



Wowza Streaming Engine™

User's Guide

Wowza Streaming Engine: User's Guide



Version: 4.0.0

<http://www.wowza.com>

This document is for informational purposes only and in no way shall be interpreted or construed to create warranties of any kind, either express or implied, regarding the information contained herein.

No Endorsement or Warranty for Third-Party Links and Software

This document contains links to third-party websites ("Linked Sites") that are not under the control of Wowza® Media Systems, LLC ("Wowza"). Wowza is not responsible for the content on or operation of Linked Sites. If you access Linked Sites, you do so at your own risk and understand that Wowza accepts no responsibility or liability for the content or operation of Linked Sites. Wowza provides these links only as a convenience, and the inclusion of a link does not imply that Wowza endorses such Linked Sites or any content, products, or services available from Linked Sites.

This document also refers to third-party software that is not licensed, sold, or distributed by Wowza (collectively, "Third-Party Software"). Wowza does not endorse, is not responsible for, and accepts no liability related to Third-Party Software. Please ensure that any and all use of Wowza® software and third-party software is properly licensed.

Wowza Trademarks

Wowza®, Wowza Media Systems, Wowza Streaming Engine™, along with other trademarks, logos, trade dress, and other proprietary colors and markings, are each trademarks or registered trademarks of Wowza in the United States and in other countries (collectively, "Wowza Marks"). No right to use Wowza Marks in any way is granted hereunder. Contact sales@wowza.com for information on obtaining the right to use Wowza Marks. Any use of Wowza Marks, authorized or otherwise, shall inure to the sole benefit of Wowza.

Third-Party Trademarks and Copyrights

Trademarks, product names, logos, designs, trade dress, and other proprietary markings of non-Wowza third parties (collectively, "Third-Party Marks") may be trademarks or registered trademarks of their respective owners. Use of Third-Party Marks is for the sole purpose of identifying third-party products and services and does not represent endorsement, sponsorship, partnership, or other affiliation between Wowza and such third parties.

A list of applicable copyright notices related to content in this document is available on the [Wowza website](#).

Document History

Version	Description	Release date
Doc v4.0.0	Initial document release for Wowza Streaming Engine 4.0	02-11-2014

Note

A more recent version of this document may be available online. See the [Wowza Media Systems Documentation webpage](#) for the latest updates.

Table of Contents

What's New	5
Wowza Streaming Engine Manager	5
MPEG-DASH	6
Media Cache.....	7
Push Publishing	7
New and Enhanced Performance Features	7
Introduction	9
Adobe HDS (Adobe Flash Player)	9
Apple HLS (iPhone, iPad, iPod touch, QuickTime, and More)	10
Microsoft Smooth Streaming (Microsoft Silverlight and More)	11
MPEG-DASH Streaming (DASH Clients)	12
Adobe RTMP (Adobe Flash Player).....	13
RTSP/RTP (QuickTime, VLC, 3GPP Devices, Set-top Boxes, and More)	14
Video and Audio Streaming, Recording, and Chat	15
Wowza Transcoder AddOn.....	15
Wowza nDVR AddOn	17
Wowza DRM AddOn.....	18
Downloadable AddOn Packages	19
Installed Examples.....	20
Wowza Streaming Engine Editions.....	22
Server Installation	23
Before Installation.....	23
Installing Wowza Streaming Engine	24
Starting and Stopping Wowza Streaming Engine	27
Running Wowza Streaming Engine as a Named User	30
Running Multiple Wowza Streaming Engine Instances.....	31
Entering a New License Key.....	31

Ports Used for Streaming	32
Server Configuration and Tuning	34
Software Updates	35
Application Configuration.....	36
Applications and Application Instances (Application.xml)	37
URL Formats	37
Stream Types	38
HTTP Streamers and Live Stream Packetizers	40
Timed Text Providers.....	43
Wowza Transcoder AddOn and Wowza nDVR AddOn Configurations	46
Modules	46
Properties	47
Media Types	48
Content Storage.....	50
Advanced Configuration Topics	52
MediaCasters, Stream Files, and Startup Streams.....	52
Live Stream Repeater (Origin/Edge Live Streaming).....	57
Live Stream Recording	59
Virtual Hosting	61
Server-side Publishing (Stream and Publisher Classes)	64
Server Management and Monitoring	65
Starting and Stopping Wowza Streaming Engine Manager	65
Managing Sign-In Credentials.....	69
Navigating in Wowza Streaming Engine Manager	70
Adobe Flash Streaming and Scripting.....	83
Streaming Basics.....	83
Pre-built Media Players.....	85
Bi-directional Remote Procedure Calls	86
Remote Shared Objects	87

Server-side Modules and HTTP Providers	89
Server-side Modules	89
HTTP Providers	92
Extending Wowza Streaming Engine Using Java	95
Custom Module Classes	95
HTTP Provider Classes	106
Event Listeners	107
Server Administration	108
Configuring SSL and RTMPS	108
Logging	109
Streaming Tutorials	116

What's New

What's new in the latest Wowza server software?

Wowza Streaming Engine™, formerly known as Wowza Media Server®, is robust, customizable, and scalable server software that powers reliable streaming of high-quality audio and video to any device anywhere. This release of the server software has a new name as well as new and updated features that continue to future-proof and simplify online video delivery by expanding its ability to stream video to any screen using any streaming protocol, streamlining server administration, and providing enhanced scalability, flexibility, and reliability.

Wowza Streaming Engine Manager

Wowza Streaming Engine Manager enables you to easily configure, manage, and monitor the Wowza Streaming Engine software from a web browser on your computer, tablet, or phone. Although programmatic and command-line configuration and management of Wowza Streaming Engine is still available, the new browser-based manager offers the following key benefits that enable publishers with a diverse range of technical abilities to have greater control and confidence when streaming video:

- Create live and video on demand (VOD) streaming applications.
- Configure streams to start automatically when the server starts.
- Create and manage SMIL files for adaptive bitrate streaming.
- Manage IP address/port-based virtual hosting environments.
- Configure and manage user name/password authentication when publishing a stream from RTMP/RTSP-based encoders to the Streaming Engine.

- Manage integrated security options for client publishing and playback.
- Replace complex stream names by configuring stream files and assigning aliases.
- Instantly scale VOD streaming content using built-in Media Cache technology and live streaming content by configuring live stream repeater (origin/edge) technology.
- Record incoming streams for later on-demand playback.
- Test playback from applications or preview and monitor streams using built-in test players.
- Monitor Streaming Engine CPU, memory, Java heap, and disk usage, and monitor incoming and outgoing connections, network throughput, and uptime at server, virtual host, and application levels.
- Secure log-in to manage license keys for the Streaming Engine software or to set administrator/read-only permissions.
- Administrator-level configuration of advanced server-level and application-level properties and custom property settings.
- Leverage the built-in security features in Wowza Streaming Engine for easier and faster implementation of content security options.

You can use the Streaming Engine manager with the latest versions of most modern web browsers that support HTML5 and Cascading Style Sheets level 3 (CSS 3). We recommend that you use the Google Chrome browser. On Windows operating systems, if you have multiple browsers installed on your computer, you can ensure that the web application always opens in the browser that you want to use by configuring the **Default Programs** feature. The [Server Management and Monitoring](#) chapter in this document has more information about Wowza Streaming Engine Manager.

MPEG-DASH

Wowza® actively works with other industry leaders to ensure its server software supports the MPEG-DASH standard (ISO/IEC 23009-1) that's intended to streamline video delivery to any device. MPEG-DASH streaming is fully supported in Wowza Streaming Engine. In general, DASH streaming is similar to proprietary adaptive streaming technologies such as Apple HTTP Live Streaming (Apple HLS), Adobe HTTP Dynamic Streaming (Adobe HDS), and Microsoft Smooth Streaming. MPEG-DASH support in Wowza Streaming Engine includes:

- Deliver audio-only or video-only DASH streams.
- Deliver audio DASH streams that use the Dolby Digital Plus (Enhanced AC-3 or E-AC-3) audio codec.
- Deliver VOD and live DASH streams when using Wowza servers in an origin/edge configuration.
- Protect VOD and live DASH streams using Common Encryption (CENC).
- Playback on DASH-AVC/264-compatible players.

Media Cache

Media Cache technology is a read-through caching mechanism that enables scaling of VOD streams. Available as an AddOn for earlier versions of Wowza server software, it's built-in with Wowza Streaming Engine and can be configured in Wowza Streaming Engine Manager. For more information, see [How to scale video on demand streaming with Media Cache](#).

Push Publishing

Push Publishing AddOn technology for earlier versions of Wowza server software is built in with Wowza Streaming Engine. It enables streams to be pushed from the server to downstream Wowza servers running Wowza Media Server 3.5 (or later) and Wowza Streaming Engine using the WOWZ™ protocol (Wowza messaging protocol). It also enables streams to be pushed from the server to downstream Wowza servers running any Wowza server software version, Adobe Media Servers, and Content Delivery Networks (CDNs) using Real Time Messaging Protocol (RTMP), Real-time Transport Protocol (RTP), and MPEG Transport Stream Protocol (MPEG-TS). For more information, see [How to push streams to CDNs and other services](#).

New and Enhanced Performance Features

Wowza Streaming Engine includes significant new performance capabilities, making it the software of choice for organizations wanting to take full advantage of the growing power of video and audio streaming. Improvements in Wowza Streaming Engine 4.0 include:

- Automated tuning of performance settings for Java heap size, garbage collection (GC), thread pool sizes, and thread allocation for the server and virtual hosts (VHosts) in development and production environments enables the best possible performance out of existing hardware without manual intervention. You can adjust any of these settings in Wowza Streaming Engine Manager.
- Support for version 3 of the NVIDIA NVENC SDK enables server resources to be used more efficiently when using Wowza Transcoder AddOn to transcode video on 64-bit Windows and Linux operating systems.
- Support for the Intel Media SDK 2013 enables accelerated video processing without a separate GPU card when using Wowza Transcoder AddOn to transcode video on 64-bit Windows and Linux operating systems. It also enables the transcoder to take advantage of the upcoming 4th-generation Intel Core ("Haswell") processors.

Introduction

What is Wowza Streaming Engine?

Wowza Streaming Engine™ is high-performance, extensible, and fully interactive media streaming software platform that provides live and on-demand streaming, chat, and remote recording capabilities to a wide variety of media player technologies. The Streaming Engine can deliver content to many popular media players such as Adobe Flash Player; Microsoft Silverlight player; Apple iPhone, iPad, and iPod touch and Apple QuickTime player (version 10 or later); Android smartphones and tablets; and IPTV/OTT set-top boxes. Wowza Streaming Engine includes support for many streaming protocols including Adobe HTTP Dynamic Streaming (Adobe HDS), Apple HTTP Live Streaming (Apple HLS), Microsoft Smooth Streaming, MPEG-DASH streaming, MPEG-2 Transport Streams (MPEG-TS), Real Time Messaging Protocol (RTMP), Real Time Streaming Protocol (RTSP), and Real-time Transport Protocol (RTP). It's an alternative to the Adobe Media Server, Darwin Streaming Server, Microsoft IIS Media Services, and other media servers.

For the most up-to-date information, tutorials, and tips, see the [Wowza Articles and Forums](#).

To get started quickly with Wowza Streaming Engine, see the [Quick Start Guide](#).

Adobe HDS (Adobe Flash Player)

Wowza Streaming Engine can stream adaptive bitrate live and video on demand (VOD) content to Adobe Flash Player 10.1 or later using the Adobe HTTP Dynamic Streaming (Adobe HDS) protocol. Adobe HDS is a chunk-based streaming protocol that uses HTTP for delivery. All media-chunking and packaging necessary to deliver a stream using this protocol is performed by the Streaming Engine. Adobe HDS is referred to as "San Jose" streaming in the Streaming Engine configuration files.

When streaming VOD content, Wowza Streaming Engine supports MP4 files (QuickTime container) and MP3 files. FLV files aren't supported. The Streaming Engine supports the following video and audio codecs when using this streaming protocol:

Video

- H.264
- On2 VP6 (live only)
- Screen video and Screen video 2 (live only)
- Sorenson Spark (live only)

Audio

- AAC, AAC Low Complexity (AAC LC), AAC High Efficiency (HE-AAC) v1 and v2
- MP3
- Nellymoser Asao (live only)
- Speex (live only)

Apple HLS (iPhone, iPad, iPod touch, QuickTime, and More)

Wowza Streaming Engine can stream adaptive bitrate live and VOD H.264, AAC, and MP3 content to iOS-based devices (iPhone/iPad/iPod touch iOS version 3.0 or later), QuickTime player (version 10 or later), Safari browser (version 4.0 or later), and other devices such as the Roku and Amino set-top boxes and some brands of smart TVs using the Apple HTTP Live Streaming (Apple HLS) protocol. Apple HLS is a chunk-based streaming protocol that uses HTTP for delivery. All media-chunking and packaging necessary to deliver a stream using this protocol is performed by the Streaming Engine. Apple HLS is referred to as "Cupertino" streaming in the Streaming Engine configuration files.

Wowza Streaming Engine supports multiple encryption methods for protecting Apple HLS streams using DRM. For more information, see [How to secure Apple HLS streaming using DRM encryption](#).

Wowza Streaming Engine can send timed data events to the iOS player in the form of ID3 tags. For more information, see [How to convert OnTextData events in a live or vod stream to timed events \(ID3 tags\) in an Apple HTTP stream](#).

Wowza Streaming Engine populates the playlist file with metadata that describes each of the available streams in an adaptive bitrate presentation. This enables iOS-based players to intelligently select the appropriate streams based on hardware device capabilities.

The iPhone, iPad, and iPod touch (iOS devices) and Apple TV digital media extender support the following media formats:

Video

- H.264

Audio

- AAC, AAC Low Complexity (AAC LC), High Efficiency AAC (HE-AAC) v1
- Dolby Digital 5.1 Surround Sound (AC-3) and Dolby Digital Plus (Enhanced AC-3 or E-AC-3)
- MP3

Microsoft Smooth Streaming (Microsoft Silverlight and More)

Wowza Streaming Engine can stream adaptive bitrate live and VOD H.264, AAC, and MP3 content to Microsoft Silverlight, Windows Phone devices, and other devices using the Microsoft Smooth Streaming protocol. Microsoft Silverlight is a cross-browser, cross-platform technology that exists on many personal computing devices. Smooth Streaming is a chunk-based streaming protocol that uses HTTP for delivery. All media chunking and packaging necessary to deliver a stream using this protocol is performed by the Streaming Engine so there's no need for an IIS web server.

The following media formats can be used when streaming to Smooth Streaming clients:

Video

- H.264

Audio

- AAC, AAC Low Complexity (AAC LC), AAC High Efficiency (HE-AAC) v1 and v2
- Dolby Digital 5.1 Surround Sound (AC-3) and Dolby Digital Plus (Enhanced AC-3 or E-AC-3)
- MP3

MPEG-DASH Streaming (DASH Clients)

Dynamic Adaptive Streaming over HTTP (DASH), also known as *MPEG-DASH*, is a new international standard for adaptive streaming that's being adopted by the streaming industry. Wowza Streaming Engine includes MPEG-DASH technology for streaming live and VOD content over HTTP to select DASH clients.

MPEG-DASH is similar to proprietary adaptive streaming technologies such as Apple HLS, Adobe HDS, and Microsoft Smooth Streaming in that it's a chunk-based streaming technology that uses HTTP for delivery. All media-chunking and packaging necessary to deliver a stream using this technology is performed by the Streaming Engine. Note that in MPEG-DASH terminology, chunks are called *segments*.

An MPEG-DASH server provides DASH clients with a list of the available media chunk URLs in a Media Presentation Description (MPD) manifest. The MPD describes chunk information such as timing, language, timed text, and media characteristics (video resolution and bitrate). Clients sequentially request media chunks based on network conditions, device capabilities, and other factors to enable uninterrupted playback of the adaptive bitrate media presentation.

The MPEG-DASH standard ([ISO/IEC 23009-1](#)) defines segment container formats for ISO Base Media File Format (ISOBMFF) and MPEG-2 Transport Streams (MPEG-2 TS). MPEG-DASH is codec-agnostic and supports multiplexed and non-multiplexed encoding. Multiple content protection (DRM) schemes are supported; however, a Common Encryption (CENC) standard ([ISO/IEC 23001-7](#)) is being developed in conjunction with MPEG-DASH to allow content to be encrypted once and then streamed to DASH clients that support different licensing systems.

The following media formats can be used when streaming to DASH clients:

Video

- H.264

Audio

- AAC, AAC Low Complexity (AAC LC), AAC High Efficiency (HE-AAC) v1 and v2
- Dolby Digital 5.1 Surround Sound (AC-3) and Dolby Digital Plus (Enhanced AC-3 or E-AC-3)

For more information about MPEG-DASH support in Wowza Streaming Engine, see [How to do MPEG-DASH streaming](#).

Adobe RTMP (Adobe Flash Player)

Wowza Streaming Engine communicates with Adobe Flash Player using the Real Time Messaging Protocol (RTMP). The Streaming Engine can deliver adaptive bitrate live and VOD content to Flash Player using RTMP and it supports other features such as shared objects, video recording, video chat, remote procedure calls, and more. The Streaming Engine supports all video and audio formats that Flash Player supports:

Video

- H.264
- On2 VP6
- Sorenson Spark
- Screen video and Screen video 2

Audio

- AAC, AAC Low Complexity (AAC LC), AAC High Efficiency (HE-AAC) v1 and v2
- MP3
- Speex
- Nellymoser Asao

Wowza Streaming Engine supports the following RTMP protocol variants:

- RTMP. The base protocol and the most efficient and fastest of the variants.
- RTMPE. A lightweight encryption variant that helps to secure the data being transmitted between the Stream Engine and Flash Player.
- RTMPS. An encryption variant that transmits data over a secure SSL connection. RTMPS uses a more robust encryption layer than RTMPE to wrap the RTMP session. Subscription and Perpetual licensees can use [Wowza StreamLock™ AddOn](#) to get free 256-bit SSL certificates for use with RTMP (RTMPS) and HTTP (HTTPS).
- RTMPT. A tunneling variant that is used to tunnel through firewalls that employ stateful packet inspection.
- RTMPTE. An encryption variant of the RTMPT protocol.

Wowza Streaming Engine includes bi-directional support for Action Message Format (AMF3 and AMF0) for data serialization (AMF3 was introduced in Flash Player 9 and ActionScript 3.0).

RTSP/RTP (QuickTime, VLC, 3GPP Devices, Set-top Boxes, and More)

Wowza Streaming Engine can stream live H.264, AAC, and MP3 content to players and devices that support the Real Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP), and MPEG-2 Transport Stream protocol (MPEG-2 TS). This includes players and devices such as QuickTime player (version 10 or later), VideoLAN VLC player, set-top boxes, and 3GPP devices. The Streaming Engine can also accept incoming streams from encoding devices that use these protocols. The Streaming Engine supports RTP and MPEG-2 TS input and output over UDP as well as multicast. In addition, the Streaming Engine supports interleaved RTSP/RTP (RTP over the RTSP TCP connection) and RTSP/RTP tunneling (RTSP/RTP over HTTP), which enables RTSP/RTP to be delivered in network environments that don't allow UDP transmission.

Wowza Streaming Engine supports the following RTSP, RTP, and MPEG specifications:

MPEG-TS	ISO/IEC 13818-1
MPEG-TS over RTP	rfc2038
RTP: AAC	rfc3640, rfc3016, ISO/IEC 14496-3
RTP: G.711	rfc3551
RTP: H.263	rfc2429
RTP: H.264	rfc3984, QuickTime Generic RTP Payload Format
RTP: MP3	rfc2250
RTP: MPEG-2 (video)	rfc2250
RTP: MPEG-4 Part 2	rfc3106
RTP: Speex	rfc5574
RTSP	rfc2326

Wowza Streaming Engine supports both Single Program (SPTS) and Multi Program (MPTS) MPEG-TS streams and enables you to specify a specific program, a specific language, and a specific audio or video track in an MPTS stream. Query parameters are part of the udp:// URL in a **.stream** file. There are four options for selecting a stream. For more information about how to use the query parameters, see [How to select MPEG-TS stream by program ID and also audio language by PID](#).

Video and Audio Streaming, Recording, and Chat

Wowza Streaming Engine can stream live and VOD content to many player technologies. It supports the following VOD file formats: MP4 (QuickTime container - .mp4, .f4v, .mov, .m4a, .m4v, .mp4a, .mp4v, .3gp, and .3g2), FLV (Flash Video - .flv), and MP3 content (.mp3). The Streaming Engine can accept live video and audio streams from encoders that support the RTMP, RTSP/RTP, native RTP, and MPEG-TS protocols and it can record any incoming live stream to either the MP4 or FLV format.

Wowza Streaming Engine can read and write Action Message Format (AMF0 and AMF3) data events to and from MP4 files. In addition, it supports MP4 multi-language caption and audio tracks.

Wowza Streaming Engine can be used to re-stream SHOUTcast and Icecast (AAC, AAC+, and MP3) audio streams and IP Camera (AAC, G.711 (μ -law and A-law), H.264, and MP3) streams to supported player technologies. It maintains a single connection to the original source stream while delivering the stream to multiple players. It can also forward embedded SHOUTcast and Icecast metadata, such as song title and artist, to Adobe Flash Player. The SHOUTcast example that's included with the Streaming Engine installation illustrates these capabilities.

Wowza Streaming Engine can deliver two-way video, audio, and text chat to Adobe Flash Player. This feature can be leveraged to deliver video conferencing applications or two-way messaging applications.

Wowza Transcoder AddOn

Wowza Transcoder is a real-time video transcoding and transrating solution that provides the ability to ingest a live stream, decode the video and audio, and then re-encode the stream for delivery to desired playback devices. It can decode and re-encode audio and video in multiple formats with key frames that are properly aligned for adaptive bitrate delivery. The following are some common scenarios:

- **Transcode:** Ingests a non-H.264 video and non-AAC/MP3 audio media stream and converts it to a set of H.263 or H.264 AAC adaptive bitrate media streams with properly aligned key frames.
- **Transrate:** Ingests an H.264 video and AAC/MP3 audio stream and creates a full set of bitrate renditions that are key frame aligned to the source stream for adaptive bitrate delivery.

- **Audio-only:** Ingests an H.264 video and Speex audio stream from Adobe Flash Player and converts the Speex audio format to AAC to make the stream compatible with additional player technologies.

You can apply static GIF, JPEG, PNG, and BMP overlay images to streams and customize the location, size, alignment, and opacity of the image to achieve stationary image effects such as a watermark to your video. In addition, you can use a Java-based API to apply dynamic overlay images to streams. The API can be configured manually or pre-programmed based on external events, making it a powerful tool for adding premium TV-like experiences. For more information, see [How to add graphic overlays to live streams with Wowza Transcoder AddOn](#).

Wowza Transcoder AddOn supports the following video and audio formats:

Video (decoding)

- H.264
- MPEG-2
- MPEG-4 Part 2

Video (encoding)

- H.263v2
- H.264

Audio (decoding)

- AAC
- G.711 (μ -law and A-law)
- MPEG-1 Layer 1/2
- MPEG-1 Layer 3 (MP3)
- Speex

Audio (encoding)

- AAC

For more information about Wowza Transcoder AddOn, see the [Wowza Transcoder AddOn User's Guide](#) and the [Wowza Transcoder Forum](#).

Note

Wowza Transcoder AddOn can be configured to take advantage of hardware acceleration, which is recommended but not required. If your configuration doesn't include hardware acceleration, then a built-in software encoder is invoked.

On Windows 64-bit operating systems, Wowza Transcoder AddOn can be configured to take advantage of the following hardware acceleration technologies:

- **Intel Quick Sync Video.** For recommended workstation and server-level hardware specifications, see [Server Specifications for Intel Quick Sync acceleration with Wowza Transcoder AddOn](#).
- **NVIDIA NVENC and NVIDIA CUDA.** For a list of supported NVIDIA graphics card that are compatible with Wowza Transcoder, see [Server Specifications for NVIDIA NVENC and NVIDIA CUDA acceleration with Wowza Transcoder AddOn](#).

To run Wowza Transcoder AddOn on 64-bit versions of the Windows Server operating system, the following server features are required:

- .NET Framework 3.5.1 or later
- Desktop Experience

On Linux 64-bit operating system distributions, Wowza Transcoder AddOn can be configured to take advantage of the following hardware acceleration technologies:

- **Intel Quick Sync Video.** For more information, see [How to configure Quick Sync accelerated encoding on Linux](#).
- **NVIDIA NVENC.** For more information, see [How to configure NVIDIA NVENC accelerated encoding on Linux](#).

Wowza nDVR AddOn

Wowza nDVR AddOn provides the ability to record a live stream into a cache on Wowza Streaming Engine. This enables viewers that join the live stream in-progress to access the cache to rewind to the beginning of the live stream (or rewind within the part of the stream that you specify) and then use DVR playback controls in their player to watch the stream from that point forward. Configuration for client playback of recorded streams is similar to playback of live streams from the Streaming Engine.

For more information about Wowza nDVR AddOn, see the [Wowza nDVR AddOn User's Guide](#) and the [Wowza nDVR Forum](#).

Wowza DRM AddOn

Wowza DRM AddOn provides integration with third-party Digital Rights Management (DRM) Key Management Service partners to enable on-the-fly encryption of premium live and VOD content for a variety of playback devices. For live workflows, per-stream encryption is available with the ability to rotate keys. For VOD workflows, per-asset and per-session encryption is available with the ability to rotate keys. Both live and VOD key rotation support is available for Apple HTTP Live Streaming (HLS).

Integration is supported for the following Key Management Service providers:

- **BuyDRM KeyOS.** Provides Microsoft PlayReady encryption services for MPEG-DASH, Apple HLS, and Microsoft Smooth Streaming and playback with BuyDRM players and Smooth Streaming clients on PCs, Macs, iOS devices, Android devices, Windows phones, game consoles, set-top boxes, and smart TVs.
- **EZDRM.** Provides Microsoft PlayReady encryption services for Smooth Streaming playback with Smooth Streaming clients on PCs, Macs, Windows phones, game consoles, set-top boxes, and smart TVs and with Discretix SecurePlayer media players on Android and iOS devices.
- **Verimatrix.** Provides Verimatrix VCAS and Microsoft PlayReady encryption services for HLS and Smooth Streaming playback with Verimatrix ViewRight and Smooth Streaming clients on PCs, Macs, iOS and Android devices, Windows phones, game consoles, set-top boxes, and smart TVs.

For more information about Wowza DRM AddOn, see the [Wowza DRM online tutorials](#) and the [Wowza DRM Forum](#).

Note

Wowza Streaming Engine has APIs that enable encryption schemes for on-the-fly encryption of live and VOD Apple HLS streams, including **SAMPLE-AES** (sample-level encryption for version 5 of the Apple HLS streaming protocol), **ENVELOPE-PLAYREADY** (supported by BuyDRM player technology with PlayReady DRM) and **CHUNK-PLAYREADY** (supported by INSIDE Secure player technology with PlayReady DRM). The Streaming Engine also has an API that enables on-the-fly encryption of live and VOD Microsoft Smooth Streaming format with PlayReady protection for INSIDE Secure player technology. Wowza DRM AddOn isn't required to use these APIs. For more information, see:

- [How to secure Apple HLS streaming using DRM encryption](#)
- [How to protect streams for delivery to INSIDE Secure player technology](#)

Downloadable AddOn Packages

Wowza® provides the following AddOn packages that you can download and install to extend and enhance Wowza Streaming Engine functionality.

AddOn Package	Description
GoCoder	The Wowza® GoCoder™ app is a live audio and video encoding application for Apple iPad (second generation and later), iPad Mini, iPhone (3GS and later), and iPod touch (fourth generation and later). It allows content providers to encode live content right from their iOS device and deliver it to Wowza Streaming Engine in real time over 4G, 3G, and Wi-Fi systems. For more information, see How to use Wowza GoCoder video broadcasting iOS app .
Wowza StreamLock	Wowza StreamLock™ AddOn is a security option for network encryption that provides near-instant provisioning of free 256-bit Secure Sockets Layer (SSL) certificates to verified Wowza users for use with Wowza Streaming Engine. StreamLock-provisioned SSL certificates provide the best security when used with RTMP. The certificates can also be used for secure HTTP streaming (HTTPS). The certificates expire after one year. StreamLock is only available for Subscription and Perpetual editions of Wowza Streaming Engine. For more information, see How to get SSL certificates from the StreamLock service .
Bandwidth Checker	This AddOn package enables server-to-client bandwidth measurement. For more information, see How to test server to client bandwidth for RTMP clients .
Central Configuration	The Central Configuration AddOn provides a system for managing multiple Wowza servers in a complex environment from a central location. The system can be modified to fit most CDN and service provider environments. For more information, see How to get Central Configuration AddOn (simplify multiple server deployments) .
Dynamic Load Balancing	This AddOn package enables RTMP streams to be dynamically distributed across multiple Wowza edge servers. The edge servers communicate with one or more load-balancing Wowza servers and clients connect to the load-balancing server to get

	the least-loaded edge server. For more information, see How to get the Dynamic Load Balancing AddOn .
GeoIP Locking	This AddOn package enables access to streamed content to be restricted based on a client's geographic location. For more information, see How to enable geographic locking .
Idle Client Disconnect	This AddOn package enables you to disconnect idle Flash RTMP clients automatically. This helps clear out connections to the server that are inactive. For more information, see How to disconnect idle Flash RTMP clients .
Silverlight Multicast Player	Wowza Streaming Engine contains a Microsoft Silverlight-based player that allows users to stream an MPEG-TS multicast from the Streaming Engine to any Silverlight-enabled desktop. The multicast feature allows users to deliver live video broadcasts across the network to thousands of Silverlight-based players simultaneously while only using the bandwidth of a single stream. For more information, see How to get the Silverlight Multicast Player AddOn .
Stream Name Aliasing	This AddOn package enables support for stream name aliases. It can be used to simplify complex URL-based stream names, provide security to limit the valid stream names used, or map one stream name to another. For more information, see How to get the StreamNameAlias AddOn .
SWF Hotlinking Protection	This AddOn package enables server-side hotlink-denial for SWF streams. For more information, see How to combat hotlinking your Adobe Flash SWF file .

Note

For an up-to-date list of the AddOn packages and information about how to use them, see the [AddOns webpage](#).

Installed Examples

The Wowza Streaming Engine software includes the following examples that highlight the server functionality. The examples are located in **[install-dir]/examples**. The **[install-dir]/examples/README.html** file describes the available examples and how to install them.

Example	Description
VideoOnDemandStreaming	This example illustrates how to configure and playback video on demand (VOD) content. It includes sample players for Apple iOS devices, Adobe Flash, Microsoft Silverlight, and DASH clients and source code for an OSMF-based Flash Player and Microsoft Silverlight 3 or later. It uses the default stream type.
LiveVideoStreaming	This example illustrates how to configure and playback live video. It includes sample players for iOS devices, Adobe Flash, Microsoft Silverlight, and DASH clients and source code for an OSMF-based Flash Player and Microsoft Silverlight 3 or later. It uses the live stream type.
LiveDVRStreaming	This example illustrates how to configure Wowza nDVR AddOn to record and playback a live video with DVR. It includes sample players for Adobe Flash and Microsoft Silverlight and source code for Microsoft Silverlight 3 or later. It uses the live stream type.
SHOUTcast	This Adobe Flash example illustrates how to re-stream SHOUTcast MP3 or AAC+ audio data through Wowza Streaming Engine. It uses the shoutcast stream type.
VideoChat	This Adobe Flash example illustrates how to implement video chat between two users. It uses the live-lowlatency stream type and the Camera and Microphone objects to get video and audio content. The example can stream video and audio data between two client connections or loop the data back to itself.
WebcamRecording	This Adobe Flash example illustrates how to implement Wowza Streaming Engine's advanced client-to-server video-recording capabilities using Adobe Flash Player. It uses the record stream type and the Camera and Microphone objects to get video and audio content. To use this example, you'll need a web camera (webcam) and Adobe Flash running in a web browser.
ServerSideModules	Developers can use this example with the Wowza Integrated Development Environment (IDE) to learn how to create custom server-side modules. The example contains server-side module class files and Flash client applications that demonstrate how Wowza Streaming Engine interacts with

	Flash clients. For more information about how to use this example with the Wowza IDE, see the Wowza IDE User's Guide .
--	--

Notes

- All Adobe Flash examples are implemented using ActionScript 3.0.
- Stream types are used to control the different types of streaming (live, VOD, chat, remote recording, and so on.) For more information, see [Stream Types](#).

Wowza Streaming Engine Editions

Wowza Streaming Engine software is available in Subscription or Perpetual editions to accommodate nearly any use case or business need. [See edition details](#).

Server Installation

How do I install Wowza Streaming Engine?

Wowza Streaming Engine™ is a small but powerful Java server. This chapter describes how to choose the correct version of Java and how to install and run Wowza Streaming Engine.

Before Installation

Wowza Streaming Engine is a Java 6 (aka 1.6) and Java 7 (aka 1.7) application and requires the installation of a Java Runtime Environment (JRE) that supports deploying Java in server environments. The server JRE has everything you need to run the Streaming Engine software on your system.

The following Java packages can be used with Wowza Streaming Engine:

- **Java Development Kit (JDK).** The JDK includes a complete JRE and enables Java developers to develop and debug server-side applications.
- **Java Server JRE.** The Server JRE is a complete JRE. Install this package if you want to run Java programs, but not develop them.
- **Java JRE.** The JRE also is a complete JRE; however, the server environment that's required to run Wowza Streaming Engine is removed from 32-bit installations of the JRE, starting in JRE 7 Update 21 (JRE 7u21). To run Wowza Streaming Engine on 32-bit platforms with a more recent Java JRE, you must use the Java Server JRE or Java JDK.

Wowza Streaming Engine is fully 64-bit compliant. It's architected to be highly multi-threaded and can take full advantage of multi-core hardware. To get the best performance, we recommend that you deploy Wowza Streaming Engine on a 64-bit operating system with

the latest 64-bit Java package (JDK, Server JRE, or JRE). Java packages can be downloaded from the [Java SE Downloads](#) webpage.

Notes

- Wowza Streaming Engine also includes Wowza Transcoder AddOn, which is only available for Windows or Linux when using a 64-bit operating system and 64-bit version of the Java VM.
- On the Windows platform, only the Server JRE and JDK include the **server** runtime environment; therefore, you should install the Server JRE or JDK when running on Windows. For more information about how to use the **server** version of the Java runtime environment that ships with Java JDK on Windows, see [Windows tuning, running the 'server' Java VM](#).

After your Java environment is installed and configured, you can validate that it's correct by opening a Command Prompt window (Windows) or terminal window (Mac OS X/Linux) and entering the command **java -version**. If correctly installed and configured, a version number that's equal to or greater than 1.6 is displayed.

Note

On the Windows platform, Wowza Streaming Engine uses the JAVA_HOME environment variable to determine the location of the Java environment under which it runs. If you have problems starting the Streaming Engine on Windows, make sure that the JAVA_HOME variable points to a valid Java environment. If you change or upgrade your Java environment, and the installation path is affected, be sure to update the JAVA_HOME variable to point to the new location. The JAVA_HOME variable should point to the base folder of the Java installation. This is the folder that contains the **bin** folder.

Installing Wowza Streaming Engine

On the Windows and Mac OS X platforms, the Wowza Streaming Engine software is installed using an installer. On Linux/Unix-based platforms, the software is installed using a self-extracting binary installer. The installers are available for download from the [Wowza Installers webpage](#).

During the installation process, you'll be asked to enter a valid product license key. You'll also be asked to create a user name and password for an Administrator account. You'll use this account to sign in to Wowza Streaming Engine Manager, a browser-based application that you can use to control the Streaming Engine software. The user name and password values are case-sensitive.

Notes

- If you're upgrading your Wowza server software to Wowza Streaming Engine 4.0.0, the previously installed version of the server software must be uninstalled. For more information about how to upgrade from an earlier version of the server software to Wowza Streaming Engine 4.0.0, see the [Wowza Streaming Engine Upgrade Guide](#).
- During the installation process, you can select options to start Wowza Streaming Engine and the browser-based Streaming Engine Manager automatically after the installation is finished. These options configure the server and manager software to start automatically as system services. If you don't choose one or both of these options, you must start the server software and/or the manager manually. See [Starting and Stopping Wowza Streaming Engine](#) and [Starting and Stopping Wowza Streaming Engine Manager](#).
- The directory that has the installed server application files, **bin**, **conf**, **content**, **examples**, **lib**, and **logs** folders, and other items is called **[install-dir]** in Wowza documentation.

Windows

To install the Wowza Streaming Engine software on Windows operating systems, double-click the installer file and follow the instructions on the screen. To find the installer file, press WIN key + F and search for **WowzaStreamingEngine-4.0.0**.

The default installation directory (**[install-dir]**) for the Streaming Engine software is:

```
/Program Files (x86)/Wowza Media Systems/Wowza Streaming Engine 4.0.0
```

To uninstall Wowza Streaming Engine, go to **Programs and Features** in Windows **Control Panel**, click **Wowza Streaming Engine 4.0.0**, and then click **Uninstall**.

Mac OS X

To install the Wowza Streaming Engine software on Mac OS X, mount the disk image (open the **WowzaStreamingEngine-4.0.0.dmg** file), double-click the installer package (**Wowza Streaming Engine 4.0.0.pkg**) file, and then follow the instructions on the screen. Files will be installed to the following locations:

```
/Applications/Wowza Streaming Engine 4.0.0
- Readme file and symlinks to server sample content, examples,
  documentation, and other items

/Library/LaunchDaemons
- background service scripts com.wowza.WowzaStreamingEngine.plist
  and com.wowza.WowzaStreamingEngineManager.plist

/Library/WowzaStreamingEngine
- symlink
```

```
/Library/WowzaStreamingEngine-4.0.0
- server application files and applications, bin, conf, content,
  examples, lib, and logs folders
```

To uninstall, move the following items to the Trash.

```
folders:
  /Applications/Wowza Streaming Engine 4.0.0
  /Library/WowzaStreamingEngine-4.0.0

files:
  /Library/LaunchDaemons/com.wowza.WowzaStreamingEngine.plist
  /Library/LaunchDaemons/com.wowza.WowzaStreamingEngineManager.plist

symlink:
  /Library/WowzaStreamingEngine
```

Note

To ensure that Wowza Streaming Engine folders are completely removed from the library, on the **Go** menu, click **Go to Folder**, type **/Library**, and then click **Go**. In the Library window, move the **WowzaStreamingEngine** and **WowzaStreamingEngine-4.0.0** folders to the Trash.

Linux

Note

The Linux package manager will extract the Wowza Streaming Engine software files to the **/usr/local/WowzaStreamingEngine-4.0.0** directory and the server will be installed as the **root** user.

Red Hat Package Manager Systems

Install

```
sudo chmod +x WowzaStreamingEngine-4.0.0.rpm.bin
sudo ./WowzaStreamingEngine-4.0.0.rpm.bin
```

Uninstall

```
sudo rpm -e WowzaStreamingEngine-4.0.0
```

Debian Package Manager Systems

Install

```
sudo chmod +x WowzaStreamingEngine-4.0.0.deb.bin
sudo ./WowzaStreamingEngine-4.0.0.deb.bin
```

Uninstall

```
sudo dpkg --purge WowzaStreamingEngine-4.0.0
```

Other Linux and Unix Systems

Install

To install the Wowza Streaming Engine software on other Linux-based and Unix-based systems, download **WowzaStreamingEngine-4.0.0.tar.bin** to any directory and then execute the self-extracting installer:

```
sudo chmod +x WowzaStreamingEngine-4.0.0.tar.bin
sudo ./WowzaStreamingEngine-4.0.0.tar.bin
```

Uninstall

```
cd /usr/local
sudo rm -f WowzaStreamingEngine
sudo rm -rf WowzaStreamingEngine-4.0.0
```

Starting and Stopping Wowza Streaming Engine

Notes

- The Wowza Streaming Engine must be started before you can start and use Wowza Streaming Engine Manager. See [Starting and Stopping Wowza Streaming Engine Manager](#).
- Wowza Streaming Engine can't run as a service and in standalone mode at the same time.

Windows

Service

To start the Wowza Streaming Engine service:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).
2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine 4.0.0**, and then click **Start**.

To stop the service:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).

2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine 4.0.0**, and then click **Stop**.

Wowza Streaming Engine can be set to start automatically as a Windows service when Windows starts. To prevent the service from starting automatically when Windows starts:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).
2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine 4.0.0**, and then click **Properties**.
3. In the **Properties** dialog box, on the **General** tab, set **Startup type** to **Manual**.

Standalone

To start Wowza Streaming Engine in standalone mode, make sure that the Wowza Streaming Engine service is stopped (see above), and then do the following:

1. Open a Command Prompt window (press WIN key + R, type **cmd** in the **Run** dialog box, and then press ENTER).
2. Execute the following commands:

```
cd %WMSAPP_HOME%\bin
startup.bat
```

To stop the server:

1. Open a Command Prompt window (press WIN key + R, type **cmd** in the **Run** dialog box, and then press ENTER).
2. Execute the following commands:

```
cd %WMSAPP_HOME%\bin
shutdown.bat
```

Notes

- The **Wowza Streaming Engine 4.0.0** service runs under the **Local System account** by default. This can limit how the Streaming Engine software interacts with the underlying operating system. For example, you may have issues streaming content from UNC paths. To address this issue, update the service in the Services MMC snap-in to run as a named user. To do this, right click the service name in the Services MMC snap-in, click **Properties**, and then on the **Log On** tab specify an alternate user account that the service can use to log on under **This account**.
- The hardware acceleration (NVIDIA CUDA and Intel Quick Sync) used by Wowza Transcoder is only available when running Wowza Streaming Engine as a Windows standalone application. The hardware acceleration isn't available when the Streaming Engine is invoked as a service.

Mac OS X

Service

To start Wowza Streaming Engine as a Mac OS X launchd service, open a terminal window and enter the following command:

```
sudo launchctl load -w  
/Library/LaunchDaemons/com.wowza.WowzaStreamingEngine.plist
```

To stop the service, enter:

```
sudo launchctl unload -w  
/Library/LaunchDaemons/com.wowza.WowzaStreamingEngine.plist
```

Standalone

To start Wowza Streaming Engine in standalone mode, invoke the **Wowza Streaming Engine Startup** script in **/Library/WowzaStreamingEngine-4.0.0/bin** or open a terminal window and enter the following commands:

```
cd /Library/WowzaStreamingEngine-4.0.0/bin  
./startup.sh
```

To stop the server, invoke the **Wowza Streaming Engine Shutdown** script in **/Library/WowzaStreamingEngine-4.0.0/bin** or open a terminal window and enter:

```
cd /Library/WowzaStreamingEngine-4.0.0/bin  
./shutdown.sh
```

Note

Invoking the **Wowza Streaming Engine Startup** and **Wowza Streaming Engine Shutdown** scripts also starts and stops Wowza Streaming Engine Manager. See [Starting and Stopping Wowza Streaming Engine Manager](#).

Linux

Note

The operations in this section must be performed as the **root** user with **sudo** access.

Service

To start Wowza Streaming Engine as a Linux service, open a terminal window and enter one of the following commands (the commands differ based on your Linux distribution):

```
sudo service WowzaStreamingEngine start
```


-or-

```
/etc/init.d/WowzaStreamingEngine start
```

To stop the service, enter:

```
sudo service WowzaStreamingEngine stop
```

-or-

```
/etc/init.d/WowzaStreamingEngine stop
```

Notes

- The method of running init.d-based services may be different on different Linux distributions. If these instructions don't apply to your Linux distribution, consult your Linux manual.
- The Linux services script subsystem doesn't use the full \$PATH definition to determine the location of Linux commands. It uses what's known as the **init** path. This can lead to an issue on Linux distributions where the default installation location for Java can't be found by applying the **init** path. For more information, see [After installing latest Java version, java command is still using old Java \(fix\)](#).

Standalone

To start the server in standalone mode, open a terminal window and enter the following commands:

```
cd /usr/local/WowzaStreamingEngine/bin  
./startup.sh
```

To stop the server, enter:

```
cd /usr/local/WowzaStreamingEngine/bin  
./shutdown.sh
```

Running Wowza Streaming Engine as a Named User

On Mac OS X and Linux platforms, the default installation of Wowza Streaming Engine runs the server as the **root** user. If you want to run the server as a different user, follow the instructions in [Run Wowza Streaming Engine as Named User](#) to create a new user and configure the server to run as that user.

Note

For security reasons, the **root** user can't bind to port numbers greater than 1024 on most Linux and Unix distributions. If you plan on running Wowza Streaming Engine on a lower-numbered ports such as 80 (HTTP), 443 (HTTPS, RTMPS), or 554 (RTSP), the server must continue to run as the **root** user.

Running Multiple Wowza Streaming Engine Instances

You can run multiple instances of the same version of Wowza Streaming Engine on one computer instead of running one large instance with multiple virtual hosts (VHosts). An *instance* is defined as a single copy of the running Streaming Engine software. This is useful in cases where there are additional resources available on the computer that can't be used by a single instance, your streaming workflow requires a custom design that won't work using a single instance, or you're testing a multiple-server deployment on a single computer. Each instance requires a license, either a Subscription license or a Perpetual license. You can use a Subscription license key for multiple instances and each instance will be counted separately and reflected in your bill. For more information, see [How to run multiple instances of Wowza Streaming Engine on one computer](#).

Entering a New License Key

License keys for all Wowza® products, including Wowza Streaming Engine and AddOns, are stored in **[install-dir]/conf/Server.license**.

%WMSCONFIG_HOME%\conf\Server.license	- Windows
/Library/WowzaStreamingEngine/conf/Server.license	- Mac OS X
/usr/local/WowzaStreamingEngine/conf/Server.license	- Linux/Unix

Subscription users can run an unlimited number of server instances and AddOns under a single license key. Perpetual Edition users must enter a separate license key for each server instance and for each AddOn.

To add a license key in Wowza Streaming Engine Manager, do the following:

1. Click the **Server** tab and then click **Server Setup** in the contents pane.
2. In the **Server Setup** page, click **Edit**.
3. Enter each new license key on a new line in the **License Keys** box, and then click **Save**.
4. Click the **Restart Now** button at the top of the **Server Setup** page.

When the standalone server is restarted, the new license(s) will be in effect. The licenses are additive, so when adding additional licenses, be sure to retain the original license information in the **License Keys** box and add each new license key on its own new line. The order in which the keys are listed isn't important. The first and last five digits of the license key are displayed in the **License Keys** box

License Keys

```
ENGP4-XXXXX-XXXXX-XXXXX-XXXXX-h5t3C
TRN14-XXXXX-XXXXX-XXXXX-XXXXX-N6fwa
DVRA4-XXXXX-XXXXX-XXXXX-XXXXX-ahFdF
```

Note

You can also open the **Server.license** file in a text editor, enter each new license key on a new line, and then restart the server. The following example shows the **Server.license** file contents for a Perpetual edition user. The list has a Wowza Streaming Engine 4.0 license key and license keys for the Wowza Transcoder AddOn and Wowza nDVR AddOn.

```
ENGP4-LaGpC-ZrTD9-F4Y3S-a9bR2-h5t3C
TRN14-y9Gj2-kneqT-2zjHp-GadzB-N6fwa
DVRA4-k3r3R-nzxCB-ypjs5-Sk3y9-ahFdF
```

Ports Used for Streaming

Before streaming with Wowza Streaming Engine, you should open ports on your firewall. The following table shows the default ports that the Streaming Engine uses for streaming. All of these port numbers are configurable through the configuration files that are described later in this document.

TCP 1935	RTMP/RTMPE/RTMPT/RTSP-interleaved streaming/WOWZ™
TCP 8086-8088	Administration
UDP 6970-9999	RTP UDP streaming

By default, Wowza Streaming Engine is configured to use only TCP port 1935 for streaming. You may want to configure additional ports for streaming such as TCP port 80 for HTTP or RTMPT streaming or TCP port 554 for RTSP streaming. To add an additional port, go to the **Virtual Host Setup** page in Wowza Streaming Engine Manager and edit the **Default Streaming** host port.

Host Ports

Name	Type	IP Address	Port	SSL/StreamLock Enabled	Actions
Default Streaming	Streaming	*	1935	false	 
Default Admin	Admin	*	8086	false	 

In the **Edit host port** dialog box, add the additional ports to the **Port(s)** list (this list is comma-delimited).

Edit host port
X

Name
Default Streaming

Type
Streaming

IP Address *

Port(s) *

Comma-separated list

☐ Enable SSL/StreamLock

Keystore Path (StreamLock certificate path)

Keystore Password (StreamLock certificate password)

Wowza Streaming Engine can't share ports with other programs or services, so make sure that there are no other programs or services running that share the added ports. The following table shows some of the common ports used for streaming.

TCP 80	Adobe HDS, Apple HLS, Microsoft Smooth Streaming, MPEG-DASH streaming, RTMPT
TCP 443	RTMPS, HTTPS
TCP 554	RTSP

Server Configuration and Tuning

Wowza Streaming Engine configuration settings are stored in a set of XML configuration and properties files in the **[install-dir]/conf** folder. The settings can be changed by configuring options and properties in Wowza Streaming Engine Manager or by editing them in a text editor. If you choose to manage the Streaming Engine configuration settings by editing the XML files directly, be sure to review the [Wowza Streaming Engine Configuration Reference](#), which describes the most commonly used configuration settings.

The following configuration files are read when the server starts:

Server Configuration Files

MediaCache.xml	- Media Cache configuration
Server.xml	- General server configuration
Tune.xml	- Server performance tuning configuration
VHosts.xml	- Virtual hosts definition
log4j.properties	- Logging configuration

VHost Configuration Files

StartupStreams.xml	- Streams started at virtual host startup
VHost.xml	- Virtual host configuration
VHosts.xml	- Virtual hosts configuration

Application Configuration Files

Application.xml	- Application configuration
-----------------	-----------------------------

Wowza Streaming Engine is automatically tuned to take best advantage of available hardware resources when the server starts. The server calculates an appropriate Java heap size, garbage collection (GC) settings, and other Java command-line options based on available hardware, the computer and Java Virtual Machine (JVM) bitness, and other factors.

By default, the server sets the Java heap size to a value that's suitable for application development environments. Before you deploy the server in production environments where it may use memory extensively when heavily loaded, you can select an option in Wowza Streaming Engine Manager that automatically sets the heap size to a predefined value that's appropriate for production use. You can also adjust many other performance settings from the default values that are calculated by the server in Streaming Engine Manager to fine-tune the server's performance. For more information, see [Performance Tuning](#).

Note

Wowza provides a Flash RTMP Load Test Tool that can be used to generate RTMP load on a single Wowza Streaming Engine instance to test configuration and performance. The Load Test Tool requires a Subscription or Perpetual license for the Wowza Streaming Engine software. For more information, see [How to get Flash RTMP Load Test Tool](#).

Software Updates

In between production releases, development builds are produced periodically in the form of updates. This allows users to get early access to new features in the latest Wowza Streaming Engine software releases and to give feedback. Information about what's included in each update is included in a README.txt file that's included in the update archive (.zip) file. For more information about how to apply an update to your server software, see [How to apply a Wowza Streaming Engine update](#).

Application Configuration

How do I create and configure an application for streaming?

Wowza Streaming Engine™ software is designed to handle multiple streaming protocols. All streaming is controlled through the creation and configuration of streaming applications. One application can be configured to simultaneously deliver either live or video on demand (VOD) content to multiple player technologies. It's easy to define an application in Wowza Streaming Engine Manager. For example, to create a new application named **myapplication**, do the following:

1. Click the **Applications** tab in Streaming Engine Manager and then click **Add Application** in the contents pane.
2. On the **Add Application** page, review the content in the Help pane to decide what type of application you want to create.
3. Click the application type in the **Add Application** page.
4. In the **New Application** dialog box, enter the name **myapplication** and then click **Add**.
5. The **myapplication** page is displayed so that you can configure the application settings.

A single application can be configured to deliver a live or video on demand (VOD) stream at the same time to Adobe Flash Player, Apple iOS devices (iPhone, iPad, or iPod touch) or Apple TV digital media extender, Roku and Amino set-top boxes, Microsoft Silverlight, DASH clients, and RTSP/RTP-based players (including 3GPP smartphones and tablets, and Android devices). The [Quick Start Guide](#) contains basic tutorials with step-by-step instructions that describe how to configure applications for common streaming tasks. The remainder of this chapter covers application configuration details. For more detailed configuration information, see the [Wowza Streaming Engine Configuration Reference](#).

Applications and Application Instances

(Application.xml)

An **Application.xml** file defines the configuration that you set up in Wowza Streaming Engine Manager for a given application. An application instance is an instantiation of an application and provides a namespace and context for streaming. An application instance is started dynamically and a single application can have multiple named application instances running simultaneously. If no name is specified for an application instance, then the default name (**_definst_**) is used. In many streaming scenarios, a single application instance is used per-application and the name is never referenced and defaults to **_definst_**. It's more common to use multiple application instances in video chat and video conferencing scenarios where you must create multiple rooms for streaming. In this case, application instances are used to separate streaming into rooms. Each room is a separate application instance, which provides separation and a namespace for each room.

When an application instance is loaded, it looks in the following locations for an **Application.xml** file (where **[application]** is the application name):

```
[install-dir]/conf/[application]/Application.xml
[install-dir]/conf/Application.xml
```

The first **Application.xml** file that's found is used.

URL Formats

All streaming in Wowza Streaming Engine is initiated with a Uniform Resource Locator (URL). The application and application instance names are specified as part of the streaming URL. The URL formats used for streaming, whether for Adobe Flash Player, Apple iOS devices, Microsoft Silverlight, DASH clients, or RTSP/RTP, follow a similar format:

```
[protocol]://[address]:[port]/[application]/[appInstance]/[streamName]/[post-fix]
```

-where-

```
[protocol]:      - streaming protocol (http, rtmp, rtsp, and so on)
[address]:      - address of the server running Wowza Streaming Engine
[port]:         - port number to use for streaming (1935 is the default)
[application]   - application name
[appInstance]   - application instance name
[streamName]    - stream name and prefix
[post-fix]      - option information specific to player technology
```


In most streaming scenarios, if [streamName] doesn't have path elements and the default [appInstance] name is used, the URL can be shortened to:

```
[protocol]://[address]:[port]/[application]/[streamName]
```

The following are example URLs for different player technologies. The examples assume that a live video with the stream name **myStream** using the application name **live** is streamed.

Adobe HDS

```
http://mycompany.com:1935/live/myStream/manifest.f4m
```

Apple HLS

```
http://mycompany.com:1935/live/myStream/playlist.m3u8
```

Microsoft Smooth Streaming

```
http://mycompany.com:1935/live/myStream/Manifest
```

MPEG-DASH Streaming

```
http://mycompany.com:1935/live/myStream/manifest.mpd
```

Adobe RTMP

```
Server: rtmp://mycompany.com/live  
Stream: myStream
```

RTSP/RTP

```
rtsp://mycompany.com:1935/live/myStream
```

Now is probably a good time to take a quick look at the default settings for applications. The rest of this chapter describes the most commonly configured items.

Stream Types

Wowza Streaming Engine uses named stream types to control the different types of streaming (live, VOD, chat, remote recording, and so on.). Stream types are automatically configured when you create different application types and configure their options in Wowza Streaming Engine Manager. You can also edit the **Streams/StreamType** property in **Application.xml** using a text editor to change the stream type for an application. The following table shows the stream types and their uses.

Stream type	Description
default	VOD
file	VOD
live	Publish and play live content (best for one-to-many streaming of live events)
live-lowlatency	Publish and play live content (best for one-to-one or one-to-few video/audio chat applications)
live-record	Same as live —in addition content is recorded
live-record-lowlatency	Same as live-lowlatency —in addition content is recorded
liverepeater-edge	Publish and play live content across multiple Wowza servers in an origin/edge configuration (used to configure edge application)
liverepeater-edge-lowlatency	Publish and play live content across multiple Wowza servers in an origin/edge configuration (used to configure edge application when latency is important)
liverepeater-edge-origin	Publish and play live content across multiple Wowza servers in an origin/edge/edge configuration (used to configure a middle-edge application)
liverepeater-origin	Publish and play live content across multiple Wowza servers in an origin/edge configuration (used to configure origin application)
liverepeater-origin-record	Same as liverepeater-origin —in addition content is recorded
record	Video recording
rtp-live	Re-stream RTSP/RTP, native RTP, or MPEG-TS streams
rtp-live-lowlatency	Re-stream RTSP/RTP, native RTP, or MPEG-TS streams when latency is important
rtp-live-record	Same as rtp-live —in addition content is recorded
rtp-live-record-lowlatency	Same as rtp-live-lowlatency —in addition content is recorded

shoutcast	Re-stream SHOUTcast/Icecast MP3 or AAC+ audio streams
shoutcast-record	Same as shoutcast —in addition content is recorded

Each stream type exposes properties that are used for tuning the stream type. For example, the stream type definitions for **live** and **live-lowlatency** differ only in the tuning that's accomplished through the stream properties. Defined properties for a stream type can be overridden on a per-application basis by defining new property values on an application's **Properties** tab in the Streaming Engine manager or by editing the **Streams/Properties** container in **Application.xml**.

HTTP Streamers and Live Stream Packetizers

HTTP streamers define the streams in an application (live or VOD) that are available for playback to different player technologies. In Wowza Streaming Engine Manager, you can select one or more of the following **Playback Types** options for an application. When selecting multiple options, the corresponding HTTP streamers are added to the **<HTTPStreamers>** section in **Application.xml** as a comma-separated list.

Playback Type	Description
Adobe HDS	Enables the application to stream live and VOD content to Flash Player using the Adobe HTTP Dynamic Streaming (HDS) protocol. It adds the sanjosestreaming HTTP streamer to the <HTTPStreamers> section in Application.xml .
Apple HLS	Enables the application to stream live and VOD content to iOS-based devices (iPhone/iPad/iPod touch iOS version 3.0 or later), QuickTime player (version 10 or later), Safari browser (version 4.0 or later), and to other devices such as Roku and Amino set-top boxes and some brands of smart TVs, using the Apple HTTP Live Streaming (HLS) protocol. It adds the cupertinostreaming HTTP streamer to the <HTTPStreamers> section in Application.xml .
Microsoft Smooth Streaming	Enables the application to stream live and VOD content to Microsoft Silverlight using the Microsoft Smooth Streaming protocol. It adds the smoothstreaming HTTP streamer to the <HTTPStreamers> section in

	Application.xml.
MPEG-Dash	Enables the application to stream live and VOD content to DASH clients using the Dynamic Adaptive Streaming over HTTP (DASH) protocol. It adds the mpegdashstreaming HTTP streamer to the <HTTPStreamers> section in Application.xml .
nDVR AddOn (live streaming only)	When you enable the nDVR AddOn feature for live or live http origin applications in the Streaming Engine manager, it enables the application to stream live content from Wowza Streaming Engine (origin) to Wowza Streaming Engine (edge). It adds the dvrchunkstreaming HTTP streamer to the <HTTPStreamers> section in Application.xml .

Live streams coming into Wowza Streaming Engine must be packaged (*packetized*) before they can be delivered to clients using HTTP streaming protocols. The **<Streams>/<LiveStreamPacketizers>** section in **Application.xml** specifies the streaming protocols that are used when packetizing live streams. There are two types of packetizers: streaming packetizers and repeater packetizers.

Streaming packetizers are used when delivering a live stream from a single Wowza server to clients. They're also used when delivering a live stream from an origin Wowza server to an edge Wowza server when using the live repeater mechanism in an origin/edge configuration. When you select **Playback Types** options in Streaming Engine Manager to create HTTP streamers for live applications, the corresponding live stream packetizer values (separated by commas) are added to the **<LiveStreamPacketizers>** section in **Application.xml**.

Playback Type	Description
Adobe HDS	Enables Adobe HDS live streaming to Flash Player. It adds the sanjosestreamingpacketizer streaming packetizer to the <LiveStreamPacketizers> section in Application.xml .
Apple HLS	Enables Apple HLS live streaming to iOS-based devices. It adds the cupertinostreamingpacketizer streaming packetizer to the <LiveStreamPacketizers> section in Application.xml .
Microsoft Smooth Streaming	Enables Microsoft Smooth Streaming to Silverlight. It adds the smoothstreaming streaming packetizer to the

	<LiveStreamPacketizers> section in Application.xml .
MPEG-Dash	Enables MPEG-DASH streaming to DASH clients. It adds the mpegdashstreamingpacketizer streaming packetizer to the <LiveStreamPacketizers> section in Application.xml .
nDVR AddOn	When you enable the nDVR AddOn feature for live or live http origin applications in the Streaming Engine manager, it adds the dvstreamingpacketizer streaming packetizer to the <LiveStreamPacketizers> section in Application.xml for use with Wowza nDVR AddOn.

Repeater packetizers are used when delivering a live stream from a Wowza edge server to clients in a live stream repeater (origin/edge) configuration. When you select **Playback Types** options in Streaming Engine Manager to create HTTP streamers for live edge applications, the corresponding repeater packetizer values (separated by commas) are added to the **<LiveStreamPacketizers>** section in **Application.xml**.

Playback Type	Description
Adobe HDS	Enables Adobe HDS live stream repeater for Flash Player. It adds the sanjosestreamingrepeater repeater packetizer to the <LiveStreamPacketizers> section in Application.xml .
Apple HLS	Enables Apple HLS live stream repeater for iOS-based devices. It adds the cupertinostreamingrepeater repeater packetizer to the <LiveStreamPacketizers> section in Application.xml .
Microsoft Smooth Streaming	Enables Microsoft Smooth Streaming live stream repeater for Silverlight. It adds the smoothstreamingrepeater repeater packetizer to the <LiveStreamPacketizers> section in Application.xml .
MPEG-Dash	Enables MPEG-DASH live stream repeater for DASH clients. It adds the mpegdashstreamingrepeater repeater packetizer to the <LiveStreamPacketizers> section in Application.xml .
nDVR AddOn	When you enable the nDVR AddOn feature for live edge applications in the Streaming Engine manager, it adds the

	dvrstreamingrepeater repeater packetizer to the <LiveStreamPacketizers> section in Application.xml for use with Wowza nDVR AddOn.
--	--

For more information about how to implement the live stream repeater (origin/edge) mechanism for delivering a live media event across multiple Wowza servers, see [Live Stream Repeater \(Origin/Edge Live Streaming\)](#).

Note

Wowza nDVR AddOn provides the ability to record a live stream while simultaneously allowing users to play or pause the live stream, rewind to a previously recorded point, or resume viewing at the live point. This capability can be extended to an edge Wowza server in an origin/edge configuration. For more information, see the [Wowza nDVR AddOn User's Guide](#).

Timed Text Providers

Wowza Streaming Engine includes support for timed text (closed captioning) for live and video on-demand streams. The Streaming Engine enables caption data from a variety of instream and file-based sources to be converted to appropriate caption formats for live and on-demand video streaming using the Adobe HDS, Apple HLS, and RTMP streaming protocols. This feature helps US broadcasters to comply with the Twenty-First Century Communications and Video Accessibility Act of 2010 and increasing requirements in the European Union by providing captioning for television programs that are distributed over the Internet.

Closed Captioning for Live Streams

For live streams, Wowza Streaming Engine can ingest instream closed caption information from CEA-608 data or AMF **onTextData** events. These ingested captions can be delivered as CEA-608-formatted SEI data in Apple HLS streams or as **onTextData** events in Adobe HDS and Adobe RTMP streams. In addition, instream CEA-608 caption data can be passed through Wowza Transcoder AddOn for delivery in Apple HLS streams. For live and live edge applications in Streaming Engine Manager, you can configure the following **Closed Caption Sources** options to enable the application to ingest the caption data

Live Closed Caption Source	Description
onTextData events in live streams	This option enables the application to monitor live streams for Action Message Format (AMF) onTextData captions, decode the captions, and convert them to CEA-608-formatted SEI data in Apple HLS streams. It adds the ModuleOnTextDataToCEA608 module to the <Modules> section in Application.xml .
Embedded CEA-608 captions in live streams	This option enables the application to monitor live streams for CEA-608 captions, decode the captions, and convert them to onTextData events in Adobe HDS and Adobe RTMP streams. It adds the ModuleCEA608ToOnTextData module to the <Modules> section in Application.xml .
onCaptionInfo events in live streams	Specified by the onCaptionInfo events in live streams option. This option enables the application to monitor live streams for AMF onTextData events and pass them through in Adobe HDS and Adobe RTMP streams. It adds the captionLiveIngest property to the <TimedText>/<Properties> section in Application.xml .

For more information, see [How to configure closed captioning for live streaming](#).

Closed Captioning for Video on Demand Streams

For video on-demand streams, Wowza Streaming Engine can extract caption data from 3GPP Timed Text data embedded in MP4 files or use caption files in a variety of formats including Timed Text Markup Language (.ttml), SubRip Text (.srt), Scenarist Closed Caption (.scc), and Web Video Text Tracks (.vtt). The ingested captions can be delivered as CEA-608-formatted SEI data in Apple HLS streams or as Action Message Format (AMF) **onTextData** events in Adobe HDS and Adobe RTMP streams. For VOD and VOD edge applications in Streaming Engine Manager, you can select one or more of the following **Closed Caption Sources** options to ingest caption data. When selecting multiple options, the corresponding timed text providers are added to the **<TimedText>** section in **Application.xml** as a comma-separated list.

VOD Closed Caption Source	Description
Embedded 3GPP/MPEG-4 Timed Text tracks	This option enables the application to pull captions directly from 3GPP tracks (codecID "tx3g") that are embedded in MP4 VOD assets. This option is enabled by default. It adds the vodcaptionprovidermp4_3gpp timed text provider to the <TimedText> section in Application.xml .
Timed Text (TTML/DXFP) file	This option enables the application to pull captions from an external TTML-formatted caption file that sits next to the VOD asset in the application's content directory. It adds the vodcaptionproviderttml timed text provider to the <TimedText> section in Application.xml .
SubRip (SRT) file	This option enables the application to pull captions from an external SRT-formatted caption file that sits next to the VOD asset in the application's content directory. It adds the vodcaptionprovidersrt timed text provider to the <TimedText> section in Application.xml .
Web Video Text Track (WebVTT) file	This option enables the application to pull captions from an external WebVTT-formatted caption file that sits next to VOD asset in the application's content directory. It adds the vodcaptionproviderwebvtt timed text provider to the <TimedText> section in Application.xml .
Scenarist Closed Caption (SCC) file	This option enables the application to pull captions from an external SCC-formatted caption file that sits next to VOD asset in the application's content directory. It adds the vodcaptionproviderscc timed text provider to the <TimedText> section in Application.xml .

For more information, see [How to configure closed captioning for video on demand streaming](#).

Wowza Transcoder AddOn and Wowza nDVR AddOn Configurations

The <Transcoder> and <DVR> containers in an **Application.xml** file serve to configure an applications to use Wowza Transcoder AddOn and Wowza nDVR AddOn respectively. For more information, see the [Wowza Streaming Engine Configuration Reference](#) and the following tutorials:

- [How to set up and run Wowza Transcoder AddOn for live streaming](#)
- [How to set-up and run Wowza nDVR for live streaming](#)

Modules

Modules are Java classes that are loaded dynamically when an application instance is loaded and provide an application's functionality. In Wowza Streaming Engine Manager, the **Modules** list defines an order-dependent list of modules to be loaded for a given application. Many AddOn packages provide additional functionality through the use of modules. For more information, see [Server-side Modules](#).

In the manager, click the **Modules** tab on an application page to see the list of modules that are loaded.

[Setup](#)
[Properties](#)
[Modules](#)

Note: Items on this page should be configured by advanced users only.

Modules Java classes that extend an application's functionality. The list below defines an order-dependent list of modules to be loaded for a given application. The modules are loaded dynamically when the application instance is loaded. The base (ModuleCore) module must be included by the application for it to operate properly.

Edit

Name	Description	Fully Qualified Class Name
base	Base	com.wowza.wms.module.ModuleCore
logging	Client Logging	com.wowza.wms.module.ModuleClientLogging
flvplayback	FLVPlayback	com.wowza.wms.module.ModuleFLVPlayback
ModuleCoreSecurity	Core Security Module for Applications	com.wowza.wms.security.ModuleCoreSecurity

Each module must have a unique **Name**. The **Description** information is for providing a detailed description of the module and isn't used in any operations. The **Class** item is the fully qualified path to the Java class that provides the module's functionality. In general, new modules are always added to the end of the **Modules** list. The Java class that makes up a

server-side module is most often bound to a **.jar** file in the Streaming Engine installation. The Wowza Streaming Engine software comes with many modules that can be added to the **Modules** list to provide additional functionality. For a complete list of the modules, see [Built-in Server Modules](#). You can also use the free [Wowza Integrated Development Environment \(Wowza IDE\)](#) tool to develop your own custom modules to provide additional functionality. For more information, see [Extending Wowza Streaming Engine Using Java](#).

Notes

- Access to the **Modules** tab is limited to administrators with advanced permissions. For more information, see [Managing Sign-In Credentials](#).
- Wowza provides a collection of utility modules that are ready to use in Wowza applications. These modules don't require you to use the Wowza IDE. For more information, see [Module Collection](#).

Properties

Properties are a list of name/value pairs that provide a means for tuning and modifying the default application configuration. Properties can also be used server-side as a means to pass data to custom modules from applications. In the **Application.xml** configuration file, a property definition has the following form:

```
<Property>
  <Name>[name]</Name>
  <Value>[value]</Value>
  <Type>[type]</Type>
</Property>
```

Where **<Name>** is the property name, **<Value>** is the property value, and **<Type>** is the property type. Valid property types are: **Boolean**, **Integer**, **Long**, and **String**.

In Wowza Streaming Engine Manager, you can click the **Properties** tab on an application page and enable default properties to either add them to the application configuration or to override existing property values. For details about the properties, see the [Wowza Streaming Engine Configuration Reference](#).

Many articles on the Wowza website prescribe custom properties for tuning the server and for adding advanced functionality. When adding custom properties, it's important to add them to the correct **<Properties>** container in **Application.xml**. The article instructions always specify the **Path** value to use in the **Add Custom Property** dialog box, which adds the property to the correct **<Properties>** container.

Add Custom Property

X

Path

Name *

Type

Value *

Cancel

+ Add

Note

Access to the **Properties** tab is limited to administrators with advanced permissions. For more information, see [Managing Sign-In Credentials](#).

Media Types

Media types aren't defined in application configuration files but are an important part of streaming. Wowza Streaming Engine supports many media types. It can read the following media or file types:

- **MP4** (QuickTime container - .mp4, .f4v, .mov, .m4a, .m4v, .mp4a, .mp4v, .3gp, .3g2, etc.)
- **FLV** (Flash Video - .flv)
- **MP3** content (.mp3)
- **SMIL** (Synchronized Multimedia Integration Language - .smil)
- **AMLST** (API-based MediaList)

Media types are specified by appending a prefix to the stream name. For example to play the MP4 file **mycoolvideo.mov**, use the stream name **mp4:mycoolvideo.mov**, where **mp4:** is the media type prefix. If no media type prefix is specified, the media type prefix defaults to **mp4:**. The following table shows the supported media type prefixes.

Media type prefix	Description
mp4:	QuickTime container (default if no prefix specified)
flv:	Flash Video
mp3:	MP3 file
id3:	MP3 file (returns only ID3 tag information)
smil:	Synchronized Multimedia Integration Language (for adaptive bitrate delivery)
ngrp:	Named Group (for adaptive bitrate delivery)
amlst:	API-based MediaList (for adaptive bitrate delivery)

The media type prefix is also used to control the file container that stores recorded live video. When publishing video, if the **mp4:** media type prefix is specified or if no prefix is specified, then the content is recorded to an **MP4** (QuickTime) container. Only H.264, AAC, and MP3 content can be recorded to an **MP4** container. If the **flv:** media type prefix is specified, an **FLV** (Flash Video) container is used.

Synchronized Multimedia Integration Language (**.smil**) files provide a means to specify a group of live streams or VOD files for adaptive bitrate switching. For stream switching to occur correctly, key frames must be properly aligned across all of the available bitrates in a live stream. For VOD, multiple files must be pre-encoded to the desired bitrates and have key frames that are aligned across all of the encoded files. The **smil:** media type prefix is used to playback the content that's specified in **.smil** files.

Wowza Transcoder AddOn uses a templating system to group streams into logical groups (called *Stream Name Groups*) for live adaptive bitrate delivery. Stream Name Groups and SMIL files serve the same purpose and either method can be used for playback of live streams. Stream Name Groups are defined in the transcoder template and are available for playback using the **ngrp:** media type prefix.

Wowza Streaming Engine has an API that can be used to intercept requests for adaptive bitrate streams and provide the stream grouping through Java API calls. To use this feature, you must use the **amlst:** stream name prefix to use a set of Java objects that describe the

adaptive bitrate stream (an *API-based MediaList*). When the Streaming Engine reads a SMIL file, it creates a MediaList and passes it back to the streaming provider. This API provides a means for intercepting the requests and creating the MediaList dynamically in a Streaming Engine module. For more information, see [How to use Java API calls to resolve SMIL file requests \(AMLST\)](#).

Content Storage

By default Wowza Streaming Engine is configured to stream VOD content from (and record VOD content to) the **[install-dir]/content** folder. You can specify a different storage location for a VOD application in Wowza Streaming Engine Manager by changing the **Content Directory** value for the application. For example, to configure an application to use an application-specific content folder, you can select the **Application-specific directory** option:

Content Directory

☐ Use default

`${com.wowza.wms.context.VHostConfigHome}/content`

☒ Application-specific directory

`${com.wowza.wms.context.VHostConfigHome}/content/vod`

☐ Use the following directory:

Using this setting, content is streamed from the **[install-dir]/content/[application]** folder, where **[application]** is the application's name (**vod**).

Files that are required for streaming live content, such as Session Description Protocol (SDP) files or **.stream** files are also stored in the **[install-dir]/content** folder by default. You can specify a different storage location for a live application in Streaming Engine Manager by changing the **Streaming File Directory** value for the application. For example, to configure an application to use an application-specific folder, you can select the **Application-specific directory** option:

Streaming File Directory *

☐ Use default

`${com.wowza.wms.context.VHostConfigHome}/content`

☒ Application-specific directory

`${com.wowza.wms.context.VHostConfigHome}/content/live`

☐ Use the following directory:

Using this setting, the files can be accessed from the **[install-dir]/content/[application]** folder, where **[application]** is the application's name (**live**).

You can further customize content storage for your applications by specifying the fully qualified path to the storage location in the **Use the following directory** box. You can substitute variables in place of path elements. The following variables are supported:

<code>\${com.wowza.wms.AppHome}</code>	- Application home directory
<code>\${com.wowza.wms.ConfigHome}</code>	- Configuration home directory
<code>\${com.wowza.wms.context.VHost}</code>	- Virtual host name
<code>\${com.wowza.wms.context.VHostConfigHome}</code>	- Virtual host configuration directory
<code>\${com.wowza.wms.context.Application}</code>	- Application name
<code>\${com.wowza.wms.context.ApplicationInstance}</code>	- Application instance name

Advanced Configuration Topics

How do I take advantage of Wowza Streaming Engine features?

This chapter covers more advanced streaming topics. Some of the functionality discussed is provided by [AddOn packages](#). AddOn packages are downloadable packages that include server extensions along with documentation for adding more advanced features to the Wowza Streaming Engine™ software.

MediaCasters, Stream Files, and Startup Streams

Wowza Streaming Engine includes a system for re-streaming live streams called *MediaCaster*. The MediaCaster system is used to re-stream IP camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and streaming output from native RTP or MPEG-TS encoders. The MediaCaster system pulls a stream from a stream source and makes it available for streaming to all player technologies supported by the Streaming Engine. This system works on demand—when the first request is received from a player for a given stream, a connection is made to the source stream and the stream is then made available to the player. When the last player stops watching the stream, the MediaCaster system waits for a timeout period. If no other players request the stream, the stream is stopped and isn't available for streaming until another request is made.

This on-demand startup methodology works great for RTMP and RTSP/RTP streaming where advanced packetization isn't required. However, the model doesn't work for the HTTP streaming protocols (Adobe HDS, Apple HLS, Microsoft Smooth Streaming, and MPEG-DASH streaming). An Apple iOS device requires about 30 seconds of video to be pre-packetized before playback can start and Microsoft Silverlight clients require three times the key frame duration. Therefore, the stream must be started before it's ready for streaming over HTTP.

Wowza Streaming Engine Manager provides features to start receiving MediaCaster streams and to keep them running.

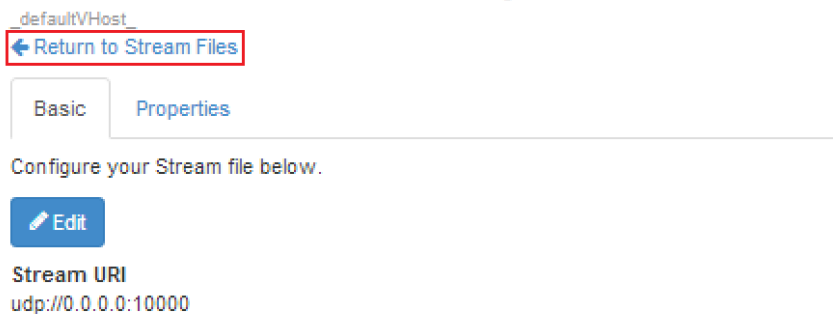
Stream Files

An easy method for re-streaming live MediaCaster streams is to configure a Stream file (a file with a **.stream** file name extension) that live applications can use to connect to the source stream through the MediaCaster system. A Stream file just contains the URI of the source stream. When the source stream is started, a live application can use the information in the Stream file to connect to the stream so that it's available for playback when requested by players.

As an example, to create a Stream file named **mycoolevent.stream**, do the following:

1. Click the **Server** tab in Streaming Engine Manager and then click **Stream Files** in the contents pane.
2. On the **Virtual Host Stream Files** page, click **Add Stream File**.
3. In the **Add Stream File** dialog box, enter the name **mycoolevent** and then click **Add**. The **mycoolevent.stream** page is displayed.
4. In **Stream URI**, enter the source stream URI and then click **Save**. For example, if you're using an MPEG-TS encoder, the URI value might be **udp://0.0.0.0:10000**.
5. Click **Return to Stream Files**.

Virtual Host Stream Files > mycoolevent.stream







6. Click the **Connect** icon for **mycoolevent.stream**.

Virtual Host Stream Files

defaultVHost

Stream Files

 Copy Stream File  Add Stream File

Name	Actions
mycoolevent.stream	   

7. In the **Connect a Stream File** dialog box, configure the options to enable a live application to connect to the stream and then click **OK**.

Connect a Stream File X

Stream Name
mycoolevent.stream

Application Name
live ▼

Application Instance
☒ Connect to default application instance: _definst_
☐ Connect to application instance:

Enter an existing application instance name. The application instance will be created if it does not exist.

MediaCaster Type
rtp ▼

Be sure to select the **MediaCaster Type** in the list that corresponds to the source stream type:

- Select **rtp** for IP Camera streams (RTSP/RTP streams) and for streams from native RTP and MPEG-TS encoders.
- Select **shoutcast** for SHOUTcast/Icecast streams.
- Select **liverepeater** if the stream is pulled from another server that's running Wowza Media Server or Wowza Streaming Engine software.

You can then use **mycoolevent.stream** in the following example URLs to play the stream:

Adobe HDS

```
http://[wowza-ip-address]/live/mycoolevent.stream/manifest.f4m
```

Apple HLS

```
http://[wowza-ip-address]/live/mycoolevent.stream/playlist.m3u8
```

Microsoft Smooth Streaming

```
http://[wowza-ip-address]/live/mycoolevent.stream/Manifest
```

MPEG-DASH

```
http://[wowza-ip-address]/live/mycoolevent.stream/manifest.mpd
```

Adobe RTMP

```
Server: rtmp://[wowza-ip-address]/live  
Stream: mycoolevent.stream
```

RTSP/RTP

```
rtsp://[wowza-ip-address]/live/mycoolevent.stream
```

Note

In the **SMIL Files** feature in Wowza Streaming Engine Manager, you can connect to live MediaCaster streams that are referenced in Synchronized Multimedia Integration Language (SMIL) files. In the **Incoming Streams** feature, you can connect to MediaCaster streams to record them.

Startup Streams

The second method for starting live MediaCaster streams is to use the **Startup Streams** feature in Streaming Engine Manager to create stream entries in the **[install-dir]/conf/StartupStreams.xml** file. Stream entries in this file are automatically started when the server is started (or more specifically, when a virtual host is started). The format of a single entry in **StartupStreams.xml** is:

```
<StartupStream>  
  <Application>[application]</Application>  
  <MediaCasterType>[mediacaster-type]</MediaCasterType>  
  <StreamName>[stream-name]</StreamName>  
</StartupStream>
```

-where-

```
[application]:
  - name of existing live application that will re-stream the source
  stream

[mediacaster-type]:
  - a valid mediacaster type: rtp, rtp-record, shoutcast, shoutcast-
  record, liverepeater

[stream-name]:
  - name of the source stream
```

As an example, to create a stream entry in **StartupStreams.xml**, do the following:

1. Click the **Server** tab in Streaming Engine Manager and then click **Startup Streams** in the contents pane.
2. On the **Virtual Host Startup Streams** page, click **Add Startup Stream**.
3. In the **Add to Startup Streams** dialog box, configure the options to create the entry in the **StartupStreams.xml** file and then click **OK**.

Add to Startup Streams X

Stream Name
mpegs.stream

Application Name
live ▼

Application Instance
☒ Connect to default application instance: _definst_
☐ Connect to application instance:

 Enter an existing application instance name. The application instance will be created if it does not exist.

MediaCaster Type
rtp ▼

Cancel OK

For more information, see [How to start streams at server startup](#).

Note

The following server-side methods can also be used to start and stop streams using the MediaCaster system:

```
IApplicationInstance.startMediaCasterStream(...);
IApplicationInstance.stopMediaCasterStream(...);
```

For more information about these methods, see the [Wowza Streaming Engine Server-Side API](#).

Live Stream Repeater (Origin/Edge Live Streaming)

A live stream repeater uses multiple Wowza® servers in an origin/edge configuration to deliver live media content across multiple servers. The encoded media content is delivered to the origin server in the same manner as if you were delivering the content to a single Wowza server. A player will request the content from an edge server, which maintains a single connection per-unique stream to the origin. Origin/Edge configuration occurs at the application level. A single Wowza Streaming Engine instance can be configured as an origin for one application and as an edge for another application.

The example in this section uses a single origin server with an application named **liveorigin**. To configure the origin server, do the following:

1. In Wowza Streaming Engine Manager, click the **Applications** tab.
2. On the **Add Application** page, click **Live**.
3. In the **New Application** dialog box, enter the following application name: **liveorigin**
4. On the **liveorigin** application page, select the following **Playback Types**:
 - **MPEG-DASH**
 - **Apple HLS**
 - **Adobe HDS**
 - **Microsoft Smooth Streaming**
5. Click **Save**.

To configure an edge server, do the following (repeat on each edge server):

1. In Wowza Streaming Engine Manager, click the **Applications** tab.
2. On the **Add Application** page, click **Live Edge**.
3. In the **New Application** dialog box, enter the following application name: **liveedge**

4. On the **liveedge** application page, select the following **Playback Types**:
 - **MPEG-DASH**
 - **Apple HLS**
 - **Adobe HDS**
 - **Microsoft Smooth Streaming**
5. If low latency is important, select the **Low-latency stream** check box (this will add extra load to the server).
6. In **Primary Origin URL**, Enter the URL of the **liveorigin** application using the WOWZ™ protocol URL prefix (wowz://). For example, if the origin server uses the domain name **origin.mycompany.com**, the value would be:


```
wowz://origin.mycompany.com/liveorigin
```
7. Click **Save**.

In the following examples, assume that the origin server uses the domain name **origin.mycompany.com** and that there are 3 edge servers with the domain names **edge1.mycompany.com**, **edge2.mycompany.com**, and **edge3.mycompany.com**. If the stream name is **mycoolevent**, the URLs for players streaming from **edge1** would be:

Adobe HDS

```
http://edge1.mycompany.com:1935/liveedge/mycoolevent/manifest.f4m
```

Apple HLS

```
http://edge1.mycompany.com:1935/liveedge/mycoolevent/playlist.m3u8
```

Microsoft Smooth Streaming

```
http://edge1.mycompany.com:1935/liveedge/mycoolevent/Manifest
```

MPEG-DASH

```
http://edge1.mycompany.com:1935/liveedge/mycoolevent/manifest.mpd
```

You can configure more than one origin server to provide a hot backup if the primary origin server goes offline. For example, if the failover origin server has the domain name **origin2.mycompany.com**, and it's configured identically as the primary origin server, you would specify the following **Secondary Origin URL** value in the **liveedge** application page on each edge server:

```
wowz://origin2.mycompany.com/liveedge
```

Edge servers will try to connect to the first origin server, and if this fails, they will try to connect to the second origin server.

This example assumes that you're using an encoder in which the stream name is a simple name and not a URL. If you're using an encoder such as an MPEG-TS encoder in which the stream name isn't a simple stream name, you can use **.stream** files on the origin server to hide the complex stream names. For example, if your complex stream name on the origin server is **udp://0.0.0.0:10000**, use the **Stream Files** feature in Streaming Engine Manager to create a file named **mycoolevent.stream** and set the contents to **udp://0.0.0.0:10000**. You can then use **mycoolevent.stream** in place of **mycoolevent** in the example URLs above to play the stream.

Notes

- The WOWZ™ protocol is a TCP-based messaging protocol in Wowza Streaming Engine and is used for server-to-server communication. It's enabled by default. If one of the Wowza servers in the origin/edge configuration is running a version of the software that doesn't support the WOWZ protocol, an RTMP connection is established between that server and other servers instead.
- You can secure the connection between Wowza servers in and origin/edge configuration by using a SecureToken shared secret. For more information, see [How to configure a live stream repeater](#).
- If you use a non-push-based encoder (native RTP or MPEG-TS) and streaming players using any of the HTTP streaming protocols, you must use the **Startup Streams** feature in Streaming Engine Manager to start the stream on the origin server and keep it running. Streams don't need to be kept running on edge servers.
- To provide load balancing between edge servers, you can use the dynamic load balancing system. For more information, see [Dynamic Load Balancing](#).

Live Stream Recording

There are multiple ways to record incoming live streams to VOD files for later playback, but the **Incoming Streams** feature for live applications in Wowza Streaming Engine Manager gives you the most control over the recording process. You can split in-process live stream recording archives into multiple on demand MP4 (QuickTime container) or FLV (Flash Video container) files automatically, with the split points based on video duration, clock time, or file size. The user interface shows all current incoming live streams and enables you to control when the recording starts and stops, the file name and locations, the container format, and other details. You can also control the live stream recording process using HTTP URL queries

and programmatically using the **LiveStreamRecordManager** APIs. For more information, see [How to record live streams](#).

For the Live and Live HTTP Origin application types in Streaming Engine Manager, you can select the **Record all incoming streams** option to record all incoming streams published to the application by an encoder. This recording option uses the **live-record** stream type and creates a recording with a file name that's the same as the incoming stream name in the application's streaming file directory. To stop recording all incoming streams to these application types, you must clear the **Record all incoming streams** option and restart the application.

Finally, you can record incoming IP camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and streaming output from native RTP or MPEG-TS encoders using the MediaCaster system. The **Stream Files** and **Startup Streams** features in Streaming Engine Manager use the MediaCaster system to pull a stream from a stream source and make it available for streaming to all player technologies supported by the Streaming Engine. You can configure these features to record the incoming streams instead by selecting an appropriate ***-record** stream type for the MediaCaster type (such as **rtp-record** for IP camera streams) and the streams will be recorded to the streaming file directory for the selected application. For more information, see [MediaCasters, Stream Files, and Startup Streams](#).

Notes

- The ***-record** stream types are the easiest to use but also give you the least amount of control. If you use this method, the entire duration of the published stream is recorded to a single file in the live application's streaming file directory. If the stream source starts and stops, the file is versioned with a version number and a new file is started. You can control the container format used (MP4 or FLV) by specifying a stream name prefix in the stream source. If you specify the **mp4:** prefix, the stream is recorded to an MP4 (QuickTime) container. An MP4 container can only record H.264, AAC, and MP3 media data. If you specify the **flv:** prefix, the stream is recorded to an FLV container. The FLV container is the only option if you're recording with Flash Player.
- If you use one of the ***-record** stream types and also configure the **Incoming Streams** feature for a live application to record an incoming live stream, two or more copies of the recording will be created in the live application's streaming file directory by default. The ***-record** stream types record the stream to a single file and the recorded file name is the same as the stream name. The **Incoming Streams** feature creates one or more recordings with file names that include the stream name and other information, depending on selected segmentation and versioning options.
- The [WebcamRecording example](#) in the Wowza Streaming Engine installation is a specialized way to record a remote live stream when using Adobe Flash Player. It uses the **record** stream type and built-in Flash Player capabilities to control the recording process.

Virtual Hosting

The Wowza Streaming Engine software can be configured to run multiple virtual host (VHost) environments on a single server. This lets multiple users share a server in separate environments. Each VHost environment has its own set of configuration files, application folders, and log files and can be configured with its own system resource and streaming limitations. By default, a Wowza server is configured with a single VHost named **_defaultVHost_**.

Configuration Files

The **VHosts.xml** configuration file in the Wowza Streaming Engine **[install-dir]/conf** folder is used to define each of the VHost environments. The following items are required in **VHosts.xml** to define a VHost:

- **VHosts/VHost/Name.** The name of the VHost.
- **VHosts/VHost/ConfigDir.** The configuration directory for the VHost. See [Typical Configuration](#) to view a sample directory structure.
- **VHosts/VHost/ConnectionLimit.** The maximum number of simultaneous connections that the VHost supports. If this value is **0**, the VHost can support an unlimited number of simultaneous connections.

Typical Configuration

A typical **VHosts.xml** file for a VHost environment contains two VHosts. The following example shows the default VHost (**_defaultVHost_**) and a new VHost (**_newVHost_**):

```
<Root>
  <VHosts>
    <VHost>
      <Name>_defaultVHost_</Name>
      <ConfigDir>${com.wowza.wms.ConfigHome}</ConfigDir>
      <ConnectionLimit>0</ConnectionLimit>
    </VHost>
    <VHost>
      <Name>_newVHost_</Name>
      <ConfigDir>${com.wowza.wms.ConfigHome}/newVHost</ConfigDir>
      <ConnectionLimit>0</ConnectionLimit>
    </VHost>
  </VHosts>
</Root>
```


The directory structure for the VHosts in the above example would be:

```
[install-dir]
  [defaultVHost]
    [applications]
    [conf]
      Application.xml
      clientaccesspolicy.xml
      crossdomain.xml
      MediaCache.xml
      StartupStreams.xml
      Tune.xml
      VHost.xml
      admin.password
      publish.password
    [content]
    [keys]
    [logs]
    [transcoder]
  [newVHost]
    [applications]
    [conf]
      Application.xml
      clientaccesspolicy.xml
      crossdomain.xml
      MediaCache.xml
      StartupStreams.xml
      Tune.xml
      VHost.xml
      admin.password
      publish.password (Optional, see Notes below)
    [content]
    [keys]
    [logs]
    [transcoder]
```

Notes

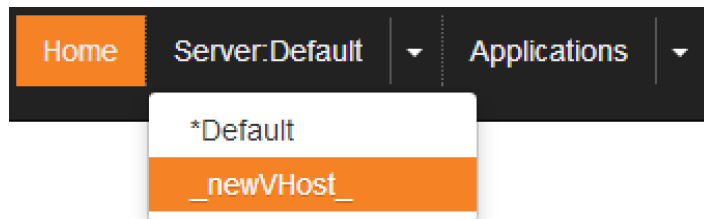
- By default, all VHost environments share the **publish.password** file for the default VHost. You can use the **Publishers** feature in Wowza Streaming Engine Manager to set up unique publishing credentials for each VHost and the unique credentials will be stored in this file.

Alternatively, you can retain the **publish.password** file when you copy the **[install-dir]/conf** folder to your new VHost environment and then configure the **securityPublishPasswordFile** property for new VHost applications to reference this file for Publisher credentials. If you do this, you can't use the **Publishers** feature in the manager to update the file. For more information, see the [Custom Password Provider class or Password File location](#).

- For more information about how to configure per-VHost logging, see [Logging](#).

The VHost configuration process is simple. VHosts are defined in the **[install-dir]/conf/VHosts.xml** file. Each VHost gets its own configuration directory structure with its own set of configuration files and **application**, **conf**, and **logs** folders. VHosts can be added, modified, and deleted through the **VHosts.xml** configuration file. If you change **VHosts.xml** while Wowza Streaming Engine is running, the changes take effect after restarting the server.

After adding a new VHost to **VHosts.xml** and creating its directory structure, sign in to Wowza Streaming Engine Manager, click the **Server** tab, and then select the new VHost in the list to manage it.



It's important to note that Wowza Streaming Engine only supports IP address/port-based virtual hosting. It doesn't support domain name-based virtual hosting. In **VHost.xml**, each VHost must define **HostPort** entries with unique IP address and port combinations that don't conflict with other VHosts that are defined on the server. The following combinations represent valid VHost port configurations:

```
defaultVHost:
<HostPort>
  <IpAddress>192.168.1.2</IpAddress>
  <Port>1935</Port>
</HostPort>

newVHost:
<HostPort>
  <IpAddress>192.168.1.2</IpAddress>
  <Port>1936</Port>
</HostPort>
```

-or-

```
defaultVHost:
<HostPort>
  <IpAddress>192.168.1.2</IpAddress>
  <Port>1935</Port>
</HostPort>

newVHost:
<HostPort>
  <IpAddress>192.168.1.3</IpAddress>
  <Port>1935</Port>
</HostPort>
```

To set up the IP address and port values, click the **Server** tab in Streaming Engine Manager, select a VHost in the list, and then click **Virtual Host Setup** in the contents pane. In the **Virtual Host Setup** page, click **Edit** to update the IP addresses and port values for the default host ports.

Server-side Publishing (Stream and Publisher Classes)

Wowza Streaming Engine includes the **Stream** class and the **Publisher** class for doing server-side publishing. The **Stream** class is a high-level server-side API for mixing live and VOD content on the fly into a single destination stream and lets you do television-style publishing. It also includes a package that enables creation of a server-side XML-based playlist. For more information about the **Stream** class, see [How to do scheduled streaming with Stream class streams](#).

The **Publisher** class is a low-level publishing API that lets you inject raw compressed video and audio frames into Wowza Streaming Engine to create a custom live stream. See the **Publisher** class server API Javadocs ([\[install-dir\]/documentation/serverapi](#)) for the current detailed documentation. The article [How to use Publisher API and JSpeex to publish an audio stream \(VOIP integration\)](#) includes an audio sample that walks through the process of publishing Speex data to a stream.

Server Management and Monitoring

How do I manage and monitor Wowza Streaming Engine?

Wowza Streaming Engine™ Manager enables you to conveniently set up, manage, monitor, and measure video and audio streams using a web browser on your computer, tablet, or phone. The new browser-based application extends the programmatic and command line configuration and management of the Wowza Streaming Engine software, enabling publishers with a diverse range of technical abilities to have greater control and confidence when streaming video.

You can use Streaming Engine Manager with the latest versions of most modern web browsers that support HTML5 and CSS 3. We recommend that you use the Google Chrome browser. On Windows operating systems, if you have multiple browsers installed on your computer, you can ensure that the web application always opens in the browser that you want to use by configuring the **Default Programs** feature.

Starting and Stopping Wowza Streaming Engine Manager

Notes

- Wowza Streaming Engine must be started to use Wowza Streaming Engine Manager. See [Starting and Stopping Wowza Streaming Engine](#).
- Streaming Engine Manager can't run as a service and in standalone mode at the same time.

- After starting Streaming Engine Manager, you can open it in a web browser with the following URL: `http://[wowza-ip-address]:8088/enginemanager`.
Where **[wowza-ip-address]** is the Wowza Streaming Engine IP address or domain name.
- On Windows operating systems, you can open Streaming Engine Manager in the default browser from the **Start** menu (**Start > All Programs > Wowza Streaming Engine 4.0.0 > Wowza Streaming Engine Manager**).
- For more information about how to sign in to Streaming Engine Manager, see [Managing Sign-In Credentials](#).

Windows

Service

To start the Wowza Streaming Engine Manager service:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).
2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine Manager 4.0.0**, and then click **Start**.

To stop the service:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).
2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine Manager 4.0.0**, and then click **Stop**.

Wowza Streaming Engine Manager can be set to start automatically as a Windows service when Windows starts. To prevent the service from starting automatically when Windows starts:

1. Open the Services MMC snap-in (press WIN key + R, type **services.msc** in the **Run** dialog box, and then click **OK**).
2. In the **Services** MMC snap-in, right-click **Wowza Streaming Engine Manager 4.0.0**, and then click **Properties**.
3. In the **Properties** dialog box, on the **General** tab, set **Startup type** to **Manual**.

Standalone

To start Wowza Streaming Engine Manager in standalone mode, make sure that the Wowza Streaming Engine Manager service is stopped (see above), and then do the following:

1. Open a Command Prompt window (press WIN key + R, type **cmd** in the **Run** dialog box, and then press ENTER).
2. Execute the following commands:

```
cd %WMSAPP_HOME%\manager\bin
startmgr.bat
```

To stop the manager:

1. Open a Command Prompt window (press WIN key + R, type **cmd** in the **Run** dialog box, and then press ENTER).
2. Execute the following commands:

```
cd %WMSAPP_HOME%\manager\bin
shutdownmgr.bat
```

Mac OS X

Service

To start the Streaming Engine Manager as a Mac OS X launchd service, open a terminal window and enter the following command:

```
sudo launchctl load -w
/Library/LaunchDaemons/com.wowza.WowzaStreamingEngineManager.plist
```

To stop the service, enter:

```
sudo launchctl unload -w
/Library/LaunchDaemons/com.wowza.WowzaStreamingEngineManager.plist
```

Standalone

To start the manager in standalone mode, invoke the **Wowza Streaming Engine Startup** script in **/Library/WowzaStreamingEngine-4.0.0/bin** or open a terminal window and enter the following commands:

```
cd /Library/WowzaStreamingEngine-4.0.0/manager/bin
./startmgr.sh
```

To stop the manager, invoke the **Wowza Streaming Engine Shutdown** script in **/Library/WowzaStreamingEngine-4.0.0/bin** or open a terminal window and enter the following commands:

```
cd /Library/WowzaStreamingEngine-4.0.0/manager/bin
./shutdownmgr.sh
```

Note

Invoking the **Wowza Streaming Engine Startup** and **Wowza Streaming Engine Shutdown** scripts also starts and stops Wowza Streaming Engine. See [Starting and Stopping Wowza Streaming Engine](#).

Linux

Note

The operations in this section must be performed as the **root** user with **sudo** access.

Service

To start the Streaming Engine Manager as a Linux service, enter one of the following commands (the commands differ based on your Linux distribution):

```
sudo service WowzaStreamingEngineManager start
```

-or-

```
/etc/init.d/WowzaStreamingEngineManager start
```

To stop the manager, enter:

```
sudo service WowzaStreamingEngineManager stop
```

-or-

```
/etc/init.d/WowzaStreamingEngineManager stop
```

Note

The method of running init.d-based services may be different on different Linux distributions. If these instructions don't apply to your Linux distribution, consult your Linux manual.

Standalone

To start the manager in standalone mode, open a terminal window and enter the following commands:

```
cd /usr/local/WowzaStreamingEngine/manager/bin  
./startmgr.sh
```

To stop the manager, enter:

```
cd /usr/local/WowzaStreamingEngine/manager/bin  
./shutdownmgr.sh
```

Managing Sign-In Credentials

The first time you start Wowza Streaming Engine Manager, you'll be asked to sign in with the case-sensitive user name and password that you created when you installed the Streaming Engine software. This account has administrator access to enable control of the Streaming Engine through the manager. However, it doesn't provide access to advanced property, server listener, and module settings, which are reserved for expert Wowza users.

After you sign in, you can enable access to the advanced settings for the default administrator account and add accounts for other users. You can create additional user accounts with both administrative and read-only access.

To enable access to advanced settings for the default administrator account

1. In Streaming Engine Manager, click the **Server** tab and then click **Users** in the contents pane.
2. On the **Users** page, click the user name for the administrator account in the **Users** list.
3. Click **Edit**, and then select the **Allow access to advanced properties and features** check box.
4. (Optional) Enter a new password in the **Password** and **Confirm Password** fields. The password values are case-sensitive.
5. Click **Save**. As the signed-in user, you'll be signed-out automatically and must sign in again.

You can also enable access to the advanced settings for the default administrator account by updating the `[install-dir]/conf/admin.password` file using a text editor. For example, to specify that the **Admin** user can access the advanced settings, specify the **advUser** group as shown in the following example:

```
# Admin password file (format [username] [space] [password] [space] [group])
#username password group|group
Admin AdminPassword admin|advUser
```

Administrators can create accounts for other users with full administrative access to the Streaming Engine manager or with read-only privileges.

To create new user accounts

1. In Streaming Engine Manager, click the **Server** tab and then click **Users** in the contents pane.
2. On the **Users** page, click **Add User**.
3. Enter a name for the user in **User Name** and a password for the user in **Password** and **Confirm Password** fields. The user name and password values are case-sensitive.
4. Specify the access level (**Read-Only** or **Administrator**) for the new user by selecting the appropriate **Access Level** option.
5. To enable the new user to either manage (**Administrator** user) or view (**Read-Only** user) advanced settings, select the **Allow access to advanced properties and features** check box.
6. Click **Add**.

You can also add new user accounts by updating the `[install-dir]/conf/admin.password` file using a text editor. For example, to add the **newAdmin** and **readOnly** user accounts with access to advanced settings, edit the **admin.password** file as follows.

```
# Admin password file (format [username] [space] [password] [space] [group])
#username password group|group
Admin AdminPassword admin|advUser

newAdmin newAdminPassword admin|advUser
readOnly readOnlyPassword advUser
```

The **readOnly** user can view the advanced settings but can't change them.

Navigating in Wowza Streaming Engine Manager

This section introduces the different parts of the Streaming Engine Manager browser-based application to help you find your way around the user interface. For additional details, see [How to find your way around Wowza Streaming Engine Manager](#).

Home Page

Welcome to Wowza Streaming Engine!

Monthly Subscription **2**

Status

Connections Incoming and outgoing

Usage CPU, Memory, Heap and Disk

Wowza CPU
Wowza Heap
Total Memory
Total Disk

Server Uptime
Since 09 Feb 2014 09:12:58 AM

AddOns
Transcoder: **Licensed**
DRM: **Licensed**
nDVR: **Licensed**

Test Video **4**

To play a video on demand test video, click the Test Players button.

Test Players...

Performance Warning! You are currently using *Developer* performance settings. If this server is running in a production environment, go to the [Performance Tuning > Java Settings](#) page to switch to *Production* performance settings.

Connection Settings **3**

Use the following settings to publish a stream to Wowza Streaming Engine:

Server IP	192.168.1.100
Port	1935
Application	A live application name on this server
Stream Name	The stream name you want to use
Login	A valid Publisher user name and password

Getting Started With Applications **5**

Wowza Streaming Engine uses **applications** to deliver streaming content. An application is a set of settings for live or video on demand (VOD) streaming. Either use the preinstalled default applications or go to the [Add Application](#) page to easily create and configure new applications.

Live Applications

A **live** streaming application is preinstalled to allow you to easily publish video directly from a video encoder or IP camera to Wowza Streaming Engine. Visit the [Wowza Forums](#) for instructions about how to work with common encoders and cameras.

- 1** Click the tabs on the menu bar to access features that help you manage the server and virtual host (the **Server** tab) and to create and manage different live and video on demand application types (the **Applications** tab). Click the **Help** tab to access popular articles and other resources on the Wowza website that can help you configure streaming scenarios.
- 2** View information in this area about how the server instance is licensed and the number of days until the license expires. You can also see if the Wowza Transcoder, Wowza DRM, and Wowza nDVR AddOns are licensed, and if they're enabled, which applications they're enabled for. The **Connections** chart shows the total number of connections (both incoming and outgoing connections) for the server. The **Usage** chart shows total server resource consumption for CPU, Java heap, memory, and disk.
- 3** Use the information (IP address and port) shown in **Connection Settings** to publish a stream to the server from your camera or encoder.
- 4** Quickly verify that the server is up-and-running by using built-in test players to stream the **sample.mp4** video file that's installed with the server software over multiple streaming protocols.

- 5 Use the **Getting Started** information to quickly jump to configuration areas in Streaming Engine Manager and to get more information about the Support resources that are available to you if you have problems.

Server Configuration

The screenshot displays the Wowza Streaming Engine Manager interface. The top navigation bar includes 'Home', 'Server', 'Applications', and 'Help'. The left sidebar (labeled 1) lists various configuration options under 'SERVER' and 'STREAMS'. The main content area (labeled 2) is titled 'Server Setup' and contains fields for Name, Description, License Keys, Default Stream Prefix, and Options. A right sidebar (labeled 4) provides additional information about license keys. A 'Restart...' button is located in the top right corner (labeled 3).

- 1 The contents pane provides access to the following features that let you configure, manage, and monitor the server and virtual hosts (VHosts).

Server Setup: Configure settings for the Streaming Engine instance such as the instance name, available license keys, and enabling/disabling the Monitoring features for the server and its applications.

Server Monitoring: Monitor server resource consumption (CPU, memory, Java heap, and disk usage), incoming and outgoing connections, network throughput, and uptime. For details, see [How to monitor server connections, load, and application](#)

[statistics.](#)

Virtual Host Setup: Manage virtual hosting environments on the server. By default, the Streaming Engine ships with a single VHost environment named `_defaultVHost_`; however, you can add more VHost environments and manage them separately with this feature. For details, see [Virtual Hosting](#).

Virtual Host Monitoring: Monitor VHost incoming and outgoing connections, network throughput, and uptime. For details, see [How to monitor server connections, load, and application statistics](#).

Transcoder AddOn: Monitor the number of concurrent inbound source streams (*channels*) ingested by Wowza Transcoder and add, modify, and delete the Transcoder templates. For details, see the [Wowza Transcoder AddOn User's Guide](#).

Media Cache: Configure the read-through caching mechanism that enables scaling of video on demand (VOD) streams by re-streaming VOD file sources from HTTP-based servers that support HTTP/1.1 range requests and from network-attached file systems. For details, see [How to scale video on demand streaming with Media Cache](#).

Users: Set up and manage administrator and read-only user accounts for Wowza Streaming Engine Manager. For details, see [Managing Sign-In Credentials](#).

Publishers: Create and manage case-sensitive user names and passwords that RTMP-based and RTSP-based encoders can use to connect and publish a live stream if the live application requires authentication.

Performance Tuning: Adjust server performance settings from the default values that are calculated when the server starts. For details, see [How to do performance tuning](#).

About: View information about the Wowza Streaming Engine platform such as the installed Wowza Streaming Engine software version and license and the installed Java Runtime Environment (JRE).

Startup Streams: Pull live IP Camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and streams from native RTP or MPEG-TS encoders and start them automatically when the VHost starts. For details, see [Startup Streams](#).

Stream Files: Replace (alias) complex stream names that are published to Wowza Streaming Engine from sources such as IP Camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and streams from native RTP or MPEG-TS encoders. For

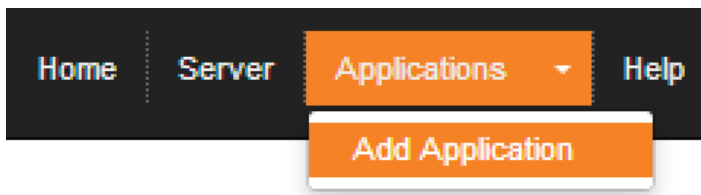
details, see [Stream Files](#).

SMIL Files: Create Synchronized Multimedia Integration Language (SMIL) files that organize streams of various bitrates into groups for HTTP adaptive bitrate streaming. For details, see [How to do adaptive bitrate streaming](#).

- 2 When you click a server feature in the contents pane, a page is displayed that enables you to configure the feature settings. Advanced settings for fine-tuning the server configuration are available for some of the server features on **Properties** and **Server Listeners** tabs. These tabs are only available to users with advanced permissions. For details, see [Advanced Properties and Settings](#).
- 3 Some features have buttons in the upper-right corner that provide additional functionality. Some server-level features let you restart the server and stop and restart the VHost.
- 4 The Help pane provides details about how to configure the controls on the feature page. You can click the **Hide Help** button to hide this information. If the information is hidden, you can click the **Show Help** button to bring it back into view.

Application Types

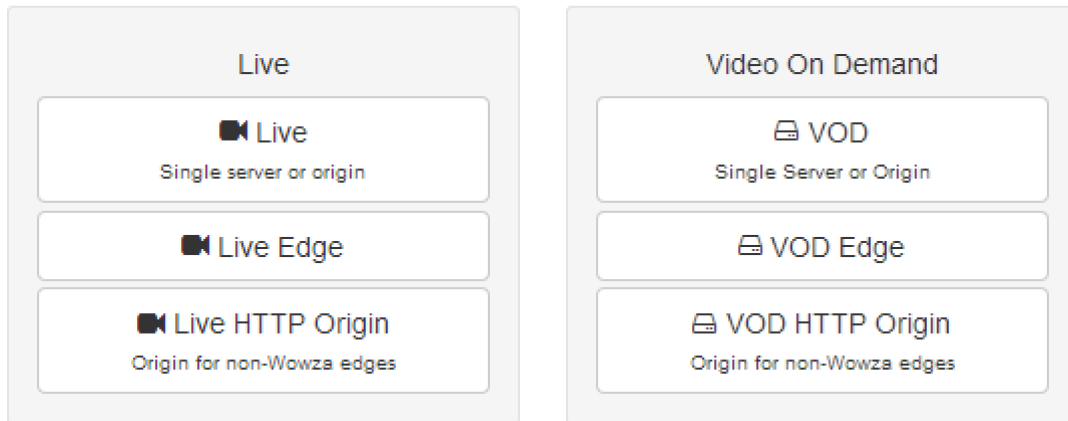
An *application* is a set of configuration options in Wowza Streaming Engine that supports a specific use case for the delivery of streaming content. To add applications in Wowza Streaming Engine Manager, click the **Applications** tab and then click **Add Application**.



In the **Add Application** page that's displayed, you can add applications for six streaming use cases.

Add Application

Select an Application Type.



Live

Use a Live application to deliver live streams to players (single server) or as an origin server to deliver live streams to other servers running Wowza Media Server® or Wowza Streaming Engine to scale content delivery to a large number of players.

Live Edge

Use a Live Edge application to ingest live streams from a live application on an origin server that's running Wowza Media Server or Wowza Streaming Engine. This application is then used to deliver the live streams to players (single server).

Live HTTP Origin

Use a Live HTTP Origin application to deliver live streams to an HTTP caching infrastructure using HTTP streaming protocols (MPEG-DASH, Apple HLS, Adobe HDS, and Microsoft Smooth Streaming).

VOD

Use a VOD application to stream video on demand (VOD) files to players (single server) or as an origin server to deliver VOD files to other servers running Wowza Media Server or Wowza Streaming Engine to scale content delivery to a large number of players.

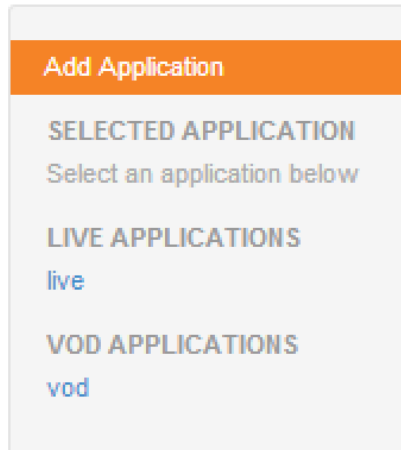
VOD Edge

Use a VOD Edge application to ingest video on demand files from a Media Cache source. This application is then used to stream the VOD files to players (single server).

VOD HTTP Origin

Use a VOD HTTP Origin application to deliver video on demand files to an HTTP caching infrastructure using HTTP streaming protocols (MPEG-DASH, Apple HLS, Adobe HDS, and Microsoft Smooth Streaming).

To add an application, click the **Application Type** in the page that corresponds to your use case, enter a name for the application in the **New Application** dialog box, and then click **Add**. Single instances of the Live application type (named **live**) and the VOD application type (named **vod**) are included in the default installation of Wowza Streaming Engine.



Application Configuration

Wowza Streaming Engine MANAGER Home Server **Applications** Help myUser | Sign Out

live Live Single Server or Origin

Test Players... Copy... Restart... Delete...

Setup

Configure playback options for your application. To configure incoming streams from your camera or encoder, use the [Incoming Publishers](#) page. Show Help

Edit

Application Description
Default application for live streaming created when Wowza Streaming Engine is installed. Use this application with its default configuration or modify the configuration as needed. You can also copy it to create another live application.

Playback Types

- ☒ MPEG-DASH
- ☒ Apple HLS
- ☒ Adobe RTMP
- ☒ Adobe HDS
- ☒ Microsoft Smooth Streaming
- ☒ RTSP/RTP

Options

- ☒ Low-latency stream (ideal for chat applications)
- ☒ Record all incoming streams

Streaming File Directory
\${com.wowza.wms.context.VHostConfigHome}/content

Closed Caption Sources
None

Maximum Connections
-Not Set-

- 1 The contents pane provides access to the following features that let you configure, manage, and monitor the different application types.

Application Setup: Modify application settings such as the playback types (http streamers and packetizers), default content storage location, closed-captioning options, and other settings. Some settings vary by application type.	All application types
Monitoring: Monitor application incoming and outgoing connections, network throughput, and uptime. For details, see How to monitor server connections, load, and application statistics .	All application types
Incoming Publishers: Get connection information for publishers that will publish a stream to this application. If you're viewing this page on your mobile device that has the Wowza® GoCoder™ encoding app installed, you can automatically configure the GoCoder app to publish a stream to this application.	All live application types
Incoming Streams: View details about live streams that are published to this application and record them to video on demand (VOD) files for later playback. For details, see How to record live streams .	All live application types
Incoming Security: Configure options for securing RTMP and RTSP-based incoming connections to this application (for example, from RTMP-based encoders).	All live application types
Outgoing Security: Configure options for securing outgoing (playback) connections to Wowza Streaming Engine. For example, require a secure RTMP connection, specify a security token ("shared secret"), and restrict playback to specific IP addresses.	All application types
Stream Files: Replace (alias) complex stream names that are published to the application from sources such as IP Camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and streams from native RTP or MPEG-TS encoders. For details, see Stream Files .	Live Live Edge
SMIL Files: Create Synchronized Multimedia Integration Language (SMIL) files that organize streams of various	Live Live Edge

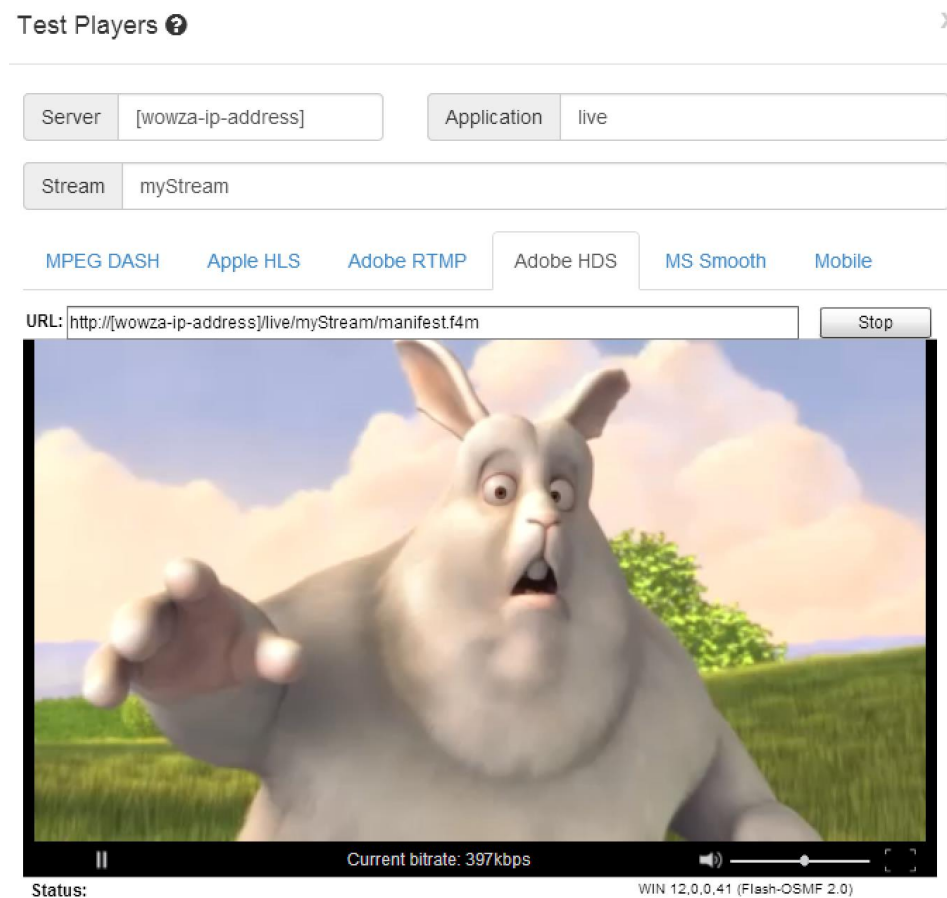
bitrates into groups for HTTP adaptive bitrate streaming. For details, see How to do adaptive bitrate streaming .	VOD VOD Edge
nDVR AddOn: Configure DVR playback of live streams using the Wowza nDVR AddOn. For details, see the Wowza nDVR AddOn User's Guide .	All live application types
Transcoder AddOn: Configure transcoding of live streams to suit desired playback devices using the Wowza Transcoder AddOn. For details, see the Wowza Transcoder AddOn User's Guide .	Live Live HTTP Origin
DRM AddOn: Integrate with DRM Key Management Service partners to enable on-the-fly DRM encryption of premium live and VOD content for a variety of playback devices. For details, see Wowza DRM AddOn .	Live Live Edge VOD VOD Edge

- 2 When you click an application or one of its features in the contents pane, a page is displayed that enables you to configure the application or feature settings. Advanced settings for fine-tuning the configuration are available for the application and some application features on **Properties** and **Modules** tabs. These tabs are only available to users with advanced permissions. For details, see [Advanced Properties and Settings](#).
- 3 Most application and feature pages have buttons in the upper-right corner that provide additional functionality. You can access [Test Players](#) to test your streams, copy application settings to create a new application with identical settings, restart an application, and delete an application.
- 4 The Help pane provides details about how to configure the controls on the application or feature page. You can click the **Hide Help** button to hide this information. If the information is hidden, you can click the **Show Help** button to bring it back into view.

Test Players

Wowza Streaming Engine Manager provides test players for all of the live and VOD application types so that you can test an application's streaming configuration. To access the test players, click the **Test Players** button in the upper-right corner of the application or feature page. Then in the **Test Players** dialog box, click the tab for the streaming protocol that you've configured for the application and want to test.

The test players for live applications are preconfigured to playback a live stream named **myStream** from the local Wowza Streaming Engine instance.




If you configured your encoder to publish a stream to the live application with a different stream name, be sure to substitute it in place of **myStream** in the **Stream** box.

The test players for VOD applications are preconfigured to playback the **[install-dir]/content/sample.mp4** video file that's installed with the server software.

Test Players ?

X

Server	[wowza-ip-address]	Application	vod
Media File Name	mp4:sample.mp4		
MPEG DASH Apple HLS Adobe RTMP Adobe HDS MS Smooth Mobile			
URL: <input type="text" value="http://[wowza-ip-address]/vod/mp4:sample.mp4/manifest.f4m"/>		<input type="button" value="Stop"/>	
			
<div> ⏮ ⏸ ⏭ ⏪ ⏩ ⏴ ⏵ </div> <div> Current bitrate: 363kbps </div>			
Status: Playing		WIN 12.0.0.41 (Flash-OSMF 2.0)	

If you want to use your own VOD file, you can copy it to the **[install-dir]/content** root folder and substitute its file name in place of **sample.mp4** in the **Media File Name** box. If your custom VOD file isn't stored in the **[install-dir]/content** root folder, you must add the default application instance name to the playback URL. For example, if the **sample.mp4** video file is in **[install-dir]/myVideos**, enter **vod/_definst_/myVideos** in the **Application** box.

Note

The test players can't display closed captions or playback encrypted streams. DVR playback is only supported by the Adobe HDS, Apple HLS, and Microsoft Smooth Streaming test players. If you change the default stream values to playback a new stream, you may need to restart the test players.

Advanced Properties and Settings

Advanced settings for fine-tuning the server and application configuration are available in Wowza Streaming Engine Manager. Some server features have advanced settings on **Properties** and **Server Listener** tabs to adjust the server configuration while applications and some application features have **Properties** and **Modules** tabs to adjust the application configuration. The tabs that provide access to the advanced properties and settings aren't visible unless the signed-in user has advanced permissions. Administrators with advanced permissions can configure the advanced properties and settings while Read-Only users can only view (and not change) the advanced properties and settings. For more information, see [Managing Sign-In Credentials](#).

Properties pages may have many properties that you can configure, so they're organized into categories. You can click a link in the **Quick Links** area at the top of the page to jump to the associated property settings. For example, you can click **Closed Captions**:

live ▶ Test Players... 📄 Copy... 🔄 Restart... 🗑 Delete...

Live Single Server or Origin

[Setup](#) [Properties](#) [Modules](#)

Note: Items on this page should be configured by advanced users only.

Quick Links Use the following links to jump to the correct section on this page.

[HTTP Streamers Cupertino Settings](#)
[MediaCaster Stream Monitor](#)
[RTP Jitter Buffer](#)

[RTSP/RTP Window Title](#)
[StreamRecorder Defaults](#)
[Streams](#)
[Closed Captions](#)

[Cupertino Streaming Packetizer](#)
[San Jose Streaming Packetizer](#)
[Smooth Streaming Packetizer](#)
[Custom](#)

To jump to the **Closed Captions** property settings for an application:

Closed Captions Properties for tuning closed captioning functionality in streams and for enabling debug logging for specific components of the closed captioning implementation. [Return to top ↑](#)

[✎ Edit](#)

Enabled	Name	Value
✖	captionUndefinedLanguageld	?
✖	cea608CaptionConverterColor	?
✖	maximumCaptionDuration	?
✖	closedCaptionLiveMaxDisplayTime	?
✖	cclngestCEA608EnableField1	?
✖	cclngestCEA608EnableField2	?
✖	cclngestCEA608LogCaptions	?

Many articles on the Wowza® website prescribe custom properties for tuning the server and applications and to add advanced functionality. Each article will describe how to add the custom properties using the **Custom Properties** area on a **Property** tab:

Custom Custom properties added by you to extend Wowza Streaming Engine functionality.

[Return to top](#) ↑



No custom properties defined.

If you can't find the property that you're looking for in the previous sections, click the **Edit** button and then click the **Add Custom Property** button on the **Custom Property** page.

For more information, see [Properties](#).

Adobe Flash Streaming and Scripting

What can I do with Wowza Streaming Engine and Adobe Flash Player?

The Wowza Streaming Engine™ software includes additional features that are only applicable to Adobe Flash Player when using the RTMP protocol (or any of its variants).

When used with Adobe Flash Player, the Streaming Engine is more than just a streaming server—it's an application server. It provides features such as shared objects, video chat, remote recording, and bi-directional remote procedure calls.

Streaming Basics

We'll start with the most basic code that's needed to play a live or video on demand stream in Flash. Assume that we've followed the instructions in [How to set up video on demand streaming](#) and that we have an application named **vod** that's configured to stream video on demand. In Adobe Flash Creative Suite 3 or later, do the following:

1. Create a new **Flash File** with ActionScript 3.0 support.
2. Open the library palette (On the **Window** menu, select **Library**).
3. Right-click in the library palette, and then select **New Video**. Enter **video** in **Symbol name**, and then click **OK** to create the video object.
4. Drag the **video** object from the library to the stage, and then in the properties palette, give it an instance name of **video1**.
5. In **Window > Actions**, select **Scene 1**.
6. Enter the following code:

```

var nc:NetConnection = new NetConnection();
var ns:NetStream = null;

function ncOnStatus(infoObject:NetStatusEvent)
{
    trace("ncOnStatus: "+infoObject.info.code);
    if (infoObject.info.code == "NetConnection.Connect.Success")
    {
        trace("Connection established");
        ns = new NetStream(nc);

        ns.bufferTime = 3;

        video1.attachNetStream(ns);

        ns.play("mp4:sample.mp4");
    }
}
nc.addEventListener(NetStatusEvent.NET_STATUS, ncOnStatus);

nc.connect("rtmp://localhost/vod");

```

7. On the **Debug** menu, select **Debug Movie**.

You should now be streaming the **sample.mp4** video file. This is the most basic ActionScript 3.0 code that's needed for live and video on demand playback. If you inspect the code, you'll see how simple it is. We create a **NetConnection** object for streaming and add an event listener so that we can be notified when the connection to Wowza Streaming Engine is established. When we're notified of a successful connection, we create a **NetStream** object and start to playback the stream.

The **LiveVideoStreaming** and **VideoOnDemandStreaming** example players that are installed with the Streaming Engine take this example a little further. The example players support progress bars, pause, stop, and full screen. Inspecting the code for the example players is a good next step for learning how to stream. The **VideoChat** and **WebcamRecording** examples are a great starting point to learn how to publish video and audio using the built-in **Camera** and **Microphone** objects. For more information, see the [Installed Examples](#) section in this document.

Pre-built Media Players

Building your own custom player with advanced functionality can be a daunting task. Another option is to use pre-built Flash video players. This section describes a few of the more popular Adobe Flash Player options.

Adobe FLVPlayback component

The Adobe FLVPlayback component is a pre-built video player component that you can add to your own Flash project. It includes features such as play, pause, seek, stop, and full screen. It comes with Adobe Flash CS3 or later. The component is updated occasionally, so it's best to keep your Adobe Flash software up-to-date to ensure that you're running the most recent version. The nice thing about this component is that it can be integrated into your custom Flash code.

JW Player

JW Player is pre-built Flash-based player offered by [LongTail Video](#). It includes a rich set of features such as playlists, skinning, closed-captioning, and ad-serving. It's fully supported and there's a commercial option. It also includes a built-in version of the Wowza SecureToken security mechanism.

For more information about how to use JW Player with Wowza Streaming Engine, see the following support articles:

- [How to use LongTail JW Player with Wowza Streaming Engine](#)
- [How to add SecureToken protection to LongTail JW Player](#)

Flowplayer

[Flowplayer](#) is an open source pre-built Flash-based player. It includes a rich set of features similar to JW Player. It also includes a built-in version of the Wowza SecureToken security mechanism.

For more information about how to use Flowplayer with Wowza Streaming Engine, see [How to use Flowplayer](#).

Strobe Media Playback player

The Strobe Media Playback player supports RTMP streaming and Adobe HDS streaming. The player is built on the Open Source Media Framework (OSMF) and is hosted by Adobe. For more information, see [How to use the Strobe Media Playback player](#).

Bi-directional Remote Procedure Calls

Wowza Streaming Engine supports bi-directional remote procedure calls to and from Adobe Flash Player. Bi-directional remote procedures calls are a way for ActionScript code running in Flash Player to invoke server-side Java methods and pass data to Wowza Streaming Engine. The server can, in turn, invoke client-side ActionScript methods and pass data to Flash Player. This enables rich client/server applications to be built using Flash Player and Wowza Streaming Engine. These features are available when using the RTMP protocol.

Calls from Flash Player to Wowza Streaming Engine are performed using the following method:

```
NetConnection.call(methodName, resultObj, params...)
```

For example, the following ActionScript 3.0 client-side code calls the server-side method **doSomething**, passes the parameters **value1** and **value2**, and receives a single return value:

```
function onMethodResult(returnVal:String):Void
{
    trace("onMethodResult: "+returnVal);
}
nc.call("doSomething", new Responder(onMethodResult), value1, value2);
```

Note

See [Custom Module Classes](#) for the server-side code for this method.

Receiving method calls from Wowza Streaming Engine is done by adding handler methods/functions to the client object that's attached to the **NetConnection** object. For example, the following ActionScript 3.0 client-side code adds the handler method **onSomethingHappened** that receives two string parameters **value1** and **value2**:

```
var clientObj:Object = new Object();
clientObj.onSomethingHappened(value1:String, value2:String):Void
{
    trace("onSomethingHappened: "+value1+"-"+value2);
}
nc.client = clientObj;
```

For more information about the programming model, see [Extending Wowza Streaming Engine Using Java](#).

Remote Shared Objects

Wowza Streaming Engine supports Adobe Flash remote shared objects (RSOs), which enable data-sharing between the Streaming Engine and multiple Flash Players. Remote shared objects are an extension of ActionScript objects that enable shared object data to be synchronized between Adobe Flash Players on the same or different client machines. Shared data is synchronized by the Streaming Engine through an event-based synchronization method. RSOs can also be persisted on the server to maintain data across sessions.

Each Flash Player that subscribes to a shared object is notified when the shared object data is updated. Shared object data can be changed client-side by Flash Player or server-side through the Wowza Streaming Engine **ISharedObject** API. The following example shows the ActionScript 3.0 code that's needed to create a remote shared object and set a value:

```
var nc:NetConnection = new NetConnection();
var test_so:SharedObject = null;
var timer:Timer = null;

function ncOnStatus(infoObject:NetStatusEvent)
{
    trace("ncOnStatus: "+infoObject.info.code);
    if (infoObject.info.code == "NetConnection.Connect.Success")
    {
        test_so = SharedObject.getRemote("test", nc.uri);
        test_so.addEventListener(SyncEvent.SYNC, syncEventHandler);
        test_so.connect(nc);

        timer = new Timer(1000, 1);
        timer.addEventListener(TimerEvent.TIMER, setSOProperty);
        timer.start();
    }
}

function syncEventHandler(ev:SyncEvent)
{
    trace("syncEventHandler");
    var infoObj:Object = ev.changeList;
    for (var i = 0; i < infoObj.length; i++)
    {
        var info:Object = infoObj[i];
        if (info.name != undefined)
            trace("  "+info.name+"="+test_so.data[info.name]);
        else
            trace("  [action]="+info.code);
    }
}
```

```
function setSOProperty(ev:TimerEvent):void
{
    test_so.setProperty("testName", "testValue");
}

nc.addEventListener(NetStatusEvent.NET_STATUS, ncOnStatus);

nc.connect("rtmp://localhost/vod");
```

Wowza provides a downloadable Adobe Flash example ([RemoteSharedObjects.zip](#)) that illustrates the basics of remote shared objects. It implements the basic remote shared object interface and the **onSync** event handler to highlight how data is synchronized between client connections.

Server-side Modules and HTTP Providers

What is a server-side module and what server-side functionality ships with Wowza Streaming Engine?

Much of the functionality delivered by the Wowza Streaming Engine™ software is done through server-side modules and HTTP Providers. Server-side modules are Java classes that are configured on a per-application basis and are loaded at application instance startup. They provide much of the functionality needed to control the streaming process. HTTP Providers are Java classes that are configured on a per-virtual host basis. They are lightweight HTTP servers that can be used to query or interact with the Wowza server. This chapter reviews these methods for extending Wowza Streaming Engine and the built-in Java classes that are available for use.

For more information about the programming model that you can use to create your own server-side extensions, see [Extending Wowza Streaming Engine Using Java](#).

Server-side Modules

Server-side modules are Java classes that are configured on a per-application basis and are dynamically loaded at application instance startup. One use of server-side modules is to provide remote methods that can be called from Adobe Flash Player. It's these methods that provide the play, publish, seek, pause, and resume functionality needed to control the streaming process in Flash Player. Another use of server-side modules is to control Adobe HDS, Apple HLS, Microsoft Smooth Streaming, MPEG-DASH, and RTSP/RTP streaming. For more information about how the API works, see [Extending Wowza Streaming Engine Using Java](#).

Server-side modules are configured by adding entries to the **Modules** list for an application in Wowza Streaming Engine Manager. The default **Modules** list for an application looks like this:

[Setup](#)
[Properties](#)
[Modules](#)

Note: Items on this page should be configured by advanced users only.

Modules Java classes that extend an application's functionality. The list below defines an order-dependent list of modules to be loaded for a given application. The modules are loaded dynamically when the application instance is loaded. The base (ModuleCore) module must be included by the application for it to operate properly.

[Edit](#)

Name	Description	Fully Qualified Class Name
base	Base	com.wowza.wms.module.ModuleCore
logging	Client Logging	com.wowza.wms.module.ModuleClientLogging
flvplayback	FLVPlayback	com.wowza.wms.module.ModuleFLVPlayback
ModuleCoreSecurity	Core Security Module for Applications	com.wowza.wms.security.ModuleCoreSecurity

Each of these modules is described in detail in [Built-in Server-side Modules](#). For more information about how to create custom server-side modules, see [Extending Wowza Streaming Engine Using Java](#).

Built-in Server-side Modules

This section briefly describes the server-side modules that are built-in with Wowza Streaming Engine. For more information about the methods that are provided in a module, see the [Wowza Streaming Engine Server-Side API](#).

ModuleCore – (com.wowza.wms.module.ModuleCore)

The ModuleCore module represents the server-side implementation of the Adobe Flash **NetConnection**, **NetStream**, and **SharedObject** objects. This module must be included by all applications for the server to operate properly. This module contains several additional server-side methods that are described in the following table.

Function call	Description
setStreamType (streamType:String) ; getStreamType () ;	Returns and sets the default stream type for this client connection.
setRepeaterOriginUrl (originUrl:Strin g) ;	Returns and sets the live stream repeater origin URL to use for this connection in an

<code>getRepeaterOriginUrl();</code>	origin/edge configuration.
<code>getStreamLength(streamName:String);</code> <code>getStreamLength(streamNames:Array);</code>	For video on demand streaming, returns the stream duration, in seconds. If an array of stream names is passed in, an array of durations is returned.
<code>getClientID();</code>	Returns the client ID for this client connection.
<code>getReferrer();</code>	Gets the referrer from the onConnect method.
<code>getPageUrl();</code>	Gets the page URL from the onConnect method.
<code>getVersion();</code>	Returns the server name and version.
<code>getLastStreamId();</code>	Returns the ID number of the last NetStream object that was created by this client.
<code>FCSubscribe(streamName, [mediaCasterType]);</code> <code>FCUnSubscribe(streamName);</code>	When using a live stream repeater (origin/edge), this method is useful for locking all bitrate renditions of an adaptive bitrate live stream on an edge server. This ensures that all streams are available when a switch is made between bitrate renditions.
<code>FCPublish(streamName);</code> <code>FCUnpublish(streamName);</code>	Called to tell Wowza Streaming Engine that a new stream is being published.

ModuleClientLogging - (com.wowza.wms.module.ModuleClientLogging)

The ModuleClientLogging module enables client-side logging to the server.

```
logDebug(logStr:String);
logInfo(logStr:String);
logWarn(logStr:String);
logError(logStr:String);
```

The following call from a Flash Player client:

```
nc.call("logDebug", null, "log this string");
```

Is the same as a server-side call to:

```
getLogger().debug("log this string");
```

ModuleFLVPlayback - (com.wowza.wms.module.ModuleFLVPlayback)

The ModuleFLVPlayback module is required by the FLVPlayback component. This module must be added to any application that uses the FLVPlayback component.

ModuleCoreSecurity - (com.wowza.wms.security.ModuleCoreSecurity)

The ModuleCoreSecurity module provides publishing and playback security. In Wowza Streaming Engine, it replaces many of the security modules that are included in the Module Collection for earlier versions of the server software. The security functionality provided by this module includes:

Publishing

- Enable/Disable RTMP publishing
- Require RTMP publishing password
- Allow duplicate stream names to be published or prevent them from being published
- Whitelist/Blacklist publishers by IP address

Playback

- Limit number of player connections
- Require secure RTMP connection for playback
- Require a security token for playback on Flash-based players
- Whitelist/Blacklist players by IP address

For more information, see the [How to configure security using Wowza Streaming Engine Manager](#).

HTTP Providers

HTTP Providers are mini HTTP servers that can be used to extend Wowza Streaming Engine functionality. They are configured on a per-port basis in `[install-dir]/conf/VHost.xml`. An individual HTTP Provider can be protected by a user name and password. Multiple HTTP Providers can be attached to one port and a specific HTTP Provider can be selected based on a request filter. An example HTTP Provider configuration looks like this:

```
<HTTPProvider>
  <BaseClass>com.wowza.wms.http.HTTPServerInfoXML</BaseClass>
  <RequestFilters>serverinfo*</RequestFilters>
  <AuthenticationMethod>admin-digest</AuthenticationMethod>
</HTTPProvider>
```

The **BaseClass** property is the fully qualified path of the class that overrides the **HTTPProvider2Base** class and implements the **IHTTPProvider** interface. The **RequestFilters** property is a pipe-separated (|) list of filters that control when this provider is invoked based on the HTTP request path. For example, the request filter in the previous example is only invoked if the path part of the HTTP request URL starts with **serverinfo** (for example, **http://[wowza-ip-address]:8086/serverinfo**). **AuthenticationMethod** is the authentication method that's used to control access to this HTTP Provider. Valid values are **admin-digest** and **none**. The **admin-digest** authentication method uses digest authentication (a challenge/response system to authenticate users—credentials are never sent in clear text) to control access to the HTTP Provider. User names and passwords for admin-digest authentication are stored in the file **[install-dir]/conf/admin.password**. The **none** method allows all access.

For more information about how to create custom HTTP Providers, see [Extending Wowza Streaming Engine Using Java](#).

Built-in HTTP Providers

The following list describes the built-in HTTP Providers that are found in **VHost.xml**:

- **HTTPClientAccessPolicy** - (com.wowza.wms.http.HTTPClientAccessPolicy)
Delivers the Microsoft Silverlight **clientaccesspolicy.xml** file when present in the **[install-dir]/conf** folder.
- **HTTPConnectionCountsXML** - (com.wowza.wms.http.HTTPConnectionCountsXML)
Returns connection information in XML format and is available through administrative port 8086 (**http://[wowza-ip-address]:8086/connectioncounts**).
- **HTTPConnectionInfo** - (com.wowza.wms.http.HTTPConnectionInfo)
Returns detailed connection information in XML format and is available through administrative port 8086 (**http://[wowza-ip-address]:8086/connectioninfo**).
- **HTTPCrossdomain** - (com.wowza.wms.http.HTTPCrossdomain)
Delivers the Adobe Flash **crossdomain.xml** file when present in **[install-dir]/conf**.

- **HTTPProviderCaptionFile** -
(com.wowza.wms.timedtext.http.HTTPProviderCaptionFile)
Delivers caption files from **[install-dir]/content** when requested by players.
- **HTTPProviderMediaList** - (com.wowza.wms.http.HTTPProviderMediaList)
Dynamic method for generating adaptive bitrate manifests and playlists from SMIL data.
- **HTTPServerInfoXML** - (com.wowza.wms.http.HTTPServerInfoXML)
Returns detailed server and connection information in XML format and is available through administrative port 8086 ([http://\[wowza-ip-address\]:8086/serverinfo](http://[wowza-ip-address]:8086/serverinfo)).
- **HTTPServerVersion** - (com.wowza.wms.http.HTTPServerVersion)
Returns the Wowza Streaming Engine version and build number. It's the default HTTP Provider on port 1935.
- **HTTPTranscoderThumbnail** -
(com.wowza.wms.transcoder.httpprovider.HTTPTranscoderThumbnail)
Returns a bitmap image from the source stream being transcoded. Available through administrative port 8086 ([http://\[wowza-ip-address\]:8086/transcoderthumbnail?application=\[application-name\]&streamname=\[stream-name\]&format=\[jpeg or png\]&size=\[widthxheight\]](http://[wowza-ip-address]:8086/transcoderthumbnail?application=[application-name]&streamname=[stream-name]&format=[jpeg or png]&size=[widthxheight]))

Extending Wowza Streaming Engine Using Java

How do I extend Wowza Streaming Engine?

The Wowza Streaming Engine™ software is built using Java technology and can be extended by writing custom Java classes that are loaded dynamically by the server at runtime. Several integration points can be used to extend the server: custom server-side extensions (also referred to as "modules"), HTTP Providers, and listeners. This chapter explores each of these integration points and provides a quick example.

Wowza Streaming Engine includes a rich set of APIs to interact with and control the streaming process. See the [Wowza Streaming Engine Server-Side API](#) for detailed information about the available APIs. The [Server-Side Modules and Code Samples](#) webpage contains additional knowledge and code snippets.

Before reading this chapter, we recommend that you download and install the free [Wowza Integrated Development Environment \(Wowza IDE\)](#), which is used to extend Wowza Streaming Engine functionality. The Wowza IDE includes documentation that describes how to create your first custom server-side module. It will point you back to this chapter for more information.

Custom Module Classes

Server-side modules are Java classes that are configured on a per-application basis. They are dynamically created at application instance startup and run at the full speed of the server. Typically, module classes are bound to **.jar** files that are located in the Wowza Streaming Engine installation. Modules can leverage third-party libraries or built-in Java functionality if the dependent **.jar** files are copied to the **[install-dir]/lib** folder in the Streaming Engine

installation. Modules are added to an application configuration by adding an entry to the **Modules** list for an application in Wowza Streaming Engine Manager.

Let's start by creating our first module. It'll have two methods: the event method **onAppStart** and the custom method **doSomething** (the details of event methods and custom methods will be discussed later):

```
package com.mycompany.module;

import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

public class MyModule extends ModuleBase
{
    public void onAppStart(IApplicationInstance appInstance)
    {
        getLogger().info("onAppStart");
    }

    public void doSomething(IClient client, RequestFunction function,
        AMFDataList params)
    {
        getLogger().info("doSomething");
    }
}
```

To add this module to an application configuration, go to the application in Wowza Streaming Engine Manager and use the **Add New Module** dialog box to add the following entry to the end of the **Modules** list for the application:

Modules Java classes that extend an application's functionality. The list below defines an order-dependent list of modules to be loaded for a given application. The modules are loaded dynamically when the application instance is loaded. The base (ModuleCore) module must be included by the application for it to operate properly.



Name	Description	Fully Qualified Class Name
base	Base	com.wowza.wms.module.ModuleCore
logging	Client Logging	com.wowza.wms.module.ModuleClientLogging
flvplayback	FLVPlayback	com.wowza.wms.module.ModuleFLVPlayback
ModuleCoreSecurity	Core Security Module for Applications	com.wowza.wms.security.ModuleCoreSecurity
MyModule	This is MyModule	com.mycompany.module.MyModule

Each module must have a unique **Name** in the **Modules** list. The **Description** information provides a detailed description of the module and isn't used in any operations. The **Class** item is the fully qualified path to the Java class that provides the module's functionality. Java combines the package path in the first line of the module to the class name to form the class path.

Event Methods

Event methods are invoked by the server based on events that occur during server processing. Event methods apply to all types of streaming: DASH Streaming, Apple HLS (Cupertino), Adobe HDS (San Jose), Microsoft Smooth Streaming, and RTSP/RTP streaming. Event methods are defined by the following interfaces:

```
IModuleOnApp
IModuleOnApp2
IModuleOnConnect
IModuleOnStream
IModuleOnHTTPSession
IModuleOnRTPSession
IModuleOnHTTPCupertinoStreamingSession
IModuleOnHTTPSmoothStreamingSession
IModuleOnHTTPSanJoseStreamingSession
IModuleOnHTTPMPEGDashStreamingSession
IModuleOnHTTPCupertinoEncryption
IModuleOnHTTPSmoothStreamingPlayReady
IModuleOnHTTPMPEGDashEncryption
```

Event methods that are defined in a module are invoked when an event occurs. If two modules implement the **onAppStart** event method, the **onAppStart** method is invoked for both modules when a new application instance is created. Module methods are invoked in the order in which entries are listed in the **Modules** list for an application in Wowza Streaming Engine Manager. So the first entry in the **Modules** list is called first, the second entry is called next, and so on, down to the last item in the list. The rest of this section describes the event method interfaces and their corresponding methods.

IModuleOnApp

```
public void onAppStart(IApplicationInstance appInstance);
public void onAppStop(IApplicationInstance appInstance);
```

- **onAppStart:** Invoked when an application instance is started.
- **onAppStop:** Invoked when an application instance is stopped.

IModuleOnApp2

This interface extends **IModuleOnApp**, providing two additional methods:

```
public void onAppCreate(IApplicationInstance appInstance);
public void onAppDestroy(IApplicationInstance appInstance);
```

- **onAppCreate:** Invoked when an application instance is created.
- **onAppDestroy:** Invoked when an application instance is destroyed.

IModuleOnConnect

```
public void onConnect(IClient client,
    RequestFunction function, AMFDataList params);
public void onDisconnect(IClient client);
public void onConnectAccept(IClient client);
public void onConnectReject(IClient client);
```

- **onConnect:** Invoked when Flash Player connects to an application instance.
- **onDisconnect:** Invoked when Flash Player disconnects from an application instance.
- **onConnectAccept:** Invoked when a Flash Player connection is accepted.
- **onConnectReject:** Invoked when a Flash Player connection is refused.

IModuleOnStream

```
public void onStreamCreate(IMediaStream stream);
public void onStreamDestroy(IMediaStream stream);
```

- **onStreamCreate:** Invoked when a new **IMediaStream** object is created.
- **onStreamDestroy:** Invoked when an **IMediaStream** object is closed.

Note

The **onStreamCreate** event method is invoked before **play** or **publish** is called for this **IMediaStream** object. For this reason, the **IMediaStream** object doesn't have a name. See [How to use event listeners](#) for more information about how to implement a server listener that's invoked when actions occur on this **IMediaStream** object.

IModuleOnHTTPSession

```
public void onHTTPSessionCreate(IHTTPStreamerSession httpSession);
public void onHTTPSessionDestroy(IHTTPStreamerSession httpSession);
```

- **onHTTPSessionCreate:** Invoked when an Apple HLS (Cupertino) or Smooth Streaming HTTP streaming session is created.
- **onHTTPSessionDestroy:** Invoked when a Cupertino or Smooth Streaming HTTP streaming session is closed.

IModuleOnRTPSession

```
public void onRTPSessionCreate(RTPSession rtpSession);
public void onRTPSessionDestroy(RTPSession rtpSession);
```

- **onRTPSessionCreate:** Invoked when an RTP session is created.
- **onRTPSessionDestroy:** Invoked when an RTP session is closed.

IModuleOnHTTPCupertinoStreamingSession

```
public void onHTTPCupertinoStreamingSessionCreate(
    HTTPStreamerSessionCupertino httpCupertinoStreamingSession);
public void onHTTPCupertinoStreamingSessionDestroy(
    HTTPStreamerSessionCupertino httpCupertinoStreamingSession);
```

- **onHTTPCupertinoStreamingSessionCreate:** Invoked when an Apple HLS (Cupertino) session is created.
- **onHTTPCupertinoStreamingSessionDestroy:** Invoked when a Cupertino session is closed.

IModuleOnHTTPSmoothStreamingSession

```
public void onHTTPSmoothStreamingSessionCreate(
    HTTPStreamerSessionSmoothStreamer httpSmoothStreamingSession);
public void onHTTPSmoothStreamingSessionDestroy(
    HTTPStreamerSessionSmoothStreamer httpSmoothStreamingSession);
```

- **onHTTPSmoothStreamingSessionCreate:** Invoked when a Smooth Streaming session is created.
- **onHTTPSmoothStreamingSessionDestroy:** Invoked when a Smooth Streaming session is closed.

IModuleOnHTTPSanJoseStreamingSession

```
public void onHTTPSanJoseStreamingSessionCreate(
    HTTPStreamerSessionSanJoseStreamer httpSanJoseStreamingSession);
public void onHTTPSanJoseStreamingSessionDestroy(
    HTTPStreamerSessionSanJoseStreamer httpSanJoseStreamingSession);
```

- **onHTTPSanJoseStreamingSessionCreate:** Invoked when an Adobe HDS (San Jose) session is created.
- **onHTTPSanJoseStreamingSessionDestroy:** Invoked when a San Jose session is closed.

IModuleOnHTTPMPEGDashStreamingSession

```
public void onHTTPMPEGDashStreamingSessionCreate(
    HTTPStreamerSessionMPEGDash httpMPEGDashStreamingSession);
public void onHTTPMPEGDashStreamingSessionDestroy(
    HTTPStreamerSessionMPEGDash httpMPEGDashStreamingSession);
```

- **onHTTPMPEGDashStreamingSessionCreate:** Invoked when an MPEG-DASH session is created.
- **onHTTPMPEGDashStreamingSessionDestroy:** Invoked when an MPEG-DASH session is closed.

IModuleOnHTTPCupertinoEncryption

```
public void onHTTPCupertinoEncryptionKeyRequest(
    HTTPStreamerSessionCupertino httpSession, IHTTPRequest req,
    IHTTPResponse resp);
public void onHTTPCupertinoEncryptionKeyCreateVOD(
    HTTPStreamerSessionCupertino httpSession, byte[] encKey);
public void onHTTPCupertinoEncryptionKeyCreateLive(
    IApplicationInstance appInstance, String streamName, byte[] encKey);
```

- **onHTTPCupertinoEncryptionKeyRequest:** Invoked when an encryption key request is made for Apple HLS (Cupertino) streaming.
- **onHTTPCupertinoEncryptionKeyCreateVOD:** Invoked when an encryption key is created for a Cupertino video on demand stream.
- **onHTTPCupertinoEncryptionKeyCreateLive:** Invoked when an encryption key is created for a Cupertino live stream.

IModuleOnHTTPSmoothStreamingPlayReady

```
public void onHTTPSmoothStreamingPlayReadyCreateVOD(
    HTTPStreamerSessionSmoothStreamer httpSession,
    PlayReadyKeyInfo playReadyKeyInfo);
public void onHTTPSmoothStreamingPlayReadyCreateLive(
    IApplicationInstance appInstance, String streamName,
    PlayReadyKeyInfo playReadyKeyInfo);
```

- **onHTTPSmoothStreamingPlayReadyCreateVOD**: Invoked when an encryption key request is made for Smooth Streaming video on demand.
- **onHTTPSmoothStreamingPlayReadyCreateLive**: Invoked when an encryption key request is made for Smooth Streaming live.

IModuleOnHTTPMPEGDashEncryption

```
public void onHTTPMPEGDashEncryptionKeyVODChunk(
    HTTPStreamerSessionMPEGDash httpSession,
    IHTTPStreamerMPEGDashIndex index, CencInfo cencInfo, long chunkId);
public void onHTTPMPEGDashEncryptionKeyLiveChunk(
    ILiveStreamPacketizer liveStreamPacketizer,
    String streamName, CencInfo cencInfo, long chunkId);
```

- **onHTTPMPEGDashEncryptionKeyVODChunk**: Invoked when an encryption key request is made for MPEG-DASH video on demand.
- **onHTTPMPEGDashEncryptionKeyLiveChunk**: Invoked when an encryption key request is made for MPEG-DASH live.

Custom Methods

You can expose public custom methods to Adobe Flash Player through calls to the client-side interface **NetConnection.call()** or in calls that are part of the **NetConnection** or **NetStream** command set. For example, **play** and **publish** are defined in **ModuleCore** as custom methods. These methods must be public and must have the argument signature (**IClient, RequestFunction, AMFDataList params**). Only public methods with this signature are available to be called from Flash Player.

Custom methods are processed differently than event methods. When a method is invoked from Flash Player, only the last module in the application's **Modules** list that defines that custom method is invoked. For example, the **ModuleCore** module defines the method **play**, which is invoked when **NetStream.play(streamName)** is called from Flash Player. If you create your own custom module that defines the method **play** and add it to the **Modules** list after the **ModuleCore** module, then your **play** method is invoked instead of the **play** method that's defined in **ModuleCore**. In your implementation of **play**, if you want to invoke the **play** method of the module that precedes your module in the **Modules** list, you call **this.invokePrevious(client, function, params)**. Wowza Streaming Engine will search upwards in the **Modules** list, find the next module that implements the **play** method, and invoke that method. This is similar to traditional object-orientated sub-classing. Each implementation of

a method in the **Modules** list can perform an operation based on the invocation of a given method and can choose to pass control to the next module above them in the **Modules** list that implements the method.

For example, assume that you want to check the stream name of calls made to **NetStream.play(streamName)** in your implementation of **play**. If the stream name starts with **goodstream/**, you want to append the phrase **_good** to the stream name and then call **this.invokePrevious(client, function, params)**. All other connections will be disconnected. The code looks like this:

```
package com.mycompany.module;
import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

public class MyModule extends ModuleBase
{
    public void play(IClient client, RequestFunction function, AMFDataList
params)
    {
        boolean disconnect = false;
        if (params.get(PARAM1).getType() == AMFData.DATA_TYPE_STRING)
        {
            String playName = params.getString(PARAM1);
            if (playName.startsWith("goodstream/"))
            {
                playName += "_good";
                params.set(PARAM1, new AMFDataItem(playName));
            }
            else
            {
                disconnect = true;
            }
        }

        if (disconnect)
            client.setShutdownClient(true);
        else
            this.invokePrevious(client, function, params);
    }
}
```

onCall Method

The **onCall** method is a catch-all for methods that aren't defined by custom methods. The **IModuleOnCall** interface class defines the interface for this method. The **onCall** method

works just like an event method in that all **onCall** methods that are defined in all modules are called. For example:

```
package com.mycompany.module;

import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

public class MyModule extends ModuleBase implements IModuleOnCall
{
    public void onCall(String handlerName, IClient client,
RequestFunction function, AMFDataList params)
    {
        getLogger().info("onCall: "+handlerName);
    }
}
```

Adobe Flash Player and Custom Methods

Parameters passed from Adobe Flash Player to Wowza Streaming Engine must be marshaled to Java primitive and object types. The **com.wowza.wms.module.ModuleBase** class includes helper functions and constants for converting the parameter values. For more complex types, the **com.wowza.wms.amf** package contains an API for object conversion. For more information, see the [Wowza Streaming Engine Server-Side API](#) and the **Server Side Coding** example (`[install-dir]/examples/ServerSideModules`).

The following example shows how to convert three incoming parameters:

```
package com.mycompany.module;

import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

public class MyModule extends ModuleBase
{
    public void myFunction(IClient client,
RequestFunction function, AMFDataList params)
    {
        String param1 = getParamString(params, PARAM1);
        int param2 = getParamInt(params, PARAM2);
        boolean param3 = getParamBoolean(params, PARAM3);
    }
}
```

A custom method called from Flash Player may return a single result value, which must be converted to an Action Message Format (AMF) object in order to be understood by Flash Player. These value types can include simple types like strings, integers, and Booleans and more complex types like objects, arrays, or arrays of objects. The **com.wowza.wms.module.ModuleBase** class includes helper functions for returning simple types. For more complex types, the **com.wowza.wms.amf** package contains an API for object creation and conversion. For more information, see the [Wowza Streaming Engine Server-Side API](#) and the **Server Side Coding** example (`[install-dir]/examples/ServerSideModules`).

The following example shows how to return simple value types from three methods:

```
package com.mycompany.module;
import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

public class MyModule extends ModuleBase
{
    public void myFunctionString(IClient client,
        RequestFunction function, AMFDataList params)
    {
        sendResult(client, params, "Hello World");
    }

    public void myFunctionInt(IClient client,
        RequestFunction function, AMFDataList params)
    {
        sendResult(client, params, 536);
    }

    public void myFunctionBoolean(IClient client,
        RequestFunction function, AMFDataList params)
    {
        sendResult(client, params, true);
    }
}
```

Adobe Flash Player and Server-to-Client Calls

A custom method can call a function in Adobe Flash Player directly by invoking the **IClient.call()** method. The client call can return a single variable that will be received by the server by creating a result object that implements the

com.mycompany.module.IModuleCallResult interface. The **IClient.call()** method has two forms:

```
public abstract void call(String handlerName);
public abstract void call(String handlerName,
    IModuleCallResult resultObj, Object ... params);
```

Methods on the client-side are made available to the server by attaching them to the **NetConnection** object. The following is sample ActionScript 3.0 client-side code:

```
var nc:NetConnection = new NetConnection();
var clientObj:Object = new Object();

clientObj.serverToClientMethod = function(param1, param2)
{
    return "Hello World";
}

nc.client = clientObj;
nc.connect("rtmp://wms.mycompany.com/mymodules");
```

To call this client-side method from the server, the custom method looks like this:

```
package com.mycompany.module;

import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.amf.*;
import com.wowza.wms.request.*;

class MyResult implements IModuleCallResult
{
    public onResult(IClient client,
        RequestFunction function, AMFDataList params)
    {
        String returnValue = getParamString(params, PARAM1);
        getLogger().info("got Result: "+ returnValue);
    }
}

public class MyModule extends ModuleBase
{
    public void myFunction(IClient client,
        RequestFunction function, AMFDataList params)
    {
        client.call("serverToClientMethod", new MyResult(),
            "param1: value", 1.5);
    }
}
```

Logging

A custom method can get access to the server's logging interface using the **getLogger()** helper method that's implemented by the **com.wowza.wms.module.ModuleBase** base class. Log messages are written to the log files by using one of the following methods:

```
getLogger().debug(logStr);
getLogger().info(logStr);
getLogger().warn(logStr);
getLogger().error(logStr);
```

HTTP Provider Classes

HTTP Providers are Java classes that are mini Java servlets that can be used to add an HTTP interface to Wowza Streaming Engine. They are configured on a per-port basis in **[install-dir]/conf/VHost.xml**. (See [Server-Side Modules and HTTP Providers](#) in this document.) The following example shows a simple HTTP Provider that returns the server version:

```
package com.mycompany.wms.http;
import java.io.*;

import com.wowza.wms.server.*;
import com.wowza.wms.stream.*;
import com.wowza.wms.vhost.*;
import com.wowza.wms.http.*;

public class HTTPServerVersion extends HTTPProvider2Base
{
    public void onHTTPRequest(IVHost vhost, IHTTPRequest req, IHTTPResponse
resp)
    {
        if (!doHTTPAuthentication(vhost, req, resp))
            return;

        String version = MediaStreamBase.p+" ";
        version += ReleaseInfo.getVersion();
        version += " build"+ReleaseInfo.getBuildNumber();

        String retStr = "<html><head><title>";
        retStr += version;
        retStr += "</title></head><body>"+version+"</body></html>";
        try
        {
            OutputStream out = resp.getOutputStream();
            byte[] outBytes = retStr.getBytes();
            out.write(outBytes);
        }
    }
}
```

```

        catch (Exception e)
        {
            System.out.println("HTMLServerVersion: "+e.toString());
        }
    }
}

```

Much of the functionality of HTTP Providers is encapsulated in the **HTTPProvider2Base** base class. Your HTTP Provider, if it extends this class, only needs to implement the **onHTTPRequest** method. The following are some interesting code snippets to aid in HTTP Provider development:

Get HTTP request URL

```
String path = super.getPath(req, false);
```

Get HTTP request header value

```
String headerValue = req.getHeader(headerName);
```

Set HTTP response header value

```
resp.setHeader(headerName, headerValue);
```

Set HTTP response status

```
resp.setResponseCode(404);
```

More complex and interesting HTTP Providers examples can be found on the [HTTP Providers](#) webpage.

Event Listeners

You can add event listeners to many points in the Wowza Streaming Engine object hierarchy. Event listeners are classes that implement a notifier interface and are notified of specific events within the server. For example, you can inject a server listener that gets notified of server startup, initialization, and shutdown or an application instance listener that gets notified when an application instance is started or stopped. For more information, see [How to use event listeners](#).

Server Administration

How do I configure, manage, and deploy Wowza Streaming Engine?

Wowza Streaming Engine™ is a powerful Java server. It can be run standalone from a command shell or installed as a system service. Running the server standalone is best for developing custom Wowza Streaming Engine applications because the server can be started and stopped quickly and server log messages can be viewed immediately in the console window. Running the server as a system service is more often used for server deployments where the server must continue to run after you log off the computer or must be automatically started when the computer is rebooted.

Configuring SSL and RTMPS

Wowza Streaming Engine supports Secure Sockets Layer (SSL) and RTMPS (RTMP over SSL) and HTTPS (HTTP over SSL) streaming protection. SSL is a technology that allows web browsers and web servers to communicate over a secure connection, with the encrypted data being sent and received in both directions. You can use Wowza StreamLock™ AddOn to get a free 256-bit SSL certificate, you can get an SSL certificate from a certificate authority, or you can create a certificate yourself (a self-signed SSL certificate).

Notes

- If you want to get an SSL certificate from Wowza for use with Wowza Streaming Engine, see [How to get SSL certificates from the StreamLock service](#).
- If you want to get an SSL certificate from a certificate authority, see [How to request an SSL certificate from a certificate authority](#).
- If you want to create a self-signed SSL certificate, see [How to create a self-signed SSL certificate](#).

Logging

Wowza Streaming Engine uses the Apache log4j logging utility as its logging implementation. The log4j logging system provides ample functionality for log formatting, log rolling, and log retrieval for most applications. By default, the Streaming Engine is configured to log basic information to the server console and detailed information in the W3C Extended Common Log Format (ECLF) to a log file. Java messaging is also captured in the log files to help monitor and aid troubleshooting. The log files are written to the **[install-dir]/logs** folder.

For more information about log messages, scenarios that may cause these messages, and suggestions for resolution, see [How to troubleshoot error messages](#).

Logging Fields

Wowza Streaming Engine can generate the following logging fields.

Field name	Description
c-client-id	Client ID number assigned by the server to the connection
c-ip	Client connection IP address
c-proto	Client connection protocol: http (Apple HLS), http (Smooth Streaming), rtmp, rtmpe, rtmps (HTTP-1.1), rtmpt (HTTP-1.1), rtmpte (HTTP-1.1)
c-referrer	URL of the Flash movie that initiated the connection to the server
c-user-agent	Version of the Flash client that initiated the connection to the server
cs-bytes	Total number of bytes transferred from client to server (cumulative)
cs-stream-bytes	Total number of bytes transferred from client to server for stream x-stream-id (cumulative)
cs-uri-query	Query parameter for stream x-stream-id
cs-uri-stem	Full connection string for stream x-stream-id (excludes query parameters)
date	Date of log event
s-ip	IP address of the server that received this event

s-port	Port number through which the server received this event
s-uri	Full connection string on which the server received this event
sc-bytes	Total number of bytes transferred from server to client (cumulative)
sc-stream-bytes	Total number of bytes transferred from server to client for stream x-stream-id (cumulative)
time	Time of log event
tz	Time zone of log event
x-app	Name of the application from which the event was generated
x-appinst	Name of the application instance from which the event was generated
x-category	Log event category (server, vhost, application, session, stream)
x-comment	Extra comment about the log event
x-ctx	Extra data about the context of the log event
x-duration	Time, in seconds, that this event occurred within the lifetime of the x-category object
x-event	Log event (see the Logging Events section of this document)
x-file-ext	File extension of stream x-stream-id
x-file-length	File length, in seconds, of stream x-stream-id
x-file-name	Full file path of stream x-stream-id
x-file-size	File size, in bytes, of stream x-stream-id
x-severity	Log event severity (DEBUG, INFO, WARN, ERROR, FATAL)
x-sname	Name of stream x-stream-id
x-sname-query	Query parameters of stream x-stream-id
x-spos	Position, in milliseconds, within the media stream
x-status	Log event status (see the Logging Status Values section of this document)
x-stream-id	Stream ID number assigned by the server to the stream object

x-suri	Full connection string for stream x-stream-id (includes query parameters)
x-suri-query	Query parameter for connection string
x-suri-stem	Full connection string for stream x-stream-id (excludes query parameters)
x-vhost	Name of the virtual host from which the event was generated

Logging Events

Wowza Streaming Engine can generate the following logging events.

Event name	Description
announce	RTSP Session Description Protocol (SDP) ANNOUNCE
app-start	Application instance start
app-stop	Application instance shutdown
comment	Comment
connect	Connection result
connect-burst	Connection accepted in burst zone
connect-pending	Connection pending approval by application and license manager
create	Media or data stream created
decoder-audio-start	Audio decoding has started for a transcoded stream
decoder-audio-stop	Audio decoding has stopped for a transcoded stream
decoder-video-start	Video decoding has started for a transcoded stream
decoder-video-stop	Video decoding has stopped for a transcoded stream
destroy	Media or data stream destroyed
disconnect	Client (session) disconnected from server
encoder-audio-start	Audio encoding has started for a transcoded stream
encoder-audio-stop	Audio encoding has stopped for a transcoded stream
encoder-video-start	Video encoding has started for a transcoded stream

encoder-video-stop	Video encoding has stopped for a transcoded stream
pause	Playback has paused
play	Playback has started
publish	Start stream publishing
record	Start stream recording
recordstop	Stop stream recording
seek	Seek has occurred
setbuffertime	Client call to NetStream.setBufferTime(secs) logged in milliseconds
setstreamtype	Client call to netConnection.call("setStreamType", null, "[streamtype]")
server-start	Server start
server-stop	Server shutdown
stop	Playback has stopped
unpause	Playback has resumed from pause
unpublish	Stop stream publishing
vhost-start	Virtual host start
vhost-stop	Virtual host shutdown

Logging Status Values

Wowza Streaming Engine can generate the following logging status values.

Status value	Description
100	Pending or waiting (for approval)
200	Success
302	Rejected by application with redirect information
400	Bad request
401	Rejected by application

413	Rejected by license manager
500	Internal error

Logging Configuration

Logging for Wowza Streaming Engine is configured in the **conf/log4j.properties** properties file. The log4j logging system has many logging configuration options. This section covers the basic options for enabling and disabling different logging fields, events, and categories.

The following is an example of a basic **log4j.properties** file for Wowza Streaming Engine:

```
log4j.rootCategory=INFO, stdout, serverAccess, serverError

# Console appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=com.wowza.wms.logging.ECLFPatternLayout
log4j.appender.stdout.layout.Fields=x-severity,x-category,x-event,x-ctx,x-
comment
log4j.appender.stdout.layout.OutputHeader=false
log4j.appender.stdout.layoutQuoteFields=false
log4j.appender.stdout.layout.Delimiter=space

# Access appender
log4j.appender.serverAccess=org.apache.log4j.DailyRollingFileAppender
log4j.appender.serverAccess.DatePattern='.'yyyy-MM-dd
log4j.appender.serverAccess.File=${com.wowza.wms.ConfigHome}/logs/wowzastre
amingengine_access.log
log4j.appender.serverAccess.layout=com.wowza.wms.logging.ECLFPatternLayout
log4j.appender.serverAccess.layout.Fields=x-severity,x-category,x-
event;date,time,c-client-id,c-ip,c-port,cs-bytes,sc-bytes,x-duration,x-
sname,x-stream-id,sc-stream-bytes,cs-stream-bytes,x-file-size,x-file-
length,x-ctx,x-comment
log4j.appender.serverAccess.layout.OutputHeader=true
log4j.appender.serverAccess.layoutQuoteFields=false
log4j.appender.serverAccess.layout.Delimiter=tab

# Error appender
log4j.appender.serverError=org.apache.log4j.DailyRollingFileAppender
log4j.appender.serverError.DatePattern='.'yyyy-MM-dd
log4j.appender.serverError.File=${com.wowza.wms.ConfigHome}/logs/wowzastre
amingengine_error.log
log4j.appender.serverError.layout=com.wowza.wms.logging.ECLFPatternLayout
log4j.appender.serverError.layout.Fields=x-severity,x-category,x-
event;date,time,c-client-id,c-ip,c-port,cs-bytes,sc-bytes,x-duration,x-
sname,x-stream-id,sc-stream-bytes,cs-stream-bytes,x-file-size,x-file-
length,x-ctx,x-comment
```

```
log4j.appender.serverError.layout.OutputHeader=true
log4j.appender.serverError.layout QuoteFields=false
log4j.appender.serverError.layout.Delimiter=tab
log4j.appender.serverError.Threshold=WARN
```

Note

Always use forward slashes when referring to file paths (even on the Windows platform).

The first statement in the **log4j.properties** file sets the logging level to INFO and defines three appenders: stdout, serverAccess, and serverError. Setting the logging level to INFO configures the logging mechanism such that it only logs events with a severity of INFO or higher. The logging severity in ascending order is: DEBUG, INFO, WARN, ERROR, and FATAL. To log all events, set the logging level to DEBUG.

Appender properties allow you to control the way that log information is formatted and filtered. The following table shows some of the important properties.

Property name	Description
CategoryExclude	Comma-separated list of logging categories. Only log events whose category isn't in this list are logged.
CategoryInclude	Comma-separated list of logging categories. Only log events with the specified categories are logged.
Delimiter	The delimiter character to use between field values. Valid values are tab , space , or the actual delimiter character.
EventExclude	Comma-separated list of logging categories. Only log events whose event name isn't in this list are logged.
EventInclude	Comma-separated list of logging events. Only log events with the specified event name are logged.
Field	Comma-delimited list of fields to log.
OutputHeader	Boolean value (true/false) that instructs the logging system to write out a W3C ECLF header whenever the server is started.
QuoteFields	Boolean value (true/false) that instructs the logging system to wrap field data in double quotes.

For more information about how to configure the log4j specific properties such as log file rolling and additional log appender types, see the [Log4j website](#).

Wowza Streaming Engine can also be configured to generate logs on a per-application and per-virtual host basis. These configurations are included, but commented-out, at the bottom of the **default [install-dir]/conf/log4j.properties** file. The first commented-out section includes configuration for per-application logging. The second commented-out section includes configuration for per-virtual host logging. To enable either of these features, remove the comments (# sign at the beginning of each of the lines) from the section.

The per-application logging generates log files using the following directory structure:

```
[install-dir]/logs/[vhost]/[application]/wowzastreamingengine_access.log  
[install-dir]/logs/[vhost]/[application]/wowzastreamingengine_error.log  
[install-dir]/logs/[vhost]/[application]/wowzastreamingengine_stats.log
```

The per-virtual host logging generates log files using the following directory structure:

```
[install-dir]/logs/[vhost]/wowzastreamingengine_access.log  
[install-dir]/logs/[vhost]/wowzastreamingengine_error.log  
[install-dir]/logs/[vhost]/wowzastreamingengine_stats.log
```

This method for generating log files can be very useful if you want to offer Wowza Streaming Engine as a shared service to several customers.

Streaming Tutorials

Where do I get step-by-step instructions?

The support [Tutorials](#) section of the Wowza® website contains tutorials with step-by-step instructions for configuring common streaming scenarios. These instructions cover how to configure streaming to common player technologies such as Adobe Flash Player, Microsoft Silverlight, Apple iOS devices, and mobile devices. The following table briefly describes and provides a link to an online tutorial for common streaming scenarios.

Tutorial name	Description
How to set up video on demand streaming	Describes how to configure an application to stream video on demand (VOD) content to all supported player technologies.
How to set up live streaming using an RTMP-based encoder	Describes how to publish a live stream from RTMP-based encoders to Wowza Streaming Engine™ and how to configure an application to deliver the live stream to all supported player technologies.
How to set up live streaming using an RTSP/RTP-based encoder	Describes how to publish a live stream from RTSP/RTP-based encoders to Wowza Streaming Engine and how to configure an application to deliver the live stream to all supported player technologies.
How to set up live streaming using a native RTP encoder with SDP file	Describes how to use a live encoder that publishes a stream using Real-time Transport Protocol (native RTP) with Session Description Protocol (SDP) files to stream live content to Flash

	Player, Silverlight, iOS devices, and RTSP/RTP based players.
How to publish and play a live stream (MPEG-TS based encoder)	Describes how to use a live encoder that publishes a stream using the MPEG-2 Transport Stream (MPEG-2 TS) protocol to stream live content to Flash Player, Silverlight, iOS devices, and RTSP/RTP based players and devices.
How to set up and run Wowza Transcoder AddOn for live streaming	Describes how to configure an application to use Wowza Transcoder AddOn.
How to set up and run Wowza nDVR for live streaming	Describes how to configure an application to use Wowza nDVR AddOn.
How to configure a live stream repeater	Describes how to configure live stream repeater (origin/edge) applications. Live stream repeater is a method for delivering a single live stream across a multiple server deployment to many viewers.
How to re-stream video from an IP camera	Describes how to re-stream and play a live stream from an IP camera.
How to re-stream audio from SHOUTcast/Icecast	Describes how to re-stream and play live SHOUTcast or Icecast audio streams.
How to set up live video recording	Describes how to configure an application for video recording using Flash Player.
How to set up live video chat	Describes how to configure an application for video chat using Flash Player.
Closed Captioning Overview	This webpage has tutorials that describe how to ingest caption data from a variety of instream and file-based sources and convert it to the appropriate caption format for video on demand and live streaming using the Apple HLS, Adobe HDS, and RTMP protocols.
How to do MPEG-DASH streaming	Describes how to configure video on-demand and live applications in Wowza Streaming Engine to deliver streams to Dynamic Adaptive Streaming over HTTP (DASH) clients.
Wowza DRM AddOn Overview	This webpage has tutorials that describe how to provide simultaneous secure key exchange with

	different DRM platforms and how to encrypt live and video on demand content on the fly for Apple HLS and Microsoft Smooth Streaming delivery to a wide range of devices including set-top boxes (STBs), smart TVs, and smartphones and tablets.
Media Security Overview	This webpage has tutorials that describe the different methods that you can use to help ensure a more secure stream when delivering content using Apple HLS, Adobe HDS, and Microsoft Smooth Streaming.