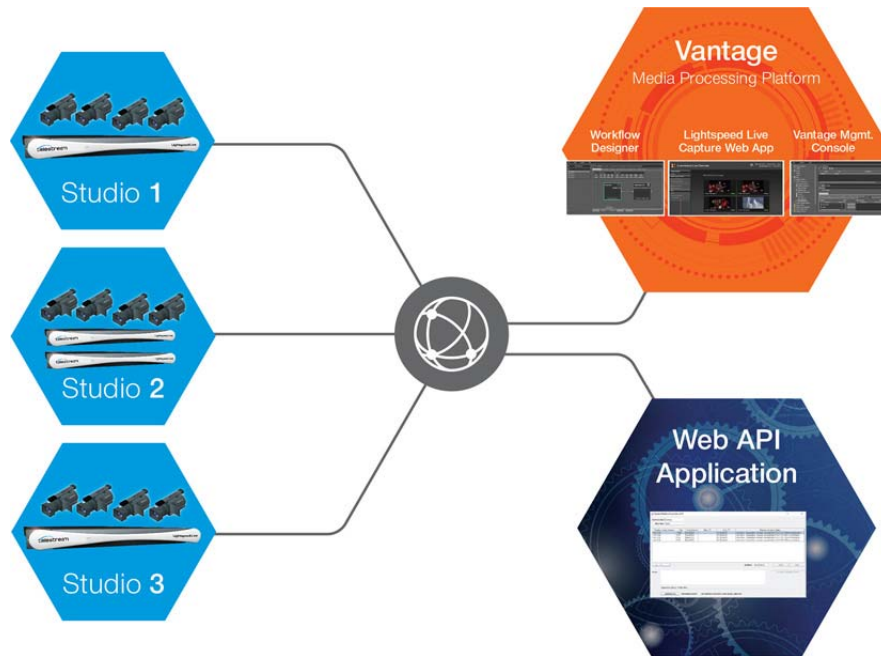
Note: This reference assumes that the programming environment being used by the developer includes a library that abstracts the process of operation submission and responses through the HTTP protocol.

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 Capture Web API

Lightspeed Live Capture enables recording of live streaming media in a Vantage workflow to be controlled manually via the Lightspeed Live Capture Web application or through the HTTP Web Service *Capture Operations* described in this reference.



Lightspeed Live Capture workflows can be configured to publish a Web Service that enables media clips to be recorded using this HTTP-based Live Capture Web Services operation set. When a Capture workflow is activated in Workflow Designer, the Web Service begins listening 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.)

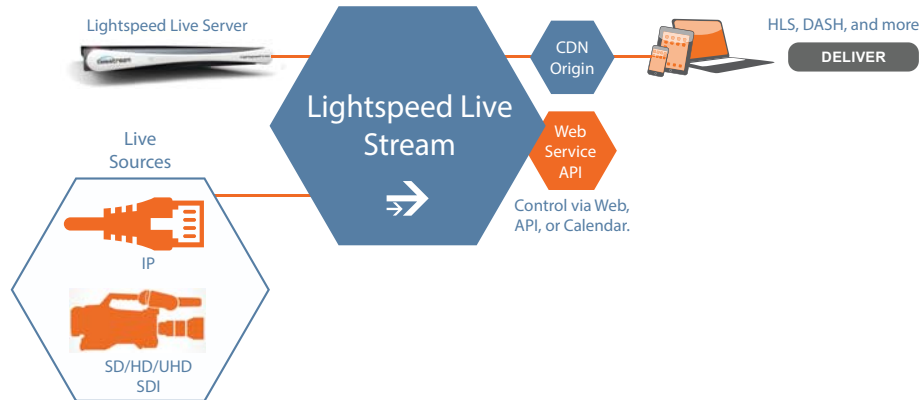
Note: See the Lightspeed Live Guide for details on enabling the Web Service.

Your custom client program can control any given SDI input device on the Lightspeed server (camera, deck, etc.) by communicating with a target Vantage workflow—which in turn is configured for a specific SDI input and a specific Web service port. Your program may be designed to target a specific SDI input device by communicating on a specific port (thus, a specific workflow and SDI device) on the domain, or it may be more broadly-designed (using 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 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 Stream Web API

Lightspeed Live Stream 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



Note: See the Lightspeed Live Guide for details on enabling the Web Service.

Lightspeed Live Stream enables you to transcode source files in real-time, streaming them via the Lightspeed Live Streaming Web application or through the HTTP Web Service [Stream Operations](#) described in this reference.

Web Service Operation & Response Formats

Most Lightspeed Live Web Service operations are invoked using an HTTP GET request in the following form:

```
http://<host>:18000/record/<operation>  
[?<parameter>=<value>[&<parameter>=<value>]]
```

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

Operations—those that alter operational data in the Live Stream server—are POST operations; such as *AddCalendarEvent*, a Stream operation.

Required and optional response elements are presented in [Live Web Service Operation Responses](#); examples are presented in each operation's topic.

Topics

- [Operation Keyword Terms](#)
- [Use of GUIDs in Operations](#)
- [Live Web Service Operation Responses](#)

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, as noted in the table following.

Note: Operation names and keywords in operations are not case-sensitive, although the keywords are represented in this guide using camel case for readability. 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. Multiple parameters are separated by an ampersand (&). Parameters in brackets ([]) are optional.

For example:

```
http://<host>:18000/record/start
[?<parameter>=<value>[&<parameter>=<value>]]
```

Here are the keyword terms you'll encounter:

Term	Description
host	The Windows domain name or the IP address of the Lightspeed Live Capture or Stream service you are targeting. For example: <i>localhost</i> <i>LightspeedServer</i> 192.168.1.23.
port	The TCP/IP port number assigned to the Web Service. For Live Capture, the port number is user-selectable and displayed in the Web Service configuration panel of the Capture action inspector in the target workflow. (See the Lightspeed Live Guide or the man page for the Capture action.) For Live Stream, the port number is fixed at 18000.
operation	Reserved word; the Web Service operation to execute.
parameter	A named parameter defined by the Web Service operation.
value	The value for the associated parameter.

Use of GUIDs in Operations

GUIDs (Globally Unique Identifier) are used in many operations to specifically target 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.

In the context of Lightspeed Live, the 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`.

Live Web Service Operation Responses

When you send an operation to a Lightspeed Live Web Service, the Web service executes the operation and sends a response appropriate to the operation and the result of the execution. The response format varies by operation and by the parameters passed.

Topics

- [Required Capture Web Service Response Elements](#)
- [Optional Capture Web Service Response Elements](#)
- [Web Service Error Responses](#)

Required Capture Web Service Response Elements

Lightspeed Live Capture Web Service responses consist of an HTTP status line (for example: *200 OK* or *404 Not Found*), HTTP headers, and an XML body consisting of one <Response> element, enclosing all other elements.

Most elements are always returned in every response from the Lightspeed Live Capture service. Some elements are returned only for specific operations, as noted:

Response	Description
Access-Control-Allow-Origin	For use only by Telestream.
End	The anticipated ending timecode of a clip. For example: 02 : 23 : 50 ; 00.
EngineTime	The current Lightspeed Live Capture timecode value. For example: 12 : 34 : 56 : 00.
MarkIn	The actual starting timecode of a clip (returned from Status and Stop operations with a UUID).
MarkOut	The actual ending timecode of a clip (returned from Status and Stop operations with UUID). For example: 01 : 23 : 45 ; 00.
Name	The name of the job. For example: LiveInOaklandConcert.
Start	The anticipated starting timecode of a clip. For example: 01 : 23 : 45 ; 00. The timecode type is determined by the configuration of the source input. The timecode can be Free Run, Computer Clock, LTC, or Input Source.

Response	Description
State	Keywords describing the current state of the clip: <i>Opened</i> —Currently capturing. <i>Waiting</i> —Connected and waiting for jobs to be queued or to start jobs in the queue. <i>Closed</i> —Capture is complete. <i>Cancelled</i> —The job was canceled before capture. <i>Failed</i> —The job failed during capture. Accompanied by the TransmitError response.
UUID	The unique identifier (GUID) assigned to a clip. For example: 5b1eb65c-3018-a4cf-8134-6e1c16b378a7.
XMLRevision	The XML response revision. For example: 2.

Optional Capture Web Service Response Elements

In addition to the required response elements, other optional elements may also be present in the response, depending on the operation and parameters you send:

Response	Description
ActionDuration (for Start Stop Status operations when UUID specified Status when UUID specified)	The estimated duration of the capture action, in seconds. Progress divided by Action Duration multiplied by 100 results in the percentage complete.
Channels	Integer; the number of audio channels in the clip.
EngineState	Keyword; the state of the Lightspeed Live Capture Engine: <i>Running</i> —Currently operating normally.
Excluding	Boolean; reports if frames are being excluded via the use of MarkOut In or EditOut In operations. Mark operations are not used on TIFO files. Excluding is <i>false</i> upon initial record and after an In operation. Excluding is <i>true</i> after an Out operation.
FPS (for Start Stop Status operations when UUID specified Status when UUID specified)	The frame rate of the media being captured.
FrameRate	Real; the clip's frame rate.
HorizontalResolution	Integer; the clip's horizontal resolution.
PercentCompleted	Not utilized in this version.

Response	Description
Progress (for Start Stop Status operations when UUID specified Status when UUID specified)	Elapsed progress (in seconds) of the associated capture.
Error	String; reports a detailed error during a Failed state. For example: "Writer stalled. This is often caused by the writer not being able to keep up with the reader due to I/O limitations."
VerticalResolution	Integer; the clip's vertical resolution

Web Service Error Responses

In the event of an error when an operation attempts to execute, the Web service returns an error message. It varies, depending on the operation and parameters you send.

Here is the format of an error response:

```
<Response>
  <Error>[Error message]</Error>
  <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
</Response>
```

For example:

```
<Response>
  <Error>Unable to start recording. Error = Deck control not
available</Error>
  <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
</Response>
```

Avoid Using Reserved Characters in Value Strings

In the Capture and Stream API operations, Telestream recommends that you do not use the following reserved characters in any of the Value element attributes.

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

Certain characters are omitted; others are changed to a space character. In some circumstances, the following error is displayed: "Request Error - The server encountered an error processing the request. See server logs for more details."

Adjusting Chrome Browser Prediction Settings

The Prediction Service settings in the Chrome web browser may cause issues when issuing Lightspeed Live Web API operations directly from Chrome. Chrome may attempt to complete the URL address and associated arguments by using its own prediction method.

To prevent possible errors, go to Chrome > Settings > Show advanced settings > Privacy, and un-check the following options:

- Use a prediction service to help complete searches and URLs typed in the address bar or the app launcher search box
- Use a prediction service to load pages more quickly.

Capture Operations

You can use the following Lightspeed Live Capture Web Service operations to control capture operations and monitor them:

- [Start](#)
- [Modify](#)
- [MarkOut | MarkIn](#)
- [EditOut | EditIn](#)
- [Stop](#)
- [Status](#)

Note: Each operation includes a brief description, including the format of the operation, an example, and all required and optional parameters in a table. Finally, the response is presented with an example.

Start

The *Start* operation initiates the recording of a new clip.

Note: A maximum of 16 jobs per workflow can be queued in a clip list for processing. Jobs are automatically removed from the list when complete. Use the Status operation to determine the number of clips currently queued for processing.

This has the following format:

```
http://<host>:18000/record/start
```

or

```
http://<host>:18000/record/start  
[?<parameter>=<value>[&<parameter>=<value>]]
```

Note: If a start time is not specified, the clip begins recording immediately. If an end time is not specified the clip will record for 9 hours. For best results, specify an end time to avoid running out of disk space or design your application to actively monitor and control the recording time.

Parameters

Timecodes can be specified in either drop frame (for example: 01:00:10;00) or non-drop frame format (01:00:10:00) but they are treated identically (assuming you are using drop frame if the source is drop frame and non-drop frame if the source is non-drop frame).

Parameter	Description
duration (optional)	<p>Timecode; specifies the duration of the clip based on the number of frames captured. Default: 9 hours.</p> <p>For example: <code>duration=00:30:00:00</code>.</p> <p>If duration is used, the capture session continues until the file contains a number of frames equal to the duration parameter. This allows for timecode jumps that occur during capture.</p> <p>When used in conjunction with end, the end timecode overrides duration when the specified timecode (or one later in time) is detected.</p> <p>If no duration or end is specified, the clip records for 9 hours.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>

Parameter	Description
end (optional)	<p>Timecode; specifies the clip's <i>exclusive</i> end timecode. For example: <code>end=01:42:35:00</code>.</p> <p>The end timecode represents the timecode of the frame after the last frame of video. Capture will stop when the specified timecode (or one later in time) is detected. If neither a duration or end timecode is specified, the clip will record indefinitely.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>
folderPath (optional)	<p>String; specifies the path to a folder where the clip file should be written. The path must end with a <code>\</code>.</p> <p>If a folder path is not specified, the clip is recorded in the storage folder associated with the respective workflow. If a new folder is added to an existing path, the folder will be created. If the folder could not be created an error will be reported.</p> <p>Requires that the Capture action option for Create Output File(s) on Job Start is enabled.</p> <p>For example: <code>folderPath=[Drive letter]:\[folder path]\</code> or <code>\\[Share server]\[Share]\[Folder Path]\</code></p>
name (optional)	<p>String; specifies a name for the clip. If a clip name is not specified, then a random name is generated.</p> <p>Note: For this name to be utilized in a Lightspeed Live Capture workflow, you must include the Base Name token in the Primary or Secondary Output's file name creation pattern in the Filename Patter Editor (see the Capture Primary and Secondary Output Panels topic in the Lightspeed Live Guide for more detail). The appropriate file extension is automatically added to the file name. If a file exists, the web service will overwrite it.</p> <p>In addition, the Capture action must also have the Create output file(s) at job start option enabled.</p> <p>For example: <code>name=LiveInOakland_Cam7_16062016_1342_PQ_234</code>.</p> <p>Error: Returns an error if the name contains invalid characters.</p>
start (optional)	<p>Timecode; specifies the clip's <i>inclusive</i> starting timecode. For example: <code>start=01:12:35:00</code>.</p> <p>The start timecode represents the timecode of the first frame. If a start time is not specified, the clip begins recording immediately.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>
tape (optional)	<p>String; specifies a name for the Tape Name or Reel. The Tape Name/Reel is embedded directly into QuickTime and MXF OP1a files.</p> <p>For example: <code>tape=LiveInOakland_Cam7</code>.</p> <p>Error: Responds with an error if the tape name contains invalid characters.</p>

Example

In this example, recording will start immediately, and record for 10 minutes:

```
http://10.5.2.1:8080/record/start?duration=00:10:00:00
```

Typical Start Operation Response

Issuing this *Start* operation with a start timecode, duration, name, folderPath and tape parameters:

```
11-pm:17000/record/  
start?Start=01:00:00:00&duration=00:05:13:00&name=myClipName&  
folderpath=\\11-pm\d\temp&tape=myTapeName
```

resulted in the following response:

```
<Response>  
  <UUID>f23049e4-9d6f-4a51-bd79-8821bec7d604</UUID>  
  <PercentCompleted>0</PercentCompleted>  
  <Progress>0</Progress>  
  <ActionDuration>312.979333333333</ActionDuration>  
  <FPS>29.97002997003</FPS>  
  <Start>01:00:00;00@29.97</Start>  
  <End>01:05:13;00@29.97</End>  
  <MarkIn></MarkIn>  
  <MarkOut>01:05:13;00@29.97</MarkOut>  
  <Excluding>False</Excluding>  
  <Name>MyClipName.tifo</Name>  
  <Path>\\11-pm\D\temp\MyClipName.tifo</Path>  
  <Tape>myTapeName</Tape>  
  <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>12:26:52;20</EngineTime>  
  <XMLRevision>2</XMLRevision>  
</Response>
```

Modify

The *Modify* operation is used to modify the start time and/or duration of a scheduled event contained in the clip list.

Note: Clips you plan to modify must be in the Waiting state. Using a Modify operation on clips in the Opened (capturing) state cause the capture to immediately stop. To modify clips in the Opened state use a Stop operation with an end parameter.

This has the following format:

```
http://<host>:18000/record/modify
?start=[value]&duration=[value]&uuid=[value]
```

Parameters

The uuid parameter is required. Either the start or duration parameter must be supplied; both are permitted.

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) but they are treated identically (assuming you are using drop frame if the source is drop frame, and non-drop frame for non-drop frame source).

Parameter	Description
uuid (required)	<p>GUID; the unique identifier of the clip you wish to modify. For example: <code>uuid=21EC2020-3AEA-4069-A2DD-08002B30309D</code></p> <p>Error: Returns <i>404 Not Found</i> if the specified clip is not in the list.</p>
start (optional)	<p>Timecode; specifies the clip's new <i>inclusive</i> starting timecode. For example: <code>start=01:12:35:00</code>.</p> <p>The start timecode represents the timecode of the first frame to be captured.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>
duration (optional)	<p>Timecode; specifies the new duration of the clip. For example: <code>duration=01:12:35:00</code>.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>

Example

```
http://LS-SVR:17000/record/modify?uuid=27638f24-2bb1-4ce6-8816-5a05a5e05897&start=11:05:06:09
```

Typical Modify Operation Response

Issuing this *Modify* operation with a UUID and start timecode results in the following response:

```
<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>
  <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>
```

Issuing this *Modify* operation with a UUID and a duration timecode:

```
lsl-pm:17000/record/modify?
uuid=27638f24-2bb1-4ce6-8816-5a05a5e05897&duration=00:45:00:00
```

generated the following response:

```
<Response>
  <UUID>27638f24-2bb1-4ce6-8816-5a05a5e05897</UUID>
  <PercentCompleted>0</PercentCompleted>
  <Progress>0</Progress>
  <ActionDuration>3239.9733066</ActionDuration>
  <FPS>29.97002997003</FPS>
  <Start></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>
  <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>11:04:41;15</EngineTime>
  <XMLRevision>2</XMLRevision>
</Response>
```

MarkOut | MarkIn

Note: Supported for TIFO format only. Using MarkOut/MarkIn with formats other than TIFO may have unintended consequences. Processing TIFO frames marked as OUT requires Vantage Transcoder 2012.1 or later in your Vantage domain.

All TIFO frames after a *MarkOut* is executed will be marked with an OUT metadata tag. The TIFO decoder included in a Vantage workflow can ignore frames marked as OUT and prevent them from being passed to a compressor or other downstream processes.

If a filler file is specified in the workflow, markout frames are replaced by frames in the same ordinal position in the filler file until the *MarkIn* operation is received.

All frames are marked as IN after a *MarkIn* has been executed.

This has the following format:

```
http://<host>:18000/record/markout | markin
```

or

```
http://<host>:18000/record/mark[out | markin]
[?parameter=value[&parameter=value]]
```

Parameters

Timecodes can be specified in drop frame (for example: 01:00:10;00) or non-drop frame format (for example: 01:00:10:00). They are treated identically (assuming you are using drop frame for drop frame source and non-drop frame for non-drop frame source).

Parameter	Description
uuid (required)	GUID; the unique identifier of the clip on which to set the MarkOut/In point. For example: <code>uuid=21EC2020-3AEA-4069-A2DD-08002B30309D</code> Returns <i>404 Not Found</i> if the specified clip is not in the list.
timecode (optional)	Timecode; specifies the timecode in the future for a MarkOut/In point in the clip. For example: <code>timecode=01:12:35:00</code> . If a timecode is not specified the MarkOut/In point will be issued immediately. MarkOut timecode is INCLUSIVE; the frame will be marked OUT. MarkIn timecode is EXCLUSIVE; the frame will be marked IN. Error: Returns an error if timecode contains invalid characters.

Example

In this example, all frames in the recording will be tagged as OUT, beginning at timecode 01:00:10:00:

```
http://10.5.2.1:8080/record/markout?timecode=01:00:10:00
```

Typical MarkIn/MarkOut Operation Response

Issuing these *MarkIn* and *MarkOut* operations:

`http://lsl-pm:17000/record/markin` (or `markout`)
resulted in this response:

```
<Response>  
  <UUID>27638f24-2bb1-4ce6-8816-5a05a5e05897</UUID>  
  <State>Opened</State>  
  <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>  
  <EngineState>Running</EngineState>  
  <EngineTime>11:06:54;27</EngineTime>  
  <XMLRevision>2</XMLRevision>  
</Response>
```


EditOut | EditIn

EditOut/EditIn functionality differs, based on whether a Lightspeed Live Capture workflow's Filler feature has been enabled in the Capture action, and a file specified. This feature is identical to using the Out and In buttons in the Capture Portal with a manually-triggered capture.

If Filler is *not checked* in the Capture action, issuing an EditOut operation prevents source frames from being added to the captured file; issuing an EditIn operation returns to adding source frames to be written to the capture file.

If Filler is *checked* in the Capture action, issuing an *EditOut* enable frames from the Filler file to be written to the captured file. Issuing an *EditIn* returns to adding source frames to be written to the capture file. If the filler is longer than the edit out period, it will loop.

Note: See the Lightspeed Live Guide for details on enabling and using Filler.

These operations have the following formats:

```
<host>:18000/record/editin | editout
```

when used without a UUID or timecode; all clips in the list are updated.

```
<host>:18000/record/editin | editout  

?[parameter=value[&parameter=value]]
```

when used with a UUID and/or timecode.

Parameters

Timecodes can be specified in drop frame (for example: 01:00:10;00) or non-drop frame (for example: 01:00:10:00) but they are treated identically (assuming you are using drop frame if the source is drop frame and non-drop frame if the source is non-drop).

Operation	Description
uuid (required)	<p>GUID; the unique identifier of the clip on which to set the EditOut EditIn point. If not set, all clips are updated.</p> <p>For example: <code>uuid=21EC2020-3AEA-4069-A2DD-08002B30309D</code></p> <p>Returns <i>404 Not Found</i> if the specified clip is not in the list.</p>
timecode (optional)	<p>Timecode; specifies the timecode for an EditOut EditIn point in the clip. If a timecode is not specified, the EditOut EditIn operations will be issued immediately.</p> <p>For example: <code>timecode=01:12:35:00</code>.</p> <p>EditOut timecode is <i>inclusive</i>; the frame will be edited out.</p> <p>EditIn timecode is <i>exclusive</i>; the frame will be edited in.</p> <p>Error: Returns an error if timecode contains invalid characters.</p>

Example

In this example, Filler has not been checked in the workflow's Capture action. Thus, the media is not being written to disk at this moment. Also, because there is no uuid parameter in this operation, all clips in the list are updated:

```
http://10.5.2.1:8080/record/editin|editout?timecode=01:00:10:00
```

Typical EditIn/EditOut Operation Response

Issuing these *EditIn* and *EditOut* operations:

```
http://lsl-pm:17000/record/editin (or editout)
```

resulted in this response:

```
<Response>  
  <UUID>27638f24-2bb1-4ce6-8816-5a05a5e05897</UUID>  
  <State>Opened</State>  
  <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>  
  <EngineState>Running</EngineState>  
  <EngineTime>11:06:54;27</EngineTime>  
  <XMLRevision>2</XMLRevision>  
</Response>
```

Message

The *Message* operation enables the application to control a VTR by sending Sony VTR operations to the VTR and retrieving the response.

Note: The Sony 9-pin operations are specified in the Sony Video Cassette Recorder/ Player Protocol of Remote (9-pin) Connector, 2nd Edition.

The SDI Source Input associated with the Capture workflow receiving this operation must be connected to a VTR device via EIA RS-422A.

You can not use this operation unless the target workflow is in an active state with a Monitor Status of VTR connected. (View the Monitor Status tab in Workflow Designer).

This has the following format:

```
address:port/record/Message?request=[Sony operation]
```

Parameters

Parameter	Description
request (required)	<p>String, reserved numeric strings identified by Sony for 9-pin device operations. For example: <code>request=2000</code></p> <p>Typical Sony operations (supported by most VTRs) include:</p> <ul style="list-style-type: none"> • Stop = 2000 • Play = 2001 • Fast Forward = 2010 • Step Forward = 2014 • Rewind = 2020 • Step Backward = 2024 • Cue with DATA (Go To) = 2431[TC DATA] <ul style="list-style-type: none"> - TC = 09:59:59:00; TC DATA = 00595909 <p>Reference your VTR guide for supported operations.</p>

Example

In this example, the lsl-pm server is contacted on port 17000, issuing a Sony VTR Stop operation:

```
http://lsl-pm:17000/record/Message?request=2000
```

Typical Message Operation Response

Responses from *Message* operations do not follow the pattern of the other responses. Instead, they are specified by a Microsoft schema. The return is a <string element, whose data value is a string. The last two characters are a checksum and is ignored. The remaining string is the return value.

Issuing this *Message* operation with a 2001 request (Play):

```
http://LS-SVR:7500/record/message?request=2001
```

resulted in the following response:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">100111</string>
```

The response string 1001 (dropping the last 2 characters—the checksum) = ACK.

In this example, the *Message* operation is issued for a Status Sense:

```
http://lsl-pm:17000/record/message?request=612002
```

The response is:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">7220000193</string>
```

The response string 7220000193 =

7220 (7X20) = Status Data returning 2 data bytes (Data No. 0 through Data No. 1)

Data No. 0 = 20 = 00100000 (Bit 5 = 1 which means Tape or Cassette IN; more properly NOT OUT)

Data No. 1 = 01 = 00000001 (Bit 0 = 1 which indicates a status of PLAY).

Stop

Stops recording and/or removes a clip from the active list.

This has the following format:

```
http://<host>:18000/record/stop
```

Parameters

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) but they are treated identically (assuming you are using drop frame if the source is drop frame and non-drop frame if the source is non-drop frame).

Parameter	Description
uuid (required)	<p>GUID; the unique identifier of the clip to stop.</p> <p>For example: <code>uuid=21EC2020-3AEA-4069-A2DD-08002B30309D</code></p> <p>When this operation is issued the response contains all information for the associated clip. If a clip isn't specified in the request, the response contains the UUID of each clip in the list.</p> <p>Error: Returns <i>404 Not Found</i> if the clip is not in the list.</p>
end (optional)	<p>Timecode; specifies the clip's <i>exclusive</i> end timecode.</p> <p>For example: <code>end=01:42:35:00</code>.</p> <p>The end timecode represents the timecode of the frame after the last frame of video. If an end time is not specified the clip will stop recording immediately. This parameter has no affect if the duration parameter was used within the job's start operation.</p> <p>Error: Returns <i>400 Bad Request</i> if the end time results in a clip with a duration longer than initially specified.</p> <hr/> <p>Note: Only change the end timecode on files captured without an initial duration or when capturing QuickTime Closed or MXF OP1a Closed container formats.</p>

Example

```
http://LS-SVR:7500/record/stop
?uuid=724c593b-8da7-4c3b-b667-78d75e16abe1
```

Typical Stop Operation Response

Issuing this *Stop* operation resulted in the following response:

```
<Response>
  <UUID>724c593b-8da7-4c3b-b667-78d75e16abe1</UUID>
  <PercentCompleted>0</PercentCompleted>
  <Progress>111.244458333333</Progress>
```

```
<ActionDuration>660.0594</ActionDuration>  
<FPS>29.97002997003</FPS>  
<Start>18:26:12;11@29.97</Start>  
<End>18:37:12;13@29.97</End>  
<MarkIn>18:26:12;11@29.97</MarkIn>  
<MarkOut/>  
<Excluding>False</Excluding>  
<Name>Bob-New Workflow_LS-SVR - SDI Input 1.1</Name>  
<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>18:28:03;29</EngineTime>  
<XMLRevision>2</XMLRevision>  
</Response>
```

Status

The *Status* operation requests status information for a clip. When you do not specify a GUID, the Web Service returns the list of recordings currently in the list. You can process the list to extract a GUID and obtain status on any recording you want.

You can also obtain the GUID of a recording from the start operation.

This has the following formats:

```
http://<host>:18000/record/status
```

or

```
http://<host>:18000/record/status
?uuid=21EC2020-3AEA-4069-A2DD-08002B30309D
```

Parameters

Parameter	Description
uuid (optional)	<p>GUID; the unique identifier of the clip to obtain status information from.</p> <p>For example: <code>uuid=21EC2020-3AEA-4069-A2DD-08002B30309D</code></p> <p>When this operation is issued with a GUID, the response will contain all status details for the associated clip.</p> <p>If a clip GUID is not specified, ALL GUIDs of all clips in the clip list are returned and the response will include general status information.</p> <p>Error: Returns <i>404 Not Found</i> if the clip is not in the list.</p>

Example

```
http://LS-SVR:7500/record/status
?uuid=724c593b-8da7-4c3b-b667-78d75e16abe1
```

Typical Status Operation Response

Issuing this Status operation resulted in the following response:

```
<Response>
  <UUID>724c593b-8da7-4c3b-b667-78d75e16abe1</UUID>
  <PercentCompleted>0</PercentCompleted>
  <Progress>62.22883333333333</Progress>
  <ActionDuration>660.0594</ActionDuration>
  <FPS>29.97002997003</FPS>
  <Start>18:26:12;11@29.97</Start>
  <End>18:37:12;13@29.97</End>
  <MarkIn>18:26:12;11@29.97</MarkIn>
  <MarkOut/>
  <Excluding>False</Excluding>
  <Name>Bob-New Workflow_LS-SVR - SDI Input 1.1</Name>
  <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>18:27:14;24</EngineTime>  
<XMLRevision>2</XMLRevision>  
</Response>
```


Stream Operations

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

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

- [Introduction](#)
- [Obtaining Help for Streaming Operations](#)
- [System Operations](#)
- [Sources Operations](#)
- [Programs Operations](#)
- [Encoders Operations](#)
- [Packages Operations](#)
- [Channels Operations](#)

Introduction

Please review the following topics for general information about the Streaming API and how operations are presented. This reference is intended for readers who understand how to use Live Stream; for information about Live Stream, please read the Live Stream User Guide.

Limits of the API

You can not create most Live Stream components or configure them using the API. Before you can control and monitor streams on the Lightstream Live Server, you must first create and configure them using the Lightstream Live Web application.

Using Live Stream Groups via the API

Live Stream groups are multiple 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.

Using Shared vs. Dedicated Components

Two types of Stream components—media sources and output channels—are associated with a specific server and must be addressed directly on that server. For example, if you are controlling a specific media source, you must execute the operation using the DNS|IP address of the server where the source was created.

The 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 be executed properly, regardless of where the component was created and configured.

Brevity in Examples

The major enclosing element returned by each Stream operation has xmlns attributes.

For example:

```
<ArrayOfBrief xmlns="http://schemas.datacontract.org/2004/07/Telestream.Soa.Live.Vocabulary" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
```

For brevity, these attributes have been removed from all of the XML examples.

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.

Obtaining Help for Streaming Operations

You can obtain information about services in a variety of ways.

Displaying a List of Live Stream Services

To obtain a list of all Live Stream services, execute the *GetServiceEndpoints* operation at the root level: `http://<host>:18000/GetServiceEndpoints`. *GetServiceEndpoints* returns an ArrayOfBrief XML with URIs that will display all of the Lightspeed Live Stream service operations. For details, see [GetServiceEndpoints](#).

Displaying the Operations of a Specific Live Stream Service

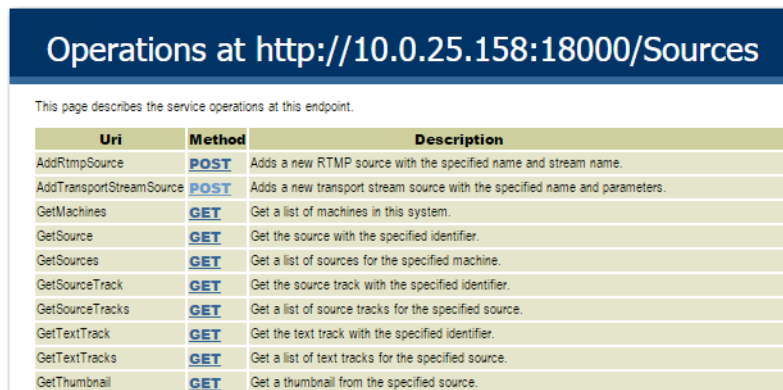
To list all of the operations in a given service, execute Help in the service you're interested in. This form of the Help has the following format:

`http://<host>:18000/<service category keyword>/help`.

Keywords: Sources | Programs | Encoders | Packages | Channels.

For example: `http://10.0.25.158:18000/Sources/help`

Help returns a web page listing all operations for the end point category you specified:



Uri	Method	Description
AddRtmpSource	POST	Adds a new RTMP source with the specified name and stream name.
AddTransportStreamSource	POST	Adds a new transport stream source with the specified name and parameters.
GetMachines	GET	Get a list of machines in this system.
GetSource	GET	Get the source with the specified identifier.
GetSources	GET	Get a list of sources for the specified machine.
GetSourceTrack	GET	Get the source track with the specified identifier.
GetSourceTracks	GET	Get a list of source tracks for the specified source.
GetTextTrack	GET	Get the text track with the specified identifier.
GetTextTracks	GET	Get a list of text tracks for the specified source.
GetThumbnail	GET	Get a thumbnail from the specified source.

Displaying Operation Details

To display details about a specific operation, execute the operation in question in the `<service>/help/operations` directory. This form of the Help has the following format:

`http://<host>:18000/<service category keyword>/help/operations/<Operation Keyword>`.

For example:

`http://10.0.25.158:18000/Sources/help/operations/GetMachines`

Help returns a web page illustrating the operation and its method:

Reference for <http://10.0.25.158:18000/Sources/GetMachines>

Get a list of machines in this system.

Url: <http://10.0.25.158:18000/Sources/GetMachines>

HTTP Method: GET

Message direction	Format	Body
Request	N/A	The Request body is empty.
Response	Unknown	The Response body cannot be inferred.

System Operations

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

- [GetMachines](#)
- [GetServiceEndpoints](#)
- [GetSystemSettings](#)
- [GetSystemLogs.zip](#)

Note: All System operations are located at the root `http://<host>:18000/`.
To display help for system operations, enter `http://<host>:18000/help`

GetMachines

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

The purpose of *GetMachines* is to identify the Live Stream server (or in the case of a group, all of the Live Stream servers in the group) by GUID. This is a pre-requisite for executing many of the Sources operations, plus [GetMachineStatistics](#).

To execute this operation, you must know the Windows domain name or IP address of the target Live Stream server or any one of the Live Stream servers in the group. It takes no parameters.

GetMachines has the following format:

```
http://<host>:18000/Sources/GetMachines
```

Operation Chain

No other operations are required to execute this operation.

Results

Upon success, *GetMachines* returns an ArrayOfBrief XML, including a Brief element for the Live Stream server (or, in the case of a group, each Live Stream server), with relevant details.

Example

This *GetMachines* returns a list of Live Stream servers from the server with IP 10.0.2.158 (you could also use the domain name of course):

```
http://10.0.2.158:18000/Sources/GetMachines
```

Typical Response

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

```
<ArrayOfBrief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>f104475a-ebe3-4d51-a414-718c41bf95c0</Identifier>
    <Name>LS-SVR-HD1</Name>
  </Brief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>p1534451-xof9-0t45-j687-803c34ug64q8</Identifier>
    <Name>LS-SVR-HD2</Name>
  </Brief>
```

```
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>s307798b-sdf5-5b56-k868-523d56vf77r3</Identifier>  
  <Name>LS-SVR-SD</Name>  
</Brief>  
</ArrayOfBrief>
```


GetServiceEndpoints

The purpose of this GET operation is to obtain a list of all Live Stream services, which is executed at the root level. There are no parameters.

GetServiceEndpoints has the following format:

```
http://<host>:18000/GetServiceEndpoints
```

Operation Chain

No other operations are required to execute this operation.

Results

Upon success, *GetServiceEndpoints* returns an ArrayOfBrief XML with URIs that display all of the Lightspeed Live Stream service operations.

You might not get a response if you target the wrong server or use the wrong port (18000), or a timeout occurs.

Example

This *GetServiceEndpoints* obtains a list of endpoints for the Live Stream system at 10.0.2.158:

```
http://10.0.2.158:18000/GetServiceEndpoints
```

Typical Response

In this response, Live Stream system responded with a list of endpoints.

```
<ArrayOfBrief>
  <Brief>
    <Description>Service for source-related operations</
Description>
    <Identifier>00000000-0000-0000-0000-000000000000</Identifier>
    <Name>http://LS-SVR:18000/Sources/Help</Name>
  </Brief>
  <Brief>
    <Description>Service for program-related operations</
Description>
    <Identifier>00000000-0000-0000-0000-000000000000</Identifier>
    <Name>http://LS-SVR:18000/Programs/Help</Name>
  </Brief>
  <Brief>
    <Description>Service for encoder-related operations</
Description>
    <Identifier>00000000-0000-0000-0000-000000000000</Identifier>
    <Name>http://LS-SVR:18000/Encoders/Help</Name>
  </Brief>
  <Brief>
    <Description>Service for package-related operations</
Description>
    <Identifier>00000000-0000-0000-0000-000000000000</Identifier>
```

```
<Name>http://LS-SVR:18000/Packages/Help</Name>  
</Brief>  
<Brief>  
  <Description>Service for channel-related operations</  
Description>  
  <Identifier>00000000-0000-0000-0000-000000000000</Identifier>  
  <Name>http://LS-SVR:18000/Channels/Help</Name>  
</Brief>  
</ArrayOfBrief>
```

GetSystemSettings

The purpose of this GET operation is to export the entire Live Stream system configuration in XML format. This XML can be used for a variety of purposes, including importing directly into another system, using the Import button in the Live Stream web app. There are no parameters.

GetSystemSettings has the following format:

```
http://<host>:18000/GetServerSettings
```

Results

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

You might not get a response if you target the wrong server or use the wrong port (18000), or a timeout occurs.

Example

This *GetSystemSettings* returns the system configuration from the system, using the server with IP 10.0.2.158:

```
http://10.0.2.158:18000/GetSystemSettings
```

Typical Response

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

```
<Domain>
  <Configuration>...</Configuration>
  <User>...</User>
  <Machine dlp1:identifier="f1c541f6-d003-41a0-b627-e6fa0add9c95"
  dlp1:name="LS-SVR" dlp1:leftaligncheckboxes="false">
    <dlp1:Parameter>...</dlp1:Parameter>
    ...
    <Source dlp1:identifier="f7338552-a221-4853-a8cb-60de6371ae1b"
  dlp1:name="LS-SVR - SDI Input 4" dlp1:description=""
  dlp1:leftaligncheckboxes="false">
    <dlp1: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"
  colspan="1">...</dlp1:Parameter>
    ...
    <SourceType>sdi</SourceType>
    <Statistics>...</Statistics>
    <Tracks>...</Tracks>
    <TextTracks/>
  </Source>
</Machine>
</Domain>
```

GetSystemLogs.zip

The purpose of this GET operation is to obtain the system logs in a zip file. The zip extension to the command identifies the file type returned. It also provides the downloaded file with the correct extension for saving in your default downloads directory (typically, C:\Users\\Downloads) when using a browser. This operation has no parameters.

Note: If you execute *GetSystemLogs.zip* during the time when a channel is streaming, an empty zip file is returned—the files are open for write by a service.

GetSystemLogs.zip has the following format:

```
http://<host>:18000/Sources/GetSystemLogs.zip
```

Results

Upon success, *GetSystemLogs.zip* returns a zip file of SNMP logs. The downloaded zip file contains two identical inner zip files. (This arrangement allows the potential for adding other logs or reports in the future.)

You might not get a response if you target the wrong server or use the wrong port (18000), or a timeout occurs.

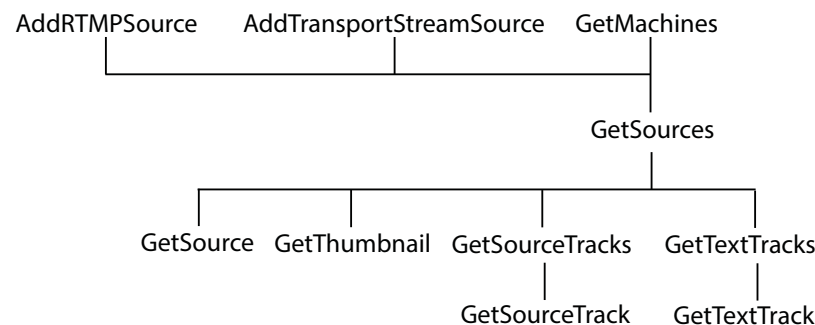
Sources Operations

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](#)
- [GetSourceTracks](#)
- [GetSourceTrack](#)
- [GetTextTracks](#)
- [GetTextTrack](#)
- [GetSourceThumbnail](#)

Note: All Sources operations start with `http://<host>:18000/Sources/`.

In this diagram, the operations are organized hierarchically, by machine GUID requirement. The operations to add sources require no GUIDs. All source operations require a machine GUID.



Note: To display help for Sources operations, enter `http://<host>:18000/Sources/help`

AddRtmpSource

The purpose of this POST operation is to add a new RTMP source to the target Live Stream server, identified by its DNS|IP address.

Note: Sources are defined specifically for a hardware port on the server where they are created. Thus the host (DNS|IP address) that you use must be that of the target server.

AddRtmpSource has the following format:

```
http://<host>:18000/Sources/AddRtmpSource?  
sourceName={SOURCE NAME}&streamName={STREAM NAME}
```

Operation Chain

No other operations are required to execute this operation.

Required 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: <code>sourceName=Boats Of Port Townsend</code>
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: <code>streamName=BoatsOfPortTownsendStream</code>

Results

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

If you are missing a required parameter, the operation returns an error XML. For example: `<Error>RTMP Stream name required.</Error>`.

You might not get a response if you target the wrong server or use the wrong port (18000), or a timeout occurs.

Example

This *AddRtmpSource* adds a new RTMP source to the target server and returns a Brief XML indicating that the source was added:

```
http://10.0.2.158:18000/Sources/AddRtmpSource?  
sourceName=Boats Of Port Townsend  
&streamName=BoatsOfPortTownsendStream
```

Typical Response

In this response, a new RTMP source named *Boats Of Port Townsend* 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.

```
<Brief>  
  <Description/>  
  <Identifier>5b25f78b-d3f3-4ab1-81e0-0c233e704626</Identifier>  
  <Name>Boats Of Port Townsend</Name>  
</Brief>
```

AddTransportStreamSource

The purpose of this POST operation is to add a new Transport Stream source to the target Live Stream server identified by its DNS|IP address, with the specified name and settings.

Note: Sources are defined specifically for a hardware port on the server where they are created. Thus the host (DNS|IP address) that you use must be that of the target server.

AddTransportStreamSource has the following format:

```
http://<host>:18000/Sources/AddTransportStreamSource  
?sourceName={SOURCE_NAME}&multicastIP={MULTICASTIP}  
&localIP={LOCALIP}&portNumber={PORTNUMBER}  
&programNumber={PROGRAMNUMBER}&sourceFilterIP={SOURCEFILTERIP}
```

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

Operation Chain

No other operations are required to 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; specifies the optional multicast group IP address being used by the Transport Stream generator program to broadcast.
localIP	IP address; specifies the IP address of the NIC card in the server where you are listening for the Transport Stream.
portNumber	Port number; specifies the port number that the Transport Stream generator program is broadcasting this Transport Stream on.
programNumber	Integer; specifies the program number in the Transport Stream that you are receiving.
sourceFilterIP (optional)	IP address; 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 Brief XML, indicating the source that was added.

If you are missing a required parameter, the operation returns an error XML. For example: `<Error>Transport Stream name required.</Error>`.

You might not get a response if you target the wrong server or use the wrong port (18000), or a timeout occurs.

Example

This `AddTransportStreamSource` adds a new Transport Stream source to the target server and returns a Brief XML indicating the source was added:

```
http://10.0.25.162:18000/Sources/AddTransportStreamSource
?sourceName=Hiking Wind Ridge
&multicastIP=239.1.2.3&localIP=10.0.25.162
&portNumber=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.

```
<Brief>
  <Description/>
  <Identifier>901ffd81-1217-4855-8c0d-e703e51d2201</Identifier>
  <Name>Hiking Wind Ridge</Name>
</Brief>
```

GetSources

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

GetSources has the following format:

```
http://<host>:18000/Sources/GetSources?machine={MACHINE GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetMachines](#) (machine GUID)

Parameters

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

Results

On success, *GetSources* returns an `ArrayOfBrief` element, with a `Brief` element for each source in the target Live Stream server with details.

Example

This *GetSources* returns a list of Sources for the target server with their details:

```
http://10.0.2.158:18000/Sources/GetSources?machine=1129625b-0d7d-490a-aa3f-315214a0b6f2
```

Typical Response

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

```
<ArrayOfBrief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>61fdec22-9257-49d1-9b3b-a889476f2a18</Identifier>  
    <Name>LS-SVR - SDI Input 3</Name>  
  </Brief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>838a4e67-616b-43cd-93d6-77175b626e0b</Identifier>  
    <Name>LS-SVR - SDI Input 4</Name>  
  </Brief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>d6000fc9-7aab-4d18-8a43-35e47e7d68e2</Identifier>
```

```
<Name>LS-SVR - SDI Input 2</Name>
</Brief>
<Brief>
  <Description i:nil="true"/>
  <Identifier>ecb3e35d-a959-4c23-89e4-8ab64de12571</Identifier>
  <Name>Slate-test</Name>
</Brief>
<Brief>
  <Description/>
  <Identifier>1689adb3-0b2d-4aa1-83da-f77cabb2ba2f</Identifier>
  <Name>LS-SVR - SDI Input 1</Name>
</Brief>
</ArrayOfBrief>
```

GetSource

The purpose of this GET operation is to obtain the details of a specific source on the target Live Stream server.

GetSource has the following format:

```
http://<host>:18000/Sources/GetSource?identifier={SOURCE GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetMachines](#) (machine GUID) > [GetSources](#) (source GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies this source.

Results

On success, *GetSource* returns a SourceBrief XML, with details about the target source. A SourceBrief has these elements:

- 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 XML is returned.

Example

This *GetSource* operation returns a SourceBrief XML for the target Identifier on the target machine, with all of its details:

```
http://10.0.2.158:18000/Sources/getsource?identifier=838a4e67-616b-43cd-93d6-77175b626e0b
```

Typical Response

This is a typical response from *GetSource*; a list of metadata defining the target source (identified by each ParameterBrief element with name/value pairs), plus tracks in the source.

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

```
<SourceBrief>
  <Description i:nil="true"/>
  <Identifier>40c55550-fd86-400e-acad-18e1c9224da6</Identifier>
  <Name>VTR-Sim1</Name>
  <Details>
    <ParameterBrief>
      <Name>Process</Name>
      <Value>6608</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Resolution</Name>
      <Value>1920x1080p</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Frame Rate</Name>
      <Value>29.97</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Bit Depth</Name>
      <Value>10-Bit</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Audio</Name>
      <Value>2</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Captions</Name>
      <Value>No</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Time Code (Rs422)</Name>
      <Value>00:04:45:00</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Deck State</Name>
      <Value>Unknown</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Deck Mode</Name>
      <Value>Remote</Value>
    </ParameterBrief>
  </Details>
  <TextTracks>
    <Briefs/>
  </TextTracks>
  <Tracks>
    <Briefs>
      <Brief>
        <Description i:nil="true"/>
        <Identifier>afad3c11-3324-4144-8122-526519759289</Identifier>
        <Name>Mono 1</Name>
      </Brief>
    </Briefs>
  </Tracks>
</SourceBrief>
```

```
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>6e35826c-8c83-4470-af3f-5416d211b36b</Identifier>  
  <Name>Mono 2</Name>  
</Brief>  
</Briefs>  
</Tracks>  
<Type>FileLoop</Type>  
</SourceBrief>
```

GetSourceTracks

The purpose of this GET operation is to obtain a list of audio tracks for the target source and return them in a list for further use.

GetSourceTracks has the following format:

```
http://<host>:18000/Sources/GetSourceTracks?source={SOURCE GUID}
```

Operation Chain

The following operations must be executed in order 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.

Results

On success, *GetSourceTracks* returns an `ArrayOfBrief` XML, with a `Brief` for each audio track associated with the source.

If no source tracks are present, the `ArrayOfBrief` is returned empty.

Example

This *GetSourceTracks* returned a list of tracks in an `ArrayOfBrief` XML associated with the target source:

```
http://10.0.2.158:18000/Sources/GetSourceTracks
```

Typical Response

In this response, the target source has two tracks, as defined in their `Brief` elements: Stereo 1 and Stereo 1 (1). Each has a GUID in the `Identifier`, and a `Name`.

```
<ArrayOfBrief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>1f3a2a14-4ea6-411d-83a7-59694f4eed7e</Identifier>
    <Name>Stereo 1</Name>
  </Brief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>863648a9-e422-4fcc-881c-7545c18c70c1</Identifier>
    <Name>Stereo 1(1)</Name>
  </Brief>
</ArrayOfBrief>
```

GetSourceTrack

The purpose of this GET operation is to obtain details about a specific audio track.

GetSourceTrack has the following format:

```
http://<host>:18000/Sources/GetSourceTrack  
?identifier={TRACK GUID}
```

In addition to Identifier and Name, a SourceTrackBrief has these relevant elements:

- ChannelConfiguration—the string identifying the track
- Details—a set of ParameterBrief elements, relevant to the type of audio track.

Operation Chain

The following operations must be executed in order to obtain the required GUID for this operation:

[GetMachines](#) (machine GUID) > [GetSources](#) (source GUID) > [GetSourceTracks](#) (track GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific track.

Results

On success, *GetSourceTrack* returns a SourceTrackBrief XML, with details about the target track.

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

Example

This *GetSourceTrack* returned a list of tracks in an SourceTrackBrief XML associated with the target track:

```
http://10.0.2.158:18000/Sources/GetSourceTrack  
?identifier=1f3a2a14-4ea6-411d-83a7-59694f4eed7e
```

Typical Response

In this response, the target track is Stereo 1, as defined in the Brief element: Stereo 1, with details. Each track has a GUID in the Identifier, and a name.

```
<SourceTrackBrief>  
  <Description i:nil="true"/>  
  <Identifier>1f3a2a14-4ea6-411d-83a7-59694f4eed7e</Identifier>  
  <Name>Stereo 1</Name>  
  <ChannelConfiguration>Stereo</ChannelConfiguration>  
  <Details>
```



```
<ParameterBrief>
  <Name>Language</Name>
  <Value>English</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Service</Name>
  <Value>Primary</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Default</Name>
  <Value>False</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Left Channel</Name>
  <Value>1</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Right Channel</Name>
  <Value>2</Value>
</ParameterBrief>
</Details>
</SourceTrackBrief>
```

GetTextTracks

The purpose of this GET operation is to obtain a list of 608 or 7088 text (caption) tracks that have been added to the target source and return them in a list for further use.

GetTextTracks has the following format:

```
http://<host>:18000/Sources/GetTextTracks?source={SOURCE GUID}
```

Operation Chain

The following operations must be executed in order 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.

Results

On success, *GetTextTracks* returns an ArrayOfBrief XML, with a Brief for each text track associated with the source.

If no text tracks are present, the ArrayOfBrief is returned empty.

Example

This *GetTextTracks* returned a list of tracks associated with the target source:

```
http://10.0.2.158:18000/Sources/GetTextTracks
```

Typical Response

In this response, the target source has two text tracks, as defined in the Brief element. Each has a GUID in the Identifier, and a Name.

```
<ArrayOfBrief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>ef1752a6-f4fa-49e3-a779-54373f31cb60</Identifier>  
    <Name>C608English</Name>  
  </Brief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>1b49058a-3d5a-4186-9e55-c52ee45f0b66</Identifier>  
    <Name>C708English</Name>  
  </Brief>  
</ArrayOfBrief>
```

GetTextTrack

The purpose of this GET operation is to obtain details about a specific text (608 or 708 caption) track.

GetTextTrack has the following format:

```
http://<host>:18000/Sources/GetTextTrack?identifier={TRACK GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetMachines](#) (machine GUID) > [GetSources](#) (source GUID) > [GetTextTracks](#) (track GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific track.

Results

On success, *GetTextTrack* returns a *TextTrackBrief* XML, with details about the target track: Identifier and Name, plus Details—a set of *ParameterBrief* elements.

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

Example

This *GetTextTrack* returned details about the text track that you've targeted:

```
http://10.0.2.158:18000/Sources/GetTextTrack  
?identifier=1f3a2a14-4ea6-411d-83a7-59694f4eed7e
```

Typical Response

In this response, the target track is C608.

```
<TextTrackBrief>  
  <Description i:nil="true"/>  
  <Identifier>7036a78a-5755-4422-b445-27c09fe6cc7e</Identifier>  
  <Name>C608</Name>  
  <Details>  
    <ParameterBrief>  
      <Name>Language</Name>  
      <Value>English</Value>  
    </ParameterBrief>  
  </Details>  
  <TextConfiguration>C608</TextConfiguration>  
</TextTrackBrief>
```

GetSourceThumbnail

The purpose of this GET operation is to obtain a thumbnail from the target source. The operation returns a JPEG, which you can render or save as a file.

GetSourceThumbnail has the following format:

```
http://<host>:18000/Sources/GetSourceThumbnail  
?source={SOURCE GUID}
```

Operation Chain

The following operations must be executed in order 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.

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

This *GetSourceThumbnail* returned a JPEG associated with the target source:

```
http://10.0.2.158:18000/Sources/GetSourceThumbnail  
?source=27602854-35a6-4f0c-827c-ebd7ac5d40a9
```

Programs Operations

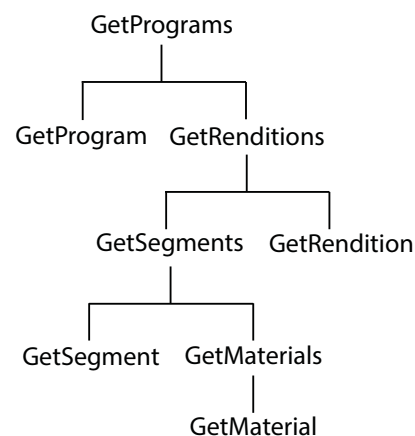
A program defines what media should be encoded. Programs are comprised of renditions; renditions are comprised of segments, and segments are comprised of materials. These operations enable you to identify all of the programs in a Live Stream server and target their components: renditions, segments, and materials, successively.

These topics are organized in the order they are often used:

- [GetPrograms](#)
- [GetProgram](#)
- [GetRenditions](#)
- [GetRendition](#)
- [GetSegments](#)
- [GetSegment](#)
- [GetMaterials](#)
- [GetMaterial](#)

Note: All Programs operations start with `http://<host>:18000/Programs/`.

In this diagram, the operations are organized hierarchically, by program GUID requirement.



Note: To display help for Programs operations, enter `http://<host>:18000/Programs/help`

GetPrograms

The purpose of this GET operation is to obtain a list of all of the programs that have been added to the target Live Stream system. There are no parameters.

GetPrograms has the following format:

```
http://<host>:18000/Programs/GetPrograms
```

Operation Chain

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

Results

On success, *GetPrograms* returns an ArrayOfBrief XML, with a Brief element for each program in the system.

If no Programs are present, the ArrayOfBrief is returned empty.

Example

This *GetPrograms* returns a list of programs in an ArrayOfBrief XML associated with the target server/server group:

```
http://10.0.2.158:18000/Programs/GetPrograms
```

Typical Response

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

```
<ArrayOfBrief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>9cc7582b-7b00-4381-8cf5-62ba444fc51e</Identifier>
    <Name>Perform1</Name>
  </Brief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>7fbb4998-f82a-44da-8d6c-47344b47c10b</Identifier>
    <Name>The Wooden Boats of Port Townsend</Name>
  </Brief>
</ArrayOfBrief>
```

GetProgram

The purpose of this GET operation is to obtain detailed information about a specific program.

GetProgram has the following format:

```
http://<host>:18000/Programs/GetProgram  
?identifier={Program GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetPrograms](#) (program GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific program.

Results

On success, *GetProgram* returns a ProgramBrief XML, with a list of renditions in the target program.

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

Example

This *GetProgram* returned a ProgramBrief XML associated with the target program:

```
http://10.0.2.158:18000/Programs/GetProgram  
?identifier=7fbb4998-f82a-44da-8d6c-47344b47c10b
```

Typical Response

In this response, the target program—The Wooden Boats of Port Townsend—has three Renditions—English Stereo, German Stereo, and English Audio. Each rendition (the Brief elements) has an Identifier with a GUID you can use to query it.

```
<ProgramBrief>  
  <Description i:nil="true"/>  
  <Identifier>7fbb4998-f82a-44da-8d6c-47344b47c10b</Identifier>  
  <Name>The Wooden Boats of Port Townsend</Name>  
  <Renditions>  
    <Briefs>  
      <Brief>  
        <Description i:nil="true"/>  
        <Identifier>1e32e2f1-3702-4d83-8460-e9d2231db3c5</Identifier>  
        <Name>English Stereo</Name>  
      </Brief>
```

```
<Brief>
  <Description i:nil="true"/>
  <Identifier>2ebd98a3-a61b-4846-a141-a6429a890c13</Identifier>
  <Name>German Stereo</Name>
</Brief>
<Brief>
  <Description i:nil="true"/>
  <Identifier>f3b2cd6c-d5a2-48af-9495-c343119c6112</Identifier>
  <Name>English Audio</Name>
</Brief>
</Briefs>
</Renditions>
<Resolution>1920 x 1080</Resolution>
</ProgramBrief>
```


GetRenditions

The purpose of this GET operation is to obtain a list of Renditions that comprise a specific program.

GetRenditions has the following format:

```
http://<host>:18000/Programs/GetRenditions  
?program={PROGRAM GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetPrograms](#) (program GUID)

Parameters

Parameter	Description
program	GUID; string that identifies a specific program.

Results

On success, *GetRenditions* returns an `ArrayOfBrief` XML, with one `Brief` for each rendition of the program.

If no Renditions are present, the `ArrayOfBrief` is returned empty.

Example

This *GetRenditions* returned a list of renditions for the target program:

```
http://10.0.2.158:18000/programs/GetRenditions  
?program=7fbb4998-f82a-44da-8d6c-47344b47c10b
```

Typical Response

In this response, there are two renditions—one is English, the other German. Each `Brief` identifies the rendition by `Identifier` and `Name`, which is presented in the Web app.

```
<ArrayOfBrief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>c678fba3-1360-456f-b621-98e9b6663578</Identifier>  
    <Name>POC1EnglishRendition</Name>  
  </Brief>  
  <Brief>  
    <Description i:nil="true"/>  
    <Identifier>2f153712-4d65-4116-9a2d-55a1ba8c41a1</Identifier>  
    <Name>POC1German</Name>  
  </Brief>  
</ArrayOfBrief>
```

GetRendition

The purpose of this GET operation is to obtain details about a specific rendition.

GetRendition has the following format:

```
http://<host>:18000/Programs/GetRendition
?identifier={RENDITION GUID}
```

Operation Chain

The following operations must be executed in sequence to obtain the required GUID for each succeeding operation:

[GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID)

Parameters

Parameter	Description
source	GUID; string that identifies a specific rendition.

Results

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

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

Example

This *GetRendition* returned a list of Segments in the RenditionBrief XML associated with the target rendition:

```
http://10.0.2.158:18000/Programs/GetRendition
?identifier=1e32e2f1-3702-4d83-8460-e9d2231db3c5
```

Typical Response

In this response, the target 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.

```
<RenditionBrief>
  <Description i:nil="true"/>
  <Identifier>1e32e2f1-3702-4d83-8460-e9d2231db3c5</Identifier>
  <Name>English Stereo</Name>
  <Details>
    <ParameterBrief>
      <Name>Channel Configuration</Name>
      <Value>Stereo</Value>
    </ParameterBrief>
    <ParameterBrief>
      <Name>Audio Service</Name>
```

```
    <Value>Primary</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Language</Name>
    <Value>English</Value>
  </ParameterBrief>
</Details>
<Segments>
  <Briefs>
    <Brief>
      <Description i:nil="true"/>
      <Identifier>27fbd99f-935e-47af-b1e4-d83131f4e023</Identifier>
      <Name>Slate</Name>
    </Brief>
    <Brief>
      <Description i:nil="true"/>
      <Identifier>6c880639-ac59-447e-a26b-248c6a244e3e</Identifier>
      <Name>Segment1</Name>
    </Brief>
    <Brief>
      <Description i:nil="true"/>
      <Identifier>714be355-bbf3-4d96-a294-0b7cbfcd8ae2</Identifier>
      <Name>Ad1</Name>
    </Brief>
  </Briefs>
</Segments>
<Type>VideoAndAudio</Type>
</RenditionBrief>
```

GetSegments

The purpose of this GET operation is to obtain a list of Segments that comprise a specific rendition of a program.

GetSegments has the following format:

```
http://<host>:18000/Programs/GetSegments  
?rendition={RENDITION GUID}
```

Operation Chain

The following operations must be executed in sequence to obtain the required GUID for each succeeding operation:

[GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID)

Parameters

Parameter	Description
rendition	GUID; string that identifies a specific rendition.

Results

On success, *GetSegments* returns an ArrayOfBrief XML, which has one Brief for each segment in the rendition.

If no Segments are present, the ArrayOfBrief is returned empty.

If the rendition you specify does not exist, the Live Stream server returns an Error:

```
<Error>Could not find Telestream.Soa.Live.Vocabulary.Rendition  
with identifier 1e32e2f1-3702-4d83-8460-e9d2231db3c5</Error>
```

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

Example

This *GetSegments* returned a list of Segments associated with the target rendition:

```
http://10.0.2.158:18000/Programs/GetSegments  
?rendition=1e32e2f1-3702-4d83-8460-e9d2231db3c5
```

Typical Response

In this response, the target rendition has 3 segments. Each is identified by Name and by an Identifier GUID.

```
<ArrayOfBrief>  
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>4be5f8d8-e7c6-44bf-a4be-9b37450a00e1</Identifier>  
  <Name>Segment1</Name>
```

```
</Brief>
<Brief>
  <Description i:nil="true"/>
  <Identifier>173e5f94-faf1-4ce6-9b73-2dd5c29cc704</Identifier>
  <Name>Segment2</Name>
</Brief>
<Brief>
  <Description i:nil="true"/>
  <Identifier>a5be008c-f8ec-49a1-94c5-2f29e2117183</Identifier>
  <Name>Segment3</Name>
</Brief>
</ArrayOfBrief>
```

GetSegment

The purpose of this GET operation is to obtain details about a specific segment.

GetSegment has the following format:

```
http://<host>:18000/Programs/GetSegment?identifier={SEGMENT GUID}
```

Operation Chain

The following operations must be executed in sequence to obtain the required GUID for each succeeding operation:

```
GetPrograms (program GUID) > GetRenditions (rendition GUID) >
GetSegments (segment GUID)
```

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific segment.

Results

On success, *GetSegment* returns a SegmentBrief XML, with details about the target segment and all of its Materials.

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

Example

This *GetSegment* returned a list of Materials associated with the target segment:

```
http://10.0.2.158:18000/Programs/GetSegment
?identifier=1e32e2f1-3702-4d83-8460-e9d2231db3c5
```

Typical Response

In this response, the target segment is Slate. It has a StartTrigger and an EndTrigger, and Materials consisting of one asset, identified by the Brief element named SlateClip.

```
<SegmentBrief>
  <Description i:nil="true"/>
  <Identifier>27fbd99f-935e-47af-b1e4-d83131f4e023</Identifier>
  <Name>Slate</Name>
  <EndTrigger>
    <Description>No condition</Description>
    <Identifier>a4094f4b-581a-4c3b-a7b3-2649a5008326</Identifier>
    <Name>None</Name>
    <Details/>
  </EndTrigger>
  <Materials>
    <Briefs>
      <Brief>
```

```
<Description>Asset with sound and visual.</Description>
<Identifier>2d3762e6-8da2-4788-8204-b249a9560755</Identifier>
<Name>SlateClip</Name>
</Brief>
</Briefs>
</Materials>
<Priority>1</Priority>
<StartTrigger>
  <Description>No condition</Description>
  <Identifier>a4094f4b-581a-4c3b-a7b3-2649a5008326</Identifier>
  <Name>None</Name>
  <Details/>
</StartTrigger>
</SegmentBrief>
```

GetMaterials

The purpose of this GET operation is to obtain a list of Materials that comprise a specific segment.

GetMaterials has the following format:

```
http://<host>:18000/Programs/GetMaterials?segment={SEGMENT GUID}
```

Operation Chain

The following operations must be executed in sequence to obtain the required GUID for each succeeding operation:

[GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID) >
[GetSegments](#) (segment GUID)

Parameters

Parameter	Description
segment	GUID; string that identifies a specific segment.

Results

On success, *GetMaterials* returns an ArrayOfBrief XML, which has one Brief for each material in the segment.

If the segment does not contain any material, the ArrayOfBrief is returned empty.

Example

This *GetMaterials* returned a list of Materials that comprise the target segment:

```
http://10.0.2.158:18000/Sources/GetMaterials  
?segment=1f3a2a14-4ea6-411d-83a7-59694f4eed7e
```

Typical Response

In this response, the target segment has two material items in a Brief, which has a GUID in the Identifier, and a Name.

```
<ArrayOfBrief>  
  <Brief>  
    <Description>A placeholder for a source.</Description>  
    <Identifier>53850181-bcf3-4c1f-8827-d2724d6c9a87</Identifier>  
    <Name>Source Placeholder</Name>  
  </Brief>  
  <Brief>  
    <Description>Asset with static visual component.</Description>  
    <Identifier>b4e31759-85a3-4ed5-a87f-fbf5bf0f0691</Identifier>  
    <Name>Image</Name>  
  </Brief>  
</ArrayOfBrief>
```


GetMaterial

The purpose of this GET operation is to obtain details about a specific item of material.

GetMaterial has the following format:

```
http://<host>:18000/Programs/GetMaterial?identifier={MATERIAL  
GUID}
```

Operation Chain

The following operations must be executed in sequence to obtain the required GUID for each succeeding operation:

[GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID) >
[GetSegments](#) (segment GUID) > [GetMaterials](#) (material GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific material.

Results

On success, *GetMaterial* returns a MaterialBrief XML, with details about the target material.

If no material exists with the specified GUID, an Error XML is returned.

Example

This *GetMaterial* returned the details about the target material:

```
http://10.0.2.158:18000/Programs/GetMaterial  
?identifier=2d3762e6-8da2-4788-8204-b249a9560755
```

Typical Response

In this response, the target material is identified by the Description, Identifier, and a Name. The ParameterBrief elements provide a set of parameters appropriate for this type of material.

```
<MaterialBrief>  
  <Description>Asset with sound and visual component.</Description>  
  <Identifier>2d3762e6-8da2-4788-8204-b249a9560755</Identifier>  
  <Name>SlateClip</Name>  
  <Details>  
    <ParameterBrief>  
      <Name>Left</Name>  
      <Value>0</Value>  
    </ParameterBrief>  
    <ParameterBrief>  
      <Name>Width</Name>
```

```
<Value>1920</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Top</Name>
  <Value>0</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Height</Name>
  <Value>1080</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Lock in Visual Editor</Name>
  <Value>False</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Resource</Name>
  <Value>D:\Material\Slate_1080i_BoatsPortTownsend.mov</Value>
</ParameterBrief>
</Details>
<ZIndex>0</ZIndex>
</MaterialBrief>
```

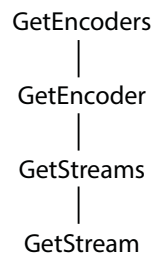
Encoders Operations

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>:18000/Encoders/`.

In this diagram, the operations are organized hierarchically, by GUID requirement. The GetEncoders operation does not require a GUID.



Note: To display help for Encoders operations, enter `http://<host>:18000/Encoders/help`

GetEncoders

The purpose of this GET operation is to obtain a list of encoders that have been created on the target Live Stream system. This operation has no parameters.

GetEncoders has the following format:

```
http://<host>:18000/Encoders/GetEncoders
```

Operation Chain

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

Results

On success, *GetEncoders* returns an ArrayOfBrief XML, which has one Brief for each encoder on the Live Stream server.

If no encoders are present, the ArrayOfBrief is returned empty.

Example

This *GetEncoders* returned a list of encoders associated with the Live Stream server whose IP address is 10.0.2.158:

```
http://10.0.2.158:18000/Encoders/GetEncoders
```

Typical Response

In this response, the target Live Stream server has 5 encoders; each is identified by Name and Description, and a GUID in the Identifier which you can use to target the encoder for further utilization.

```
<ArrayOfBrief>
  <Brief>
    <Description>HEVC</Description>
    <Identifier>67118539-5869-413d-86df-e42582dc428a</Identifier>
    <Name>HEVC5</Name>
  </Brief>
  <Brief>
    <Description>HEVC</Description>
    <Identifier>c49e8d6e-dc0d-4b7a-b756-71ccfa59861a</Identifier>
    <Name>HEVC3</Name>
  </Brief>
  <Brief>
    <Description>AVC</Description>
    <Identifier>9ca0cd8c-2323-4d4d-9cf0-5a71b86c0c33</Identifier>
    <Name>AVC3</Name>
  </Brief>
  <Brief>
    <Description>AAC</Description>
    <Identifier>1bad0882-7cac-4cdd-b2a7-28dd909de467</Identifier>
    <Name>AAC</Name>
  </Brief>
  <Brief>
```

```
<Description>AAC</Description>  
<Identifier>930ae733-6e1f-49a2-85ef-05a74d4c0d74</Identifier>  
<Name>AAC</Name>  
</Brief>  
</ArrayOfBrief>
```

GetEncoder

The purpose of this GET operation is to obtain details about a specific encoder.

GetEncoder has the following format:

```
http://<host>:18000/Encoders/GetEncoder?identifier={ENCODER_GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetEncoders](#) (encoder GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific encoder.

Results

On success, *GetEncoder* returns an EncoderBrief XML, with details about the target encoder.

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

Example

This *GetEncoder* returned the details about the target encoder:

```
http://10.0.2.158:18000/Encoders/GetEncoder  
?identifier=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

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

```
<EncoderBrief>  
  <Description>AAC</Description>  
  <Identifier>1bad0882-7cac-4cdd-b2a7-28dd909de467</Identifier>  
  <Name>AAC</Name>  
  <Codec i:nil="true"/>  
  <Details/>  
  <Streams>  
    <Briefs>  
      <Brief>  
        <Description>Mono, 44.1kHz</Description>  
        <Identifier>96777e1b-ed21-4387-8d55-f336e62df96a</Identifier>  
        <Name>Facebook_AAC_128kbp</Name>  
      </Brief>  
    </Briefs>  
  </Streams>  
</EncoderBrief>
```

```
<Brief>  
  <Description>Stereo, 48kHz</Description>  
  <Identifier>5dd39185-e5b8-4687-8629-6222cfce2eb8</Identifier>  
  <Name>AAC_128kbps</Name>  
</Brief>  
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>7cae9e93-732d-4117-8a27-37e2c9b298d6</Identifier>  
  <Name>AAC_256_44.1_HEv1_5.1</Name>  
</Brief>  
</Briefs>  
</Streams>  
</EncoderBrief>
```


GetStreams

The purpose of this GET operation is to obtain a list of streams that have been added to a specific encoder.

GetStreams has the following format:

```
http://<host>:18000/Encoders/GetStreams?encoder={ENCODER_GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetEncoders](#) (encoder GUID)

Parameters

Parameter	Description
encoder	GUID; string that identifies a specific encoder.

Results

On success, *GetStreams* returns an `ArrayOfBrief` XML, which has one `Brief` for each stream in the encoder.

If no streams are present, the `ArrayOfBrief` is returned empty.

Example

This *GetStreams* returned a list of streams associated with the target encoder:

```
http://10.0.2.158:18000/Encoders/GetStreams  
?encoder=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

In this response, the target encoder has 3 streams: one for Facebook, another for AAC at 128kbps, and another custom AAC stream. Each is identified by Name and by an Identifier GUID. You can use the GUID to target a specific stream for further use.

```
<ArrayOfBrief>  
  <Brief>  
    <Description>Mono, 44.1kHz</Description>  
    <Identifier>96777e1b-ed21-4387-8d55-f336e62df96a</Identifier>  
    <Name>Facebook_AAC_128kbp</Name>  
  </Brief>  
  <Brief>  
    <Description>Stereo, 48kHz</Description>  
    <Identifier>5dd39185-e5b8-4687-8629-6222cfce2eb8</Identifier>  
    <Name>AAC_128kbps</Name>  
  </Brief>  
</ArrayOfBrief>
```

```
<Description i:nil="true"/>  
<Identifier>7cae9e93-732d-4117-8a27-37e2c9b298d6</Identifier>  
<Name>AAC_256_44.1_HEv1_5.1</Name>  
</Brief>  
</ArrayOfBrief>
```

GetStream

The purpose of this GET operation is to obtain details about a specific stream.

GetStream has the following format:

```
http://<host>:18000/Encoders/GetStream?identifier={STREAM GUID}
```

Operation Chain

One of the following operations must be executed to obtain the required GUID before you can execute this operation.

[GetStreams](#) or [GetEncoder](#) (stream GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific stream.

Results

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

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

Example

This *GetStream* returned the details about the target stream:

```
http://10.0.2.158:18000/Encoders/GetStream  
?identifier=7cae9e93-732d-4117-8a27-37e2c9b298d6
```

Typical Response

In this response, the target stream is identified by the Description, Identifier, and a Name, plus its Essence. The ParameterBrief elements provide a set of parameters appropriate for this type of material.

```
<StreamBrief>  
  <Description i:nil="true"/>  
  <Identifier>7cae9e93-732d-4117-8a27-37e2c9b298d6</Identifier>  
  <Name>AAC_256_44.1_HEv1_5.1</Name>  
  <Details>  
    <ParameterBrief>  
      <Name>Bitrate</Name>  
      <Value>256 kbps</Value>  
    </ParameterBrief>  
    <ParameterBrief>  
      <Name>Sample rate</Name>  
      <Value>44.1 kHz</Value>  
    </ParameterBrief>  
    <ParameterBrief>
```

```
<Name>AAC profile</Name>  
<Value>HEv1</Value>  
</ParameterBrief>  
<ParameterBrief>  
  <Name>Channel layout</Name>  
  <Value>5.1 (C L R BL BR LFE)</Value>  
</ParameterBrief>  
</Details>  
<Essence>Audio</Essence>  
</StreamBrief>
```

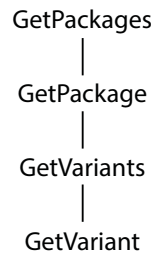
Packages Operations

The Packages operations enable you to choose the type of packages you want to create—Apple HLS, DASH, or RTMP— specify a program to use with the package, and configure package settings.

- [GetPackages](#)
- [GetPackage](#)
- [GetVariants](#)
- [GetVariant](#)

Note: All Packages operations start with `http://<host>:18000/Packages/`.

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



Note: To display help for Packages operations, enter `http://<host>:18000/Packages/help`

GetPackages

The purpose of this GET operation is to obtain a list of packages that have been created on the target Live Stream system. This operation has no parameters.

GetPackages has the following format:

```
http://<host>:18000/Packages/GetPackages
```

Operation Chain

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

Results

On success, *GetPackages* returns an ArrayOfBrief XML, which has one Brief for each package on the Live Stream server.

If no packages are present, the ArrayOfBrief is returned empty.

Example

This *GetPackages* returned a list of packages associated with the Live Stream server whose IP address is 10.0.2.158:

```
http://10.0.2.158:18000/Packages/GetPackages
```

Typical Response

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

```
<ArrayOfBrief>
  <Brief>
    <Description>Apple HLS</Description>
    <Identifier>e3b942bb-506f-44fe-8052-cbf3559f9513</Identifier>
    <Name>Apple HLS</Name>
  </Brief>
  <Brief>
    <Description>DASH</Description>
    <Identifier>da041b7f-59ef-4352-b26d-b3cb357dfc92</Identifier>
    <Name>DASH</Name>
  </Brief>
</ArrayOfBrief>
```

GetPackage

The purpose of this GET operation is to obtain details about a specific package.

GetPackage has the following format:

```
http://<host>:18000/Packages/GetPackage?identifier={PACKAGE GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetPackages](#) (package GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific package.

Results

On success, *GetPackage* returns a PackageBrief XML, with details about the target package.

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

Example

This *GetPackage* returns details about the target package:

```
http://10.0.2.158:18000/Packages/GetPackage  
?identifier=e3b942bb-506f-44fe-8052-cbf3559f9513
```

Typical Response

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

```
<PackageBrief>  
  <Description>Apple HLS</Description>  
  <Identifier>e3b942bb-506f-44fe-8052-cbf3559f9513</Identifier>  
  <Name>Apple HLS</Name>  
  <Details>  
    <ParameterBrief>  
      <Name>Segment Duration</Name>  
      <Value>10</Value>  
    </ParameterBrief>  
    <ParameterBrief>  
      <Name>Playlist Name</Name>  
      <Value>playlist</Value>  
  </Details>  
</PackageBrief>
```

```

</ParameterBrief>
<ParameterBrief>
  <Name>HLS Version</Name>
  <Value>4.0</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Enable Byte Range</Name>
  <Value>False</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Playlist Type</Name>
  <Value>Rolling</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Elements</Name>
  <Value>10</Value>
</ParameterBrief>
<ParameterBrief>
  <Name>Encryption</Name>
  <Value>None</Value>
</ParameterBrief>
</Details>
<Program>7fbb4998-f82a-44da-8d6c-47344b47c10b</Program>
<Type>Hls</Type>
<Variants>
  <Briefs>
    <Brief>
      <Description i:nil="true"/>
      <Identifier>5ae52d4c-eb4e-4751-a4e5-ae3a035f6</Identifier>
      <Name>AAC 128 Audio English</Name>
    </Brief>
    <Brief>
      <Description i:nil="true"/>
      <Identifier>c94a6e78-3b5a-44a7-9389-c687b2757</Identifier>
      <Name>AAC 256 Audio English</Name>
    </Brief>
  </Briefs>
</Variants>
</PackageBrief>

```


GetVariants

The purpose of this GET operation is to obtain a list of variants that comprise the target package.

GetVariants has the following format:

```
http://<host>:18000/Packages/GetVariants?package={PACKAGE GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetPackages](#) (package GUID)

Parameters

Parameter	Description
package	GUID; string that identifies a specific package.

Results

On success, *GetVariants* returns an ArrayOfBrief XML, which has one Brief for each variant in the target package.

If no variants are present, the ArrayOfBrief is returned empty.

Example

This *GetVariants* returned a list of variants associated with the package identified with the GUID:

```
http://10.0.2.158:18000/Packages/GetVariants  
?package=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

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

```
<ArrayOfBrief>  
  <Brief>  
    <Description>HEVC</Description>  
    <Identifier>67118539-5869-413d-86df-e42582dc428a</Identifier>  
    <Name>HEVC5</Name>  
  </Brief>  
  <Brief>  
    <Description>HEVC</Description>  
    <Identifier>c49e8d6e-dc0d-4b7a-b756-71ccfa59861a</Identifier>  
    <Name>HEVC3</Name>  
  </Brief>  
</ArrayOfBrief>
```

```
<Description>AVC</Description>  
<Identifier>9ca0cd8c-2323-4d4d-9cf0-5a71b86c0c33</Identifier>  
<Name>AVC3</Name>  
</Brief>  
<Brief>  
<Description>AAC</Description>  
<Identifier>1bad0882-7cac-4cdd-b2a7-28dd909de467</Identifier>  
<Name>AAC</Name>  
</Brief>  
</ArrayOfBrief>
```

GetVariant

The purpose of this GET operation is to obtain details about a specific variant.

GetVariant has the following format:

```
http://<host>:18000/Packages/GetVariant?identifier={VARIANT_GUID}
```

Operation Chain

One of the following operations must be executed to obtain the required GUID before you can execute this operation. You'd need to do *GetPackages* first to get the package GUID

GetVariants (variant GUID)

or

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

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific variant.

Results

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

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

Example

This *GetVariant* returned the details about the target variant:

```
http://10.0.2.158:18000/Packages/GetVariant
?identifier=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

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

```
<VariantBrief>
  <Description i:nil="true"/>
  <Identifier>5ae52d4c-eb4e-4751-a4e5-ae3ae2b035f6</Identifier>
  <Name>AAC 128 Audio English</Name>
  <Details>
    <ParameterBrief>
      <Name>Default</Name>
      <Value>False</Value>
    </ParameterBrief>
  </Details>
  <Rendition>f3b2cd6c-d5a2-48af-9495-c343119c6112</Rendition>
```

```
<Streams xmlns:a="http://schemas.microsoft.com/2003/10/  
Serialization/Arrays">  
  <a:guid>5dd39185-e5b8-4687-8629-6222cfce2eb8</a:guid>  
</Streams>  
</VariantBrief>
```

Channels Operations

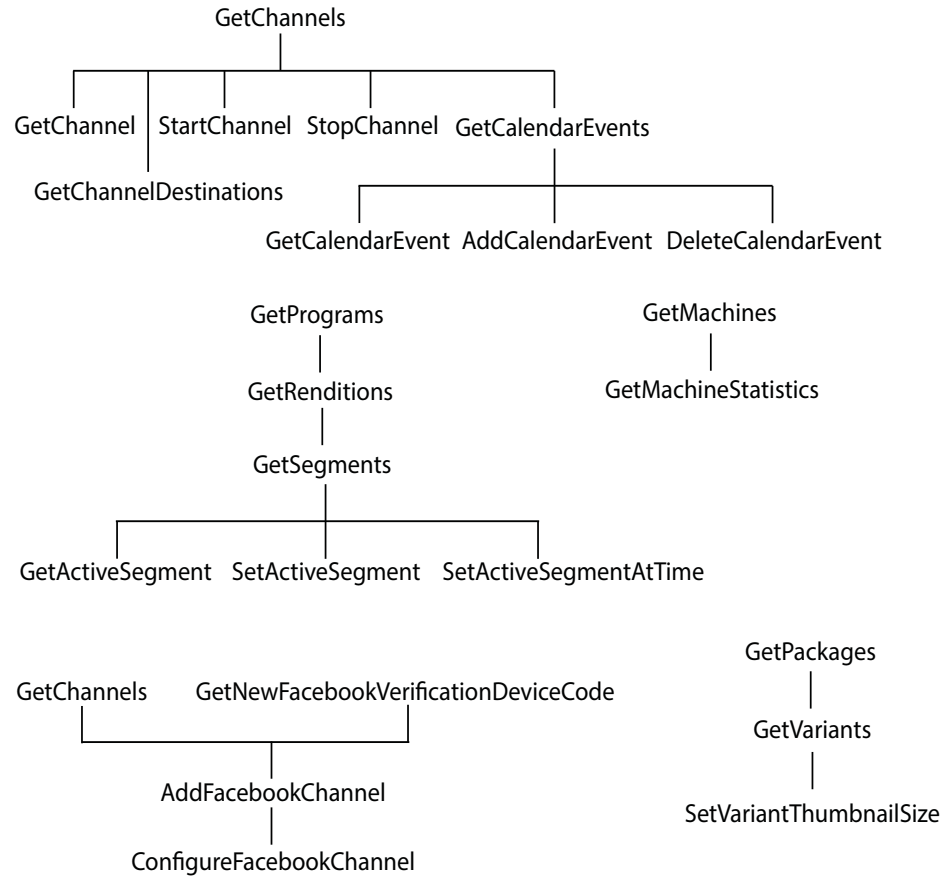
The Channels category of operations enable you to identify all of the channels in your Live Stream server. You can use these operations to query and control each channel and work with channel calendar events, and you can set the active segment on a channel.

- [GetChannels](#)
- [GetChannel](#)
- [GetChannelOutputLocations](#)
- [SetVariantThumbnailSize](#)
- [GetChannelThumbnail](#)
- [StartChannel](#)
- [StopChannel](#)
- [GetCalendarEvents](#)
- [GetCalendarEvent](#)
- [AddCalendarEvent](#)
- [DeleteCalendarEvent](#)
- [GetActiveSegment](#)
- [SetActiveSegment](#)
- [SetActiveSegmentAtTime](#)
- [GetMachineStatistics](#)
- [GetChannelStatistics](#)
- [GetNewFacebookVerificationDeviceCode](#)
- [AddFacebookChannel](#)
- [ConfigureFacebookChannel](#)

Note: All Channels operations start with `http://<host>:18000/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 diagram, the Channels operations are organized hierarchically in four different trees, based on their GUID requirements. Each operation in this category requires a channel GUID to execute correctly.



Note: To display help for Channels operations, go to <http://<Live Server DNS|IP Address>:18000/Channels/help>.

GetChannels

The purpose of this GET operation is to obtain a list of channels on the target Live Stream system. No parameters are required.

GetChannels has the following format:

```
http://<host>:18000/Channels/GetChannels
```

Operation Chain

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

Results

On success, *GetChannels* returns an ArrayOfBrief XML, with one Brief for each channel in the Live Stream system.

If there are no channels, the ArrayOfBrief is returned empty.

Example

This *GetChannels* returns a list of channels in the Live Stream system, accessed through the server with IP address 10.0.2.158:

```
http://10.0.2.158:18000/Channels/GetChannels
```

Typical Response

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

```
<ArrayOfBrief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>9378dd79-a374-46f8-9f23-dc68d8d52d07</Identifier>
    <Name>Northwest Passage</Name>
  </Brief>
  <Brief>
    <Description i:nil="true"/>
    <Identifier>d5d2a977-68c9-4fa3-a687-5d60535d4958</Identifier>
    <Name>NWP</Name>
  </Brief>
</ArrayOfBrief>
```

GetChannel

The purpose of this GET operation is to obtain details about a specific channel on the target Live Stream server.

GetChannel has the following format:

```
http://<host>:18000/Channels/GetChannel?identifier={CHANNEL GUID}
```

Operation Chain

The following operations must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific channel.

Results

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

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

Example

This *GetChannel* returns details about the target channel:

```
http://10.0.2.158:18000/Channels/GetChannel
?identifier=9378dd79-a374-46f8-9f23-dc68d8d52d07
```

Typical Response

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

```
<ChannelBrief>
  <Description i:nil="true"/>
  <Identifier>9378dd79-a374-46f8-9f23-dc68d8d52d07</Identifier>
  <Name>Northwest Passage</Name>
  <Active>false</Active>
  <Assignments/>
  <CalendarEvents>
    <Briefs>
      <Brief>
        <Description i:nil="true"/>
        <Identifier>2f5df131-deaf-4789-87dd-7ff850bcf1dc</
Identifier>
        <Name>Scheduled Stream Activation</Name>
      </Brief>
```



```
</Briefs>
</CalendarEvents>
<Machine>ad98ae66-25e2-480c-aaba-cf38857ce677</Machine>
<PrimaryPackage>
  <Description i:nil="true"/>
  <Identifier>af06a78f-d467-4112-bd28-671a0eca5bcb</Identifier>
  <Name>General outputs locations</Name>
  <Details>
    <ParameterBrief>
      <Name>Output Location</Name>
      <Value>Local Origin Server</Value>
    </ParameterBrief>
  </Details>
  <Package>eb353608-44f5-4b1b-bcca-43fef8e9bb60</Package>
</PrimaryPackage>
<SecondaryPackage/>
</ChannelBrief>
```

GetChannelOutputLocations

The purpose of this GET operation is to obtain the destinations for the target channel.

GetChannelOutputLocations has the following format:

```
http://<host>:18000/Channels/
GetChannelOutputLocations?channel={CHANNEL GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies a specific channel.

Results

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

Example

This *GetChannelOutputLocations* returned the destination associated with the channel identified with the GUID:

```
http://10.0.2.158:18000/Channels/GetChannelOutputLocations
?package=1bad0882-7cac-4cdd-b2a7-28dd909de467
```

Typical Response

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

```
<ArrayOfOutputConfigurationBrief>
  <OutputConfigurationBrief>
    <Description i:nil="true"/>
    <Identifier>af06a78f-d467-4112-bd28-671a0eca5bcb</Identifier>
    <Name>General outputs locations</Name>
    <Details>
      <ParameterBrief>
        <Name>Output Location</Name>
        <Value>Local Origin Server</Value>
      </ParameterBrief>
    </Details>
    <Package>eb353608-44f5-4b1b-bcca-43fef8e9bb60</Package>
  </OutputConfigurationBrief>
</ArrayOfOutputConfigurationBrief>
```

SetVariantThumbnailSize

The purpose of this SET operation is to specify the size of thumbnails obtained from the target variant. Variants can be shared across channels, so *SetVariantThumbnailSize* affects all instances of this variant in every channel where it is used.

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).

SetVariantThumbnailSize has the following format:

```
http://<host>:18000/Channels/SetVariantThumbnailSize  
?variant={VARIANT_GUID}&width={INT}&height={INT}
```

Operation Chain

The following operations must be executed in order to obtain the required GUID for this operation:

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

Parameters

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

This *SetVariantThumbnailSize* specifies that thumbnails obtained from the target thumbnail should be set at 1280 x 720:

```
http://10.0.25.162:18000/Channels/SetVariantThumbnailSize  
?variant=b0e7a6c0-a920-4702-b4f7-5fafa599a26c  
&width=1280&height=720
```

Typical Response

In this response, the operation was successful:

```
<Status>Set thumbnail size for variant 720p-X-AVC.</Status>
```

GetChannelThumbnail

The purpose of this GET operation is to obtain a thumbnail of a specific variant of the target channel. The operation returns a JPEG, which you can display (render) or save as a file.

Note: Channels that are not active do not have thumbnails.

GetChannelThumbnail has the following format:

```
http://<host>:18000/Channels/GetChannelThumbnail  
?channel={CHANNEL GUID}&variant={VARIANT GUID}
```

Operation Chain

The following operations must be executed in order to obtain the required GUIDs for this operation:

[GetChannels](#) (channel GUID)

[GetPackages](#) (package GUID) > [GetVariants](#) (variant GUID)

Parameters

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

Results

On success, *GetchannelThumbnail* returns a JPEG.

If the channel you are targeting does not have a thumbnail, an HTTP 404 error is returned.

Example

This *GetChannelThumbnail* returned a JPEG associated with the target channel and variant:

```
http://10.0.2.158:18000/Channels/GetChannelThumbnail  
?channel=27602854-35a6-4f0c-827c-ebd7ac5d40a9  
&variant=b0e7a6c0-a920-4702-b4f7-5fafa599a26c
```

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: Channels are defined for a hardware port on the server where they are created. Thus the host (DNS|IP address) that you use must be that of the target server.

StartChannel has the following format:

```
http://<host>:18000/Channels/StartChannel?channel={CHANNEL GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies a specific channel.

Results

On success, *StartChannel* returns a Status XML indicating success: `<Status>Channel successfully started.</Status>`

Example

This *StartChannel* started streaming the channel specified by the GUID:

```
http://10.0.2.158:18000/Channels/StartChannel  
?channel=d5d2a977-68c9-4fa3-a687-5d60535d4958
```

Typical Response

In this response, the operation was successful:

```
<Status>Channel successfully started.</Status>
```

StopChannel

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

Note: Channels are defined for a hardware port on the server where they are created. Thus the host (DNS|IP address) that you use must be that of the target server.

StopChannel has the following format:

```
http://<host>:18000/Channels/StopChannel?channel={CHANNEL_GUID}
```

When successful, *StopChannel* returns a Status XML.

Operation Chain

The following operations must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies a specific channel.

Results

On success, *StopChannel* returns a Status XML indicating success: `<Status>Channel successfully stopped.</Status>`

Example

This *StopChannel* terminated the streaming of the channel specified by the GUID:

```
http://10.0.2.158:18000/Channels/StopChannel  
?channel=d5d2a977-68c9-4fa3-a687-5d60535d4958
```

Typical Response

In this response, the operation was successful:

```
<Status>Channel successfully stopped.</Status>
```

GetCalendarEvents

The purpose of this GET operation is to obtain a list of CalendarEvents that comprise the target channel.

GetCalendarEvents has the following format:

```
http://<host>:18000/Channels/GetCalendarEvents?channel={CHANNEL  
GUID}
```

Operation Chain

The following operations must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies a specific channel.

Results

On success, *GetCalendarEvents* returns an ArrayOfBrief XML, which has one Brief for each calendar event in the target channel.

If no calendar events are present, the ArrayOfBrief is returned empty.

Example

This *GetCalendarEvents* returned a list of Brief elements; one for each calendar event in the channel identified with the GUID:

```
http://10.0.2.158:18000/Channels/GetCalendarEvents  
?channel=9378dd79-a374-46f8-9f23-dc68d8d52d07
```

Typical Response

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

```
<ArrayOfBrief>  
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>2f5df131-deaf-4789-87dd-7ff850bcf1dc</Identifier>  
  <Name>Scheduled Stream Activation</Name>  
</Brief>  
</ArrayOfBrief>
```

GetCalendarEvent

The purpose of this GET operation is to obtain details about a specific calendar event in the target channel.

GetCalendarEvent has the following format:

```
http://<host>:18000/Channels/  
GetCalendarEvent?identifier={CalendarEvent GUID}
```

Operation Chain

The following operations must be executed to obtain the required GUIDs before you can execute this operation:

[GetChannels](#) (channel GUID) > [GetCalendarEvents](#) (calendar event GUID)

Parameters

Parameter	Description
identifier	GUID; string that identifies a specific calendar event.

Results

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

If no calendar event exists with the specified GUID, an empty XML is returned.

Example

This *GetCalendarEvent* returned the *CalendarEventBrief* associated with the *CalendarEvent* identified with the GUID:

```
http://10.0.2.158:18000/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, and the start and end timestamps.

```
<CalendarEventBrief>  
  <Description i:nil="true"/>  
  <Identifier>2f5df131-deaf-4789-87dd-7ff850bcf1dc</Identifier>  
  <Name>Scheduled Stream Activation</Name>  
  <Color>  
    <primary>green</primary>  
    <secondary i:nil="true"/>  
  </Color>  
  <Details>  
    <ParameterBrief>  
      <Name>Frequency</Name>
```



```
    <Value>Does Not Repeat</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>By day</Name>
    <Value/>
  </ParameterBrief>
</Details>
<End>2017-03-28T14:29:52.4426153-07:00</End>
<Start>2017-03-28T13:29:52.4426153-07:00</Start>
<Title i:nil="true"/>
</CalendarEventBrief>
```

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.

AddCalendarEvent has the following format:

```
http://<host>:18000/Channels/AddCalendarEvent
?name={NAME}&channel={CHANNEL GUID}&start={START}&end={END}
```

Operation Chain

The following operations must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
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.
channel	GUID; string that identifies this channel.
start	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.
end	String; identifies the date and time to stop the channel, as described above in start.

Results

On success, *AddCalendarEvent* returns an *ArrayOfBrief* XML with the GUID of the new event.

Example

This *AddCalendarEvent* created a new event in the NWP channel (identified with its GUID) that starts on March 15, 2017 at 9:00 AM and ends at 10:00 AM on the same day:

```
http://10.0.2.258:18000/Channels/AddCalendarEvent
?channel=9378dd79-a374-46f8-9f23-dc68d8d52d07
&start=03-15-2017 09:00:00&end=03-15-2017 10:00:00
```

Typical Response

In this response, the new event was successfully added to the channel; its Identifier is returned, which you can use to target the event for further utilization.

```
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>a82a7f8e-d3c1-4bab-8644-67a4f5917a52</Identifier>  
  <Name i:nil="true"/>  
</Brief>
```

DeleteCalendarEvent

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

DeleteCalendarEvent has the following format:

```
http://<host>:18000/Channels/DeleteCalendarEvent  
?channel={CHANNEL GUID}&calendar-event={CALENDAR EVENT GUID}
```

Operation Chain

The following operations must be executed to obtain the required GUID before you can execute this operation:

[GetChannels](#) (channel GUID) > [GetCalendarEvents](#) or [AddCalendarEvent](#) (calendar event GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies this channel.
calendar-event	GUID; string that identifies this calendar event.

Results

On success, *DeleteCalendarEvent* returns a Status XML: `<Status>Calendar event successfully deleted.</Status>`.

Example

This *DeleteCalendarEvent* deleted an event in the NWP channel (identified with its GUID):

```
http://10.0.2.258:18000/Channels/DeleteCalendarEvent  
?channel=9378dd79-a374-46f8-9f23-dc68d8d52d07  
&calendar-event=a82a7f8e-d3c1-4bab-8644-67a4f5917a52
```

Typical Response

In this response, the new event was successfully deleted; a successful Status is returned.

```
<Status>Calendar event successfully deleted.</Status>
```

GetActiveSegment

The purpose of this GET operation is to obtain details about the active segment on the target channel.

GetActiveSegment has the following format:

```
http://<host>:18000/Channels/GetActiveSegment?  
channel={CHANNEL GUID}
```

Operation Chain

The following operation must be executed to obtain the required GUID before you can execute this operation.

[GetChannels](#) (Channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies this channel.

Results

On success, *GetActiveSegment* returns a Brief XML, with details about the current active segment on the target channel.

If the channel isn't active, it returns an error:

```
<Error>Channel is not active.</Error>.
```

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

Example

This *GetActiveSegment* returns details about the active segment on the target channel:

```
http://10.0.2.158:18000/Channels/GetActiveSegment?  
channel=68db056e-6ab7-4898-a6cd-5c888422606d
```

Typical Response

In this response, the active segment is identified by its Identifier, and Name.

```
<Brief>  
  <Description i:nil="true"/>  
  <Identifier>509a4f9a-bab2-41b9-ba26-56ec295c18d4</Identifier>  
  <Name>Segment2</Name>  
</Brief>
```

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 using [StartChannel](#) or manually in the Web app. 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 segment 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).

SetActiveSegment has the following format:

```
http://<host>:18000/Channels/SetActiveSegment
?channel={CHANNEL GUID}&segment={SEGMENT GUID}
```

Operation Chain

For the Channel GUID: [GetChannels](#)

For the Segment GUID: [GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID)
> [GetSegments](#) (segment GUID)

Parameters

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

Results

On success, *SetActiveSegment* returns a Status XML, with the success message:

```
<Status>Segment successfully triggered.</Status>.
```

If the channel is not active, an error is returned:

```
<Error>Channel is not active.</Error>.
```

Example

This *SetActiveSegment* returned a Status XML:

```
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 Status XML reports the successful operation.

```
<Status>Segment successfully triggered.</Status>
```

SetActiveSegmentAtTime

The purpose of this POST operation is to schedule the specified segment to start broadcasting at the specified time for the current date (no date is specified). The channel must be running—that is, you must have started the channel using [StartChannel](#) or manually in the Web app. 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.

SetActiveSegment has the following format:

```
http://<host>:18000/Channels/SetActiveSegment
?channel={CHANNEL GUID}&segment={SEGMENT GUID}&time={HH:MM:SS}
```

Operation Chain

For the Channel GUID: [GetChannels](#)

For the Segment GUID: [GetPrograms](#) (program GUID) > [GetRenditions](#) (rendition GUID)
> [GetSegments](#) (segment GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies this channel.
segment	GUID; string that identifies this segment.
time	Time code in format HH:MM:SS in 24-hour time; time at which the segment is scheduled to begin.

Results

On success, *SetActiveSegmentAtTime* returns a Status XML, with the success message:

```
<Status>Successfully added trigger.</Status>
```

If the channel is not active, an error is returned:

```
<Error>Channel is not active.</Error>
```

Example

This *SetActiveSegment* returned a Status XML:

```
http://10.0.25.158:18000/Channels/SetActiveSegment
?channel=ad1c45b7-67fb-419d-8c5b-8ba474bd6dfd
```



```
&segment=1d20e392-8876-411a-9681-70e08e7baca9  
&time=10:40:00
```

Typical Response

In this response, the Status XML reports the successful operation.

```
<Status>Successfully added trigger.</Status>.
```

GetMachineStatistics

The purpose of this GET operation is to obtain statistical information about the target server. Statistics are only available when a channel on the target server is streaming.

GetMachineStatistics has the following format:

```
http://<host>:18000/Channels/GetMachineStatistics?
machine={MACHINE_GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetMachines](#) (machine GUID)

Parameters

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

Results

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

If the machine GUID does not exist, an error is returned:

```
<Error>Failed to generate statistics. No channels belonging to
machine b6347bb5-c8c8-4bb1-8cec-53342ca6225d are currently
active.</Error>
```

Example

This *GetMachineStatistics* returned a MachineStatisticsBrief XML:

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

Typical Response

In this response, the MachineStatisticsBrief XML reports a variety of statistics:

```
<MachineStatisticsBrief>
  <Description i:nil="true"/>
  <Identifier>ff0f694b-3f17-4b8f-af13-40486d488016</Identifier>
  <Name>LS-SVR</Name>
  <CpuUsage>5</CpuUsage>
  <DiskSpaceUsed>unknown</DiskSpaceUsed>
  <GpuComputeUtilization>11</GpuComputeUtilization>
  <GpuEncoderUtilization>0</GpuEncoderUtilization>
  <GpuMemory>6</GpuMemory>
  <Memory>35</Memory>
</MachineStatisticsBrief>
```

GetChannelStatistics

The purpose of this GET operation is to obtain statistical information about the target active channel.

GetChannelStatistics has the following format:

```
http://<host>:18000/Channels/GetChannelStatistics  
?channel={CHANNEL GUID}
```

Operation Chain

The following operation must be executed in order to obtain the required GUID for this operation:

[GetChannels](#) (channel GUID)

Parameters

Parameter	Description
channel	GUID; string that identifies this channel.

Results

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

If the channel GUID does not exist, an error is returned:

```
<Error>Could not find channel with identifier 68db056e-6ab7-4898-  
a6cd-5c888422606d.</Error>
```

If the channel is not active, an error is returned:

```
<Error>Channel is not active.</Error>
```

Example

This *GetChannelStatistics* returned a ChannelStatisticsBrief XML with real-time statistical information:

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

Typical Response

In this response, the ChannelStatisticsBrief XML reports a variety of statistics:

```
<ChannelStatisticsBrief>  
  <Description i:nil="true"/>  
  <Identifier>68db056e-6ab7-4898-a6cd-5c888422606d</Identifier>  
  <Name>HLS 1</Name>  
  <CpuUsage>2</CpuUsage>  
  <GpuMemory>0</GpuMemory>
```

```
<Memory>1817</Memory>  
<OutputLocation>Local Origin Server</OutputLocation>  
<Uptime>0:35:19</Uptime>  
</ChannelStatisticsBrief>
```

GetNewFacebookVerificationDeviceCode

The purpose of this GET operation is to obtain a new device code which you can use to authenticate a new Facebook user.

Note: After executing this operation, you must enter the URL returned to display the authentication page and use the device code returned to authenticate this use.

GetNewFacebookVerificationDeviceCode has the following format:

`http://<host>:18000/Channels/GetNewFacebookVerificationDeviceCode`

There are no parameters required for this operation.

Results

On success, *GetNewFacebookVerificationDeviceCode* returns a `FacebookAuthenticationBrief` XML, with a new device code.

Example

This *GetNewFacebookVerificationDeviceCode* returned a Status XML:

`http://livestr:18000/Channels/GetNewFacebookVerificationDeviceCode`

Typical Response

In this response, the `FacebookAuthenticationBrief` XML includes the new device code you can use to authenticate streaming.

```
<FacebookAuthenticationBrief>
  <Description/>
  <Identifier>00000000-0000-0000-0000-000000000000</Identifier>
  <Name">Facebook Authentication</Name>
  <DeviceCode>RR8JOGT7</DeviceCode>
  <VerificationURI>https://www.facebook.com/device</
VerificationURI>
</FacebookAuthenticationBrief>
```

AddFacebookChannel

The purpose of this POST operation is to add a new Facebook channel to the Live Stream system.

AddFacebookChannel has the following format:

```
http://<host>:18000/Channels/AddFacebookChannel?  
name={NAME}&package={PACKAGE GUID}&userName={USERNAME}
```

or

```
http://<host>:18000/Channels/AddFacebookChannel?  
name={NAME}&package={PACKAGE GUID}&deviceCode={DEVICECODE}
```

Note: After authenticating your account manually in Facebook, you must supply the device code for this operation one time, to authenticate your account using Live Stream as well. After executing this operation one time, you should always use the `userName` parameter, supplying the user name of the account.

Operation Chain

The following operations must be executed in order to obtain the required device code and package GUID for this operation:

To obtain the device code: [GetNewFacebookVerificationDeviceCode](#) (device code). This device code must be used directly in Facebook before being used in this operation.

To obtain the package GUID: [GetPackages](#) (package GUID).

Parameters

Parameter	Description
name	String that identifies this channel; displayed in the Web app.
package	String; GUID that identifies the primary package; obtained from GetPackages .
userName	String that identifies the user of an authenticated Facebook account. When using <code>userName</code> , do not use <code>deviceCode</code> .
deviceCode	Code returned to Live Stream as a result of the GetNewFacebookVerificationDeviceCode operation; required to authenticate a new Facebook account one time. When authenticating, do not include <code>userName</code> .

Results

On success, *AddFacebookChannel* returns a ChannelBrief XML.

If the device code you use is not valid, this error is returned:

```
<Error>An error occurred while creating channel: Invalid Facebook verification  
device code.</Error>
```

If the user you supply is not authenticated, this error is returned:

```
<Error>PuppyDogma_N45PLJ is not authenticated for Facebook Live.</Error>
```

Example

This *AddFacebookChannel* returned a *ChannelBrief* XML, indicating success:

```
http://LS-SVR:18000/Channels/AddFacebookChannel?  
name=FBChannel007&package=1af0ba42-de0e-48d0-8fa4-db14401e5bda&  
userName=LiveStreamer&deviceCode=J24EMZRY
```

Typical Response

In this response, the *ChannelBrief* XML reports the details of adding the new *FBChannel007* channel to the Live Stream system.

```
<ChannelBrief>  
  <Description i:nil="true"/>  
  <Identifier>b01c8be8-6c0e-492e-ba1c-12d78d77687f</Identifier>  
  <Name>FBChannel007</Name>  
  <Active>false</Active>  
  <Assignments/>  
  <CalendarEvents>  
    <Briefs/>  
  </CalendarEvents>  
  <Machine>4b6f71b5-65dd-4df1-a4fb-209139737c21</Machine>  
  <PrimaryPackage>  
    <Description i:nil="true"/>  
    <Identifier>90ad1d8f-d10b-425e-b39d-a7e447cc766b</Identifier>  
    <Name>FacebookLive Outputs</Name>  
    <Details>  
      <ParameterBrief>  
        <Name>User Name</Name>  
        <Value>LiveStreamer</Value>  
      </ParameterBrief>  
    </Details>  
    <Package>1af0ba42-de0e-48d0-8fa4-db14401e5bda</Package>  
  </PrimaryPackage>  
  <SecondaryPackage/>  
</ChannelBrief>
```

ConfigureFacebookChannel

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

ConfigureFacebookChannel has the following format:

```
http://<host>:18000/Channels/ConfigureFacebookChannel?  
channel={CHANNEL_GUID}&startDate={STARTDATE}&title={TITLE}&  
description={DESCRIPTION}&postToPage={POSTTOPAGE}&  
postToGroup={POSTTOGROUP}&postToEvent={POSTTOEVENT}&  
deleteVideo={DELETEVIDEO}&privacy={PRIVACY}
```

Operation Chain

The following operation must be executed in order to obtain the required channel GUID for this operation:

[GetChannels](#) (Facebook channel GUID)

Parameters

Parameter	Description
channel	String GUID; identifies this Facebook channel; obtained by GetChannels .
startDate (optional)	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 will be set to <i>Go Live Now</i> . If the startDate is included, the schedule is set to <i>Schedule for Later</i> .
title (optional)	String; displayed as the title of the Facebook Live event.
description (optional)	String; description displayed as the description of the event.

Parameter	Description
postToPage postToGroup postToEvent (optional)	The postToPage, postToGroup, and postToEvent parameters are mutually exclusive. Use only one (or none) to select where to post the video. The value of the parameter is a string of the target page, group, or event name. If none of these parameters are present, Post To is set to User.
deleteVideo (optional)	Keywords: True False to specify whether the video should be deleted after broadcast.
privacy (optional)	Keywords: Public Friends Only Me to specify the privacy level for this broadcast. This is only relevant if the Post To parameter is set to <i>User</i> . When posting to Pages, Groups, and Events, privacy is <i>Public</i> .

Results

On success, *ConfigureFacebookChannel* returns a Status XML, with the success message:

```
<Status>Successfully added trigger.</Status>
```

If the channel is not a Facebook channel, an error is returned:

```
<Error>Channel's primary package is not a Facebook package.</Error>
```

Example

This *ConfigureFacebookChannel* (which only sets the *deleteVideo* parameter to true) returns a ChannelBrief XML:

```
http://10.0.25.158:18000/Channels/Configurefacebookchannel?channel=90e90139-cace-4d10-8024-6533b1b74edd&deleteVideo=true
```

Typical Response

In this response, the Status XML reports the successful operation.

```
<ChannelBrief>
  <Description i:nil="true"/>
  <Identifier>90e90139-cace-4d10-8024-6533b1b74edd</Identifier>
  <Name>FBChannel</Name>
  <Active>>false</Active>
  <Assignments/>
  <CalendarEvents>
    <Briefs/>
  </CalendarEvents>
  <Machine>4b6f71b5-65dd-4df1-a4fb-209139737c21</Machine>
  <PrimaryPackage>
    <Description i:nil="true"/>
    <Identifier>90ad1d8f-d10b-425e-b39d-a7e447cc766b</Identifier>
    <Name>FacebookLive Outputs</Name>
    <Details>
      <ParameterBrief>
        <Name>User Name</Name>
```

```
    <Value>Telestream Tester A</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Schedule</Name>
    <Value>Go Live Now</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Event Title</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Broadcast Description (Optional)</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Delete Video from Facebook After Broadcast</Name>
    <Value>True</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>StreamId</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Status</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Stream URL</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Stream Name</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Secure Stream URL</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Secure Stream Name</Name>
    <Value/>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Post To</Name>
    <Value>User</Value>
  </ParameterBrief>
  <ParameterBrief>
    <Name>Privacy</Name>
    <Value/>
  </ParameterBrief>
</Details>
<Package>1af0ba42-de0e-48d0-8fa4-dbd14401e5bda</Package>
</PrimaryPackage>
<SecondaryPackage/>
</ChannelBrief>
```