Wowza Media Server 3:
User's Guide

Version: 3.6.4

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

**Third Party Information**

This document contains links to third party websites that are not under the control of Wowza Media Systems, LLC ("Wowza") and Wowza is not responsible for the content on any linked site. If you access a third party website mentioned in this document, then you do so at your own risk. Wowza provides these links only as a convenience, and the inclusion of any link does not imply that Wowza endorses or accepts any responsibility for the content on third party sites.

This document refers to third party software that is not licensed, sold, distributed or otherwise endorsed by Wowza. Please ensure that any and all use of Wowza® software and third party software is properly licensed.

**Trademarks**

Wowza, Wowza Media Systems, Wowza Media Server and related logos are either registered trademarks or trademarks of Wowza Media Systems, LLC in the United States and/or other countries.

Adobe and Flash are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Microsoft and Silverlight are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

QuickTime, iPhone, iPad and iPod touch are either registered trademarks or trademarks of Apple, Inc. in the United States and/or other countries.

Other product names, logos, designs, titles, words or phrases mentioned may be third party registered trademarks or trademarks in the United States and/or other countries.

Third party trademarks are used solely to identify and describe third party products as being compatible with Wowza products. Wowza is in no way sponsored, endorsed by or otherwise affiliated with any such third party trademark owners.

**Third Party Copyright Notices**

Apache Commons Lang: Copyright © 2001-2011, The Apache Software Foundation

Apache Commons Modeler Component: Copyright © 2002-2008, The Apache Software Foundation
Bouncy Castle Crypto APIs: Copyright © 2000 – 2008, Legion of the Bouncy Castle

Jackson JSON Parser: Copyright © 2009, FasterXML, LLC


Libgcc_s-4 library and Libstdc++ library: Copyright © 2011, Free Software Foundation, Inc.

LibVA libraries: Copyright © 2007, Intel Corporation. All rights reserved.

Log4j and Mina: Copyright © 2006, The Apache Software Foundation

NVIDIA Video Codec SDK: Copyright © 2013, NVIDIA Corporation

Silver Egg Technology: Copyright © 2001, Silver Egg Technology

Speex Codec: Copyright © 2002-2003, Jean-Marc Valin/Xiph.Org Foundation

VideoEncoderH264VAAPImpl: Copyright © 2012, Intel Corporation. All Rights Reserved.

Vorbis/Ogg libraries: Copyright © 2011, Xiph.Org Foundation

WebM VP8 Codec libraries: Copyright © 2010, Google Inc. All rights reserved.
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Release date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc v3.5.0</td>
<td>Initial document release for Wowza Media Server 3.5</td>
<td>11-08-2012</td>
</tr>
<tr>
<td>Doc v3.6.0</td>
<td>Updated for Wowza Media Server 3.6.0</td>
<td>05-28-2013</td>
</tr>
<tr>
<td>Doc v3.6.1</td>
<td>Updated for Wowza Media Server 3.6.1</td>
<td>05-29-2013</td>
</tr>
<tr>
<td>Doc v3.6.2</td>
<td>Updated for Wowza Media Server 3.6.2</td>
<td>06-06-2013</td>
</tr>
<tr>
<td>Doc v3.6.3</td>
<td>Updated for Wowza Media Server 3.6.3</td>
<td>11-01-2013</td>
</tr>
<tr>
<td>Doc v3.6.4</td>
<td>Updated for Wowza Media Server 3.6.4</td>
<td>01-03-2014</td>
</tr>
</tbody>
</table>

### 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
  MPEG-DASH Technology Preview .................................................................5
  Closed Captioning .........................................................................................5
  Mobile Encoding .........................................................................................6
  NVIDIA NVENC Accelerated Encoding ..................................................6
  Verimatrix KMS with PlayReady Encryption ........................................6
  Live Stream Record ...................................................................................7
  AC-3 Passthrough .......................................................................................7
  New features and functionality in Wowza Media Server 3.5 ..................7

Introduction ...................................................................................................................11
  Adobe HDS (Adobe Flash Player) .................................................................11
  Real Time Messaging Protocol (Adobe Flash Player) ..........................12
  Apple HLS (iPhone, iPad, iPod touch, QuickTime, and more) ............13
  Microsoft Smooth Streaming (Microsoft Silverlight and More) ........14
  MPEG-DASH Streaming (DASH Clients) ....................................................14
  Real Time Streaming Protocols (QuickTime, VLC, 3GPP Devices, Set-top Boxes, and More) .................................................................15
  Video and Audio Streaming, Recording, and Chat ...........................16
  Wowza Transcoder AddOn .......................................................................16
  Wowza nDVR AddOn ................................................................................17
  Wowza DRM AddOn ................................................................................17
  Server Architecture ....................................................................................18
  Wowza Media Server Editions ..................................................................19

Server Installation .........................................................................................................21
  Before Installation .......................................................................................21
  Installing the Server ....................................................................................22
  Starting and Stopping the Server ..............................................................25
  Entering a New License Key ......................................................................29
  Ports Used for Streaming ........................................................................29
  Server Configuration and Tuning ..............................................................30
  Run Server as Named User ......................................................................31
  Upgrading from a Previous Release ..........................................................31
  Co-Existence of Multiple Wowza Media Server Instances ..................32
  Patch Updates .............................................................................................32

Application Configuration .........................................................................................33
  Applications and Application Instances (Application.xml) ..............33
  URL Formats .............................................................................................34
  Stream Types ............................................................................................35
  HTTP Streamers and Live Stream Packetizers ..................................36
  Wowza Timed Text Configuration ............................................................38
  Wowza Transcoder AddOn and Wowza nDVR AddOn Configurations .38
  Modules .......................................................................................................38
  Properties .....................................................................................................39
  Media Types ..................................................................................................40
  Content Storage ............................................................................................41

Advanced Configuration Topics ..............................................................................42
  MediaCasters, Stream Manager, and StartupStreams.xml ..................42
  Live Stream Repeater (Origin/Edge Live Streaming) .........................44
  Live Stream Recording ..............................................................................47
  Server-side Publishing (Stream and Publisher classes) .....................47

Adobe Flash Streaming and Scripting .................................................................48
  Streaming Basics .......................................................................................48
  Pre-built Media Players .............................................................................49
Bi-directional Remote Procedure Calls .......................................................... 50
Remote Shared Objects ................................................................................... 51

**Server-side Modules and HTTP Providers** .................................................. 53
  Server-side Modules ..................................................................................... 53
  HTTP Providers ........................................................................................... 55

**Extending Wowza Media Server Using Java** ................................................. 58
  Custom Module Classes ................................................................................ 58
  HTTP Provider Classes ................................................................................. 67
  Event Listeners ............................................................................................ 68

**Server Administration** .................................................................................. 69
  Configuring SSL and RTMPS ........................................................................ 69
  Logging ......................................................................................................... 70

**Server Management Console and Monitoring** ............................................ 75
  Local Management Using JConsole .............................................................. 75
  Remote JMX Interface Configuration .......................................................... 76
  Remote Management ..................................................................................... 79
  Object Overview ........................................................................................... 79

**Virtual Hosting** ............................................................................................. 82
  Configuration Files ......................................................................................... 82
  Typical Configuration ..................................................................................... 82

**Examples & AddOn Packages** ..................................................................... 86
  Examples ....................................................................................................... 86
  AddOn Packages ........................................................................................... 87

**Streaming Tutorials** ...................................................................................... 92
What's New

Wowza Media Server® 3.6 continues to future-proof and simplify online video delivery by expanding its ability to stream video to any screen using any streaming protocol, augmenting its options and improving the workflow for delivering closed captions, and providing enhanced scalability, flexibility, and reliability. Wowza Media Server 3.6 includes new features and enhancements to features that were introduced in Wowza Media Server 3.5 and earlier versions. Additional functionality and services have also been made available to work exclusively with Wowza Media Server 3.6.

What's new in Wowza Media Server 3.6?

MPEG-DASH Technology Preview

Wowza Media Server 3.6 adds basic support for Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, to its existing suite of supported streaming formats. MPEG-DASH is a new international standard for adaptive streaming that’s being adopted by the streaming industry. Wowza has tested its MPEG-DASH support using several DASH clients. For more information about MPEG-DASH support in Wowza Media Server 3.6, see the MPEG-DASH Overview.

Closed Captioning

Wowza Media Server 3.6 includes expanded support for closed captioning formats for live and video on-demand streams, which were first introduced in Wowza Media Server 3.5. Wowza Media Server 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 Apple® HTTP Live Streaming (Apple HLS), Adobe® HTTP Dynamic Streaming (Adobe HDS), and RTMP 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.
For live streams, Wowza Media Server can ingest instream closed caption information from Action Message Format (AMF) `onTextData` events or CEA-608 data. These ingested captions can be delivered in Adobe HDS and RTMP streams as AMF `onTextData` events, in Apple HLS streams as CEA-608 captions, or in Apple HLS streams for Apple iOS devices (iOS 6 or later) as Web Video Text Tracks (WebVTT) subtitles. In addition, instream CEA-608 caption data can be passed through Wowza Transcoder AddOn for delivery in Apple HLS streams to iOS-based devices.

For video on-demand streams, Wowza Media Server 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). These ingested captions can be delivered in Adobe HDS and RTMP streams as AMF `onTextData` events, in Apple HLS streams as CEA-608 captions, or in Apple HLS streams for Apple iOS devices (iOS 6 or later) as WebVTT subtitles. Delivery of VOD sidecar files is also supported.

For more information, see the Closed Captioning Overview.

**Mobile Encoding**

The Wowza® GoCoder™ app is a live audio and video application for Apple iPod touch® (iOS 4.0 or later) and iPhone® (3GS or later) that allows content providers to encode live content right from their iOS device and deliver it to Wowza Media Server in real time over 4G, 3G, and Wi-Fi systems. For more information, see How to use Wowza GoCoder video broadcasting iOS app with Wowza Media Server.

**NVIDIA NVENC Accelerated Encoding**

Wowza Media Server 3.6 leverages the NVIDIA® Encoder (NVENC) API to access the high-performance H.264 hardware video encoder introduced in graphics cards with the new Kepler GPU architecture. NVENC-based video encoding is faster and consumes less power than legacy CUDA-based or CPU-based encoding, enabling accelerated transcoding with Wowza Transcoder AddOn at greater scale and speed. Support for NVIDIA NVENC accelerated encoding is included in Wowza Media Server 3.6 for 64-bit Windows® and Linux operating systems. For more information, see Server specifications for NVIDIA NVENC and NVIDIA CUDA acceleration with Wowza Transcoder AddOn.

**Verimatrix KMS with PlayReady Encryption**

Wowza Media Server 3.6 integrates with Verimatrix® Key Management Services (KMS) for Microsoft Smooth Streaming. This allows studio-grade Microsoft® PlayReady® DRM to be applied to Smooth Streams that are delivered to a wide variety of Smooth Streaming clients.
This bolsters existing support for Verimatrix VCAS™ protected Apple HLS playback with ViewRight® clients on iOS-based devices, Android™ devices, PCs, and set-top boxes. For more information, see How to set up Verimatrix DRM.

**Live Stream Record**

Enhancements to the Live Stream Record feature in Wowza Media Server 3.6 give you more flexibility and control over how live streams are recorded by providing options to record live streams at both the server and application level and by enabling support for property overrides. Live Stream Record, formerly a free AddOn, was incorporated into Wowza Media Server 3.5 as an API that gives you more control over recording live streams to video on demand files. It enables you to control the live stream recording process using a web-based user interface, HTTP URL queries, or programmatically. 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. A developer user interface is also included, which shows all current live streams and gives the option to record or stop recording a stream. For more information, see How to record live streams (HTTPLiveStreamRecord).

**AC-3 Passthrough**

Wowza Media Server 3.6 allows Dolby® Digital 5.1 Surround Sound (AC-3) and Dolby Digital Plus (Enhanced AC-3 or E-AC-3) audio from MPEG-TS encoders to pass through unaltered for delivery using Apple HLS and Microsoft Smooth Streaming. This allows your customers to experience immersive surround-sound audio when streaming live and on-demand video on media devices such as Apple TV®, Xbox 360®, Sony PlayStation® 3 (PS3), hybrid set-top boxes, and smart TVs.

**New features and functionality in Wowza Media Server 3.5**

**Wowza StreamLock AddOn**

Wowza StreamLock™ AddOn is a new security option for network encryption from Wowza Media Systems. It provides near-instant provisioning of free 256-bit Secure Sockets Layer (SSL) certificates to verified Wowza users for use with Wowza Media Server. StreamLock-provisioned SSL certificates provide the best security when used with RTMP. The certificates can also be used for secure HTTP streaming (HTTPS).

StreamLock is only available for subscription (Daily and Monthly) and Perpetual licensees running Wowza Media Server 3.0 or later. It's not available in the Trial and Developer
editions of Wowza Media Server. For more information, see How to get SSL certificates from the StreamLock service.

**Note**

StreamLock certificates expire after one year.

## Wowza Transcoder AddOn

Wowza Transcoder AddOn 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. Several enhancements were made to the AddOn for use with Wowza Media Server:

### Transcoder overlays

A new module for Wowza Transcoder AddOn enables you to overlay static and dynamic images on top of video by using a Java-based API. It can be configured manually or pre-programmed based on external events, making it a powerful tool for adding premium TV-like experiences. Examples of how this feature can be used include:

- Advertising
- Titling
- Watermarking
- Company logos or symbols
- Sports/Stock tickers

For more information, see How to add graphic overlays to live streams with Wowza Transcoder AddOn.

### Faster transcoding on Linux (technology preview)

Wowza Transcoder AddOn contains preview technology that makes the best use of 3rd generation Intel® Core™ processors on Linux-based operating system distributions by adding support for Intel Quick Sync Video. Using Intel Quick Sync Video can double transcoding speeds over 2nd generation Intel Core processors by moving video -conversion tasks to dedicated media processing space. The final version of the technology that supports Quick Sync Video will be available in a later release of Wowza Transcoder AddOn. For more information, see How to configure Quick Sync accelerated encoding on Linux.

### H.263 video compression

Wowza Transcoder AddOn supports H.263 encoding for live video streams, enabling users to stream to older devices.
Wowza DRM AddOn

Wowza DRM AddOn provides integration with third party Digital Rights Management (DRM) Key Management Systems (KMS) to add on-the-fly encryption for live and on-demand video workflows. Wowza DRM AddOn supports BuyDRM™ KMS services for Apple HLS. This allows studio-grade PlayReady DRM to be applied to Apple HLS streams that are delivered to Apple iOS devices that have a BuyDRM player. For more information, see How to set up and test BuyDRM KeyOS DRM (PlayReady).

Digital Rights Management

Wowza Media Server has APIs that enable several encryption schemes for on-the-fly encryption of live and on-demand 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). In addition, an API is provided that enables decryption of PlayReady assets. 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
- How to decrypt PlayReady encrypted video on demand content on the fly

MediaSecurity

MediaSecurity, formerly a free AddOn, is incorporated into Wowza Media Server. MediaSecurity features such as SecureToken, RTMP authentication, RTSP authentication, StreamNameAlias, and secure streaming (RTMPE, RTMPT, and RTMPS) help to ensure a more secure stream when delivering content using Apple HLS, Adobe HDS, Microsoft Smooth Streaming, and MPEG-DASH streaming. For more information, see Media Security Overview.

HTTP Origin

Wowza Media Server can be used as an origin that fulfills requests from HTTP caching infrastructures for live and on-demand Apple HLS, Adobe HDS, and Microsoft Smooth Streaming. Downstream HTTP caches are ideal for scaling out streaming across larger regions and audiences.

HTTP Origin functionality is available only to Trial, subscription (Daily and Monthly), and Perpetual licensees running Wowza Media Server. It’s not available in the Developer edition of Wowza Media Server. For more information, see How to configure Wowza Media Server as an HTTP caching origin.
Push Publishing

Push Publishing AddOn for Wowza Media Server includes support for publishing to
downstream Wowza servers running Wowza Media Server 3.5 (or later) using the WOWZ™
protocol (Wowza messaging protocol). It also includes support for publishing to downstream
Wowza Media Servers (all versions), Adobe Media Servers, and 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 get Push Publishing AddOn
(push to CDNs and other services).

Silverlight Multicast Player

Wowza Media Server contains a Microsoft Silverlight®-based player that allows users to
stream an MPEG-TS multicast from Wowza Media Server 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.
Introduction

What is Wowza Media Server?

Wowza Media Server® 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.

Wowza Media Server 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 Media Server 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 Wowza Media Server Articles and Forums.

To get started quickly with Wowza Media Server, see the Quick Start Guide.

Adobe HDS (Adobe Flash Player)

Wowza Media Server 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 Wowza Media Server. Adobe HDS is referred to as "San Jose" streaming in Wowza Media Server configuration files.

When streaming video on demand content, Wowza Media Server supports MP4 files (QuickTime container) and MP3 files (FLV files aren't supported). Wowza Media Server 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)

Real Time Messaging Protocol (Adobe Flash Player)

Wowza Media Server communicates with Adobe Flash Player using the Real Time Messaging Protocol (RTMP). Wowza Media Server can deliver adaptive bitrate live and on-demand media, data, and remote procedure call information to and from Flash Player using RTMP. It supports media streaming and other features such as shared objects, video recording, video chat, remote procedure calls, and more. Wowza Media Server 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 Media Server 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 secures the data being transmitted between Adobe Flash Player and Wowza Media Server. RTMPE transmission is supported through the use of Wowza RTMPE AddOn.
- 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. In Wowza Media Server 3.0 or later, subscription (Daily and Monthly) 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 can be used to tunnel through firewalls that employ stateful packet inspection.
- RTMPTE. An encryption variant of the RTMPT protocol.

Wowza Media Server 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).

**Apple HLS (iPhone, iPad, iPod touch, QuickTime, and more)**

Wowza Media Server can stream adaptive bitrate live and video on demand 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 Wowza Media Server. Apple HLS is referred to as "Cupertino" streaming in Wowza Media Server configuration files.

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

Wowza Media Server populates the playlist file with metadata that describes each of the available streams in an adaptive bitrate presentation. This enables the 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 Media Server can stream adaptive bitrate live and video on demand 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 Wowza Media Server 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 Media Server 3.6 includes a preview version of MPEG-DASH technology for streaming live and on-demand video 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 Wowza Media Server. 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 preview version of MPEG-DASH technology in Wowza Media Server 3.6 doesn't support all of the features that are defined in the MPEG-DASH standard. It currently supports a subset of the available chunk-addressing schemes, container formats, and content-encoding formats for live and on-demand streaming to select DASH clients. Wowza will continue to develop MPEG-DASH streaming features in subsequent releases of Wowza Media Server to meet industry needs. For more information about MPEG-DASH support in Wowza Media Server 3.6, see the MPEG-DASH Overview.

**Real Time Streaming Protocols (QuickTime, VLC, 3GPP Devices, Set-top Boxes, and More)**

Wowza Media Server 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. Wowza Media Server can also accept incoming streams from encoding devices that use these protocols. Wowza Media Server supports RTP and MPEG-2 TS input and output over UDP as well as multicast. In addition, Wowza Media Server 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 Media Server supports the following RTSP, RTP, and MPEG specifications:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG-TS</td>
<td>ISO/IEC 13818-1</td>
</tr>
<tr>
<td>MPEG-TS over RTP</td>
<td>rfc2038</td>
</tr>
<tr>
<td>RTP: AAC</td>
<td>rfc3640, rfc3016, ISO/IEC 14496-3</td>
</tr>
<tr>
<td>RTP: G.711</td>
<td>rfc3551</td>
</tr>
<tr>
<td>RTP: H.263</td>
<td>rfc2429</td>
</tr>
<tr>
<td>RTP: H.264</td>
<td>rfc3984, QuickTime Generic RTP Payload Format</td>
</tr>
<tr>
<td>RTP: MP3</td>
<td>rfc2250</td>
</tr>
<tr>
<td>RTP: MPEG-2 (video)</td>
<td>rfc2250</td>
</tr>
<tr>
<td>RTP: MPEG-4 Part 2</td>
<td>rfc3106</td>
</tr>
<tr>
<td>RTP: Speex</td>
<td>rfc5574</td>
</tr>
<tr>
<td>RTSP</td>
<td>rfc2326</td>
</tr>
</tbody>
</table>
Wowza Media Server supports both Single Program (SPTS) and Multi Program (MPTS) MPEG-TS streams and provides the ability to specify a specific program, a specific language, or a specific audio or video track in an MPTS stream. Query parameters are part of the udp:// URL inside of 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 Media Server can stream live and on-demand video to many player technologies. Wowza Media Server supports the following video on demand file formats: FLV (Flash Video - .flv), MP4 (QuickTime container - .mp4, .f4v, .mov, .m4a, .m4v, .mp4a, .mp4v, .3gp, and .3g2) and MP3 content (.mp3). Wowza Media Server can accept live video and audio streams from encoders that support the RTMP, RTSP/RTP, native RTP, and MPEG-TS protocols. Wowza Media Server can record any incoming live stream to either the Flash Video (FLV) or MP4 (QuickTime container) format.

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

Wowza Media Server 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. Wowza Media Server maintains a single connection to the original source stream while delivering the stream to multiple players. Wowza Media Server 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 Wowza Media Server installation illustrates these capabilities.

Wowza Media Server 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 AddOn provides the ability to ingest a live stream, decode the video and audio, and then re-encode the stream to suit desired playback devices. Wowza Transcoder is a real-time video transcoding and transrating solution. 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.

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.

### Wowza nDVR AddOn

Wowza nDVR AddOn provides the ability to record a live stream into a cache on Wowza Media Server while allowing users to play or pause a live stream, rewind to a previously recorded point, or resume viewing at the current live point. Customization is possible through XML configurations and the available APIs. Configuration for client playback of recorded streams is similar to playback of live streams from Wowza Media Server.

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 Systems (KMS) to add on-the-fly encryption for live and video on demand workflows. For live workflows, per-stream encryption is available with the ability to rotate keys. For on-demand workflows, per-asset and per-session encryption is available with the ability to rotate keys. Both live and video on demand key rotation support is available for Apple HTTP Live Streaming (HLS).

Currently, integration is supported for the following Key Management Systems:

- BuyDRM™ KeyOS™. Supports Microsoft® PlayReady® protected Apple HLS and Microsoft Smooth Streaming playback with BuyDRM players on iOS-based devices (iPhone/iPad) and Windows® Phone devices.
- EZDRM. Supports Smooth Streaming playback with Silverlight clients.
- Verimatrix®. Supports VCAS™ protected Apple HLS playback with ViewRight® clients on iOS-based devices, Android™ devices, PCs, and set-top boxes and PlayReady® protected Smooth Streaming playback with Smooth Streaming clients.

For more information about Wowza DRM AddOn, see the Wowza DRM online tutorials and the Wowza DRM Forum.

Note

Wowza Media Server has APIs that enable encryption schemes for on-the-fly encryption of live and on-demand 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). In addition, Wowza Media Server has an API that enables on-the-fly encryption of live and on-demand 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

Server Architecture

Wowza Media Server is a pure Java server. It's written in Java and can be extended dynamically using custom Java classes. Wowza Media Server can be deployed in any environment that supports the Java 6 and Java 7 virtual machine or later. Wowza Media Server is fully 64-bit compliant. It's architected to be highly multi-threaded and can take full advantage of multi-core hardware. All logging is done using the Apache log4j logging utility and uses the W3C Extended Common Log Format (ECLF).
Wowza Media Server was designed to handle multiple streaming protocols. The server-side API makes it easy to control the streaming process of each supported streaming protocol and player technology. Streaming is controlled through the creation and configuration of a streaming application. One application can be configured to simultaneously deliver live or video on demand content to multiple player technologies.

Wowza Media Server includes the ability to share a single server using a virtual hosting configuration. Virtual hosts can be configured with their own system resource and streaming limitations.

**Wowza Media Server Editions**

Wowza Media Server 3.6 comes in five editions: Trial, Monthly, Daily, Perpetual, and Developer.

<table>
<thead>
<tr>
<th>Edition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial Edition</td>
<td>The free Trial Edition provides full, unrestricted functionality of Wowza Media Server and AddOns, but is limited to 30 days of use from the date of issue. Wowza Transcoder streams contain audio/video watermarks. Other restrictions apply as described in the Wowza Media Software EULA.</td>
</tr>
<tr>
<td>Monthly/Daily Edition</td>
<td>These licenses provide full, unrestricted functionality of Wowza Media Server and AddOns, and allow the use of an unlimited number of server instances and AddOns under a single license key. Fees apply for each server instance and AddOn in use. The Monthly and Daily Editions differ only in payment terms. The use of these Editions is further permitted on Amazon Elastic Compute Cloud™ (EC2) and other computing cloud environments. See the Wowza Media Software EULA for more information.</td>
</tr>
<tr>
<td>Perpetual Edition</td>
<td>The Perpetual Edition provides full, unrestricted functionality of Wowza Media Server, but requires separate license keys for each server. In addition, each AddOn feature is licensed separately. Wowza nDVR AddOn and Wowza DRM AddOn licenses provide unlimited connection capacity per instance. Each Wowza nDVR and Wowza DRM license must be used with a Wowza Media Server Perpetual Edition license. Each Wowza Transcoder AddOn license is limited to one incoming channel (stream) and an unlimited number of outbound streams per the Wowza Media Server Perpetual Edition license. Multiple Wowza Transcoder AddOn licenses can be stacked on a single Wowza Media Server Perpetual Edition license for additional channel capacity. See the Wowza Media Software EULA for more information.</td>
</tr>
<tr>
<td>Developer Edition</td>
<td>The Developer Edition provides full, unrestricted functionality of Wowza Media Server and AddOns, but is limited to 180</td>
</tr>
</tbody>
</table>
days of use from the date of issue, and is further limited to ten (10) concurrent connections with live streaming restricted to two (2) inbound and ten (10) total combined concurrent inbound and outbound streams. Wowza Transcoder streams contain audio/video watermarks. See the Wowza Media Software EULA for more information.
Server Installation

How do I install Wowza Media Server?

Wowza Media Server® is a small and powerful Java server. This chapter describes how to choose the correct version of Java and install and run Wowza Media Server.

Before Installation

Wowza Media Server 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 JRE has everything needed to run Wowza Media Server on your system.

The following Java packages can be used with Wowza Media Server:

- **Java Development Kit (JDK).** The JDK includes a complete JRE and enables Java developers to develop and debug server-side applications. It also includes the JConsole monitoring tool, which enables you to manage and monitor a local or remote Wowza Media Server through the Java Management Extensions (JMX) interface.

- **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 Media Server is removed from 32-bit installations of the JRE, starting in JRE 7 Update 21 (JRE 7u21). To run Wowza Media Server on 32-bit platforms with a more recent Java JRE, you must use the Java Server JRE or Java JDK.

To get the best performance, we recommend that you deploy Wowza Media Server 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](http://www.oracle.com/technetwork/java/javase/downloads/index.html) webpage.
Notes

- Wowza Media Server 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 (command shell) and entering the command java –version. If correctly installed and configured, it will return a version number that's equal to or greater than 1.6.

Note

The Articles tab in Wowza Media Server Articles and Forums has more information and links to help you get the right Java environment and tools for your platform.

Note

On the Windows platform, Wowza Media Server uses the JAVA_HOME environment variable to determine the location of the Java environment under which it runs. If you have problems starting Wowza Media Server on Windows, make sure that the JAVA_HOME variable points to a Java 6 (aka 1.6) or later Java environment. If you make changes or upgrades to your Java environment, and 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 the Server

On the Windows and Mac OS® X platforms, Wowza Media Server is installed using an installer. On Linux®, Solaris®, and other Unix®-based platforms, the software is installed using a self-extracting binary installer. The installers are available for download from the Wowza Installers webpage.
Windows

To install Wowza Media Server on Windows operating systems, double-click the installer file and follow the instructions on the screen. (To find the installer file on Windows 8 and Windows Server 2012 operating systems, press WIN key + F and then search for WowzaMediaServer-3.6.4.) During the installation process, you'll be asked to enter the product license key. You can't proceed with the installation until you've entered a valid license key. Files will be installed to the following location:

```
/Program Files (x86)/Wowza Media Systems/Wowza Media Server 3.6.4
```

Here you'll find documentation, server application files, and folders: bin, conf, content, examples, lib and logs.

Mac OS X

To install Wowza Media Server on Mac OS X, mount the disk image (double-click .dmg) file, double-click the installer package (.pkg) file, and then follow the instructions on the screen. Files will be installed to the following locations:
To uninstall, move the following folders and files to the trash.

<table>
<thead>
<tr>
<th>folder</th>
<th>/Applications/Wowza Media Server 3.6.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder</td>
<td>/Library/WowzaMediaServer-3.6.4</td>
</tr>
<tr>
<td>symlink</td>
<td>/Library/LaunchDaemons/com.wowza.WowzaMediaServer.plist</td>
</tr>
</tbody>
</table>

**Note**

To ensure that Wowza Media Server folders are completely removed from the library, in Mac OS X, on the Go menu, click Go to Folder, type /Library, and then click Go. In the Library window, move the WowzaMediaServer-3.6.4 and WowzaMediaServer folders to the trash.

**Linux**

This section describes how to install Wowza Media Server on Linux systems. During the installation process, you'll be asked to agree to the Wowza Media Software End User License Agreement ("Wowza Media Software EULA"). The package manager will extract and install the files in the /usr/local/WowzaMediaServer-3.6.4 directory and the server will be installed as the root user.

**Red Hat Package Manager Systems**

**Install**

```bash
sudo chmod +x WowzaMediaServer-3.6.4.rpm.bin
sudo ./WowzaMediaServer-3.6.4.rpm.bin
```

**Uninstall**

```bash
sudo rpm -e WowzaMediaServer-3.6.4
```

**Debian Package Manager Systems**

**Install**

```bash
sudo chmod +x WowzaMediaServer-3.6.4.deb.bin
sudo ./WowzaMediaServer-3.6.4.deb.bin
```
Uninstall

```bash
sudo dpkg --purge wowzamediaserver-3.6.4
```

Other Linux and Unix Systems

Install

To install Wowza Media Server on other Linux and Unix based systems such as Solaris, open a terminal window, download `WowzaMediaServer-3.6.4.tar.bin` to any directory, and then execute the self-extracting installer:

```bash
sudo chmod +x WowzaMediaServer-3.6.4.tar.bin
sudo ./WowzaMediaServer-3.6.4.tar.bin
```

Uninstall

```bash
sudo rm -rf WowzaMediaServer-3.6.4
```

Starting and Stopping the Server

Windows

Standalone

To start/stop Wowza Media Server on Windows 7 and Windows Server 2008 operating systems, choose Wowza Startup/Shutdown from the Start menu (Start > All Programs > Wowza Media Server 3.6.4 > Wowza Startup/Shutdown).

To start/stop Wowza Media Server on Windows 8 and Windows Server 2012 operating systems, choose Wowza Startup/Shutdown from the Start screen (Start > All Apps > Wowza Media Server 3.6.4 > Wowza Startup/Shutdown).

The server can also be started from a command prompt. To do this:

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

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

Note

When you start Wowza Media Server in standalone mode on Windows operating systems, a Getting Started webpage is automatically displayed. The webpage can help you to get up-and-running quickly by playing a sample video file from your local Wowza Media Server installation. It also provides links to tutorials and documentation. If you installed Wowza Media Server with a Trial or Developer license, this page will also tell you when your license will expire.
If you don't want this webpage to be displayed, you can turn it off. See Turning off the Windows Getting Started webpage.

To stop the server, open another Command prompt window and execute the following commands:

```bash
cd %WMSAPP_HOME%/bin
shutdown.bat
```

Service

To start the server as a Windows 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 Media Server 3.6.4, and then click Start.

To stop the Wowza Media Server 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 Media Server 3.6.4, and then click Stop.

To configure the Wowza Media Service to start 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 Media Server 3.6.4, and then click Properties.
3. In the Properties dialog box, on the General tab, set Startup type to either Automatic or Automatic (Delayed Start).

Note

The Windows service runs under the Local System Account by default. This can limit how Wowza Media Server interacts with the underlying operating system. For example, you may not be able to connect to Wowza Media Server using JConsole/JMX or you may have issues streaming content from UNC paths. To address these issues, update the Wowza Media Server 3.6.4 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.
Note

The hardware acceleration (NVIDIA CUDA® and Intel® Quick Sync) used by Wowza Transcoder is only available when running Wowza Media Server as a Windows standalone application. The hardware acceleration won’t be available when Wowza Media Server is invoked as a service.

Mac OS X

Standalone

On Mac OS X, the server can be started in standalone mode either by invoking it from the Server Startup script in /Applications/Wowza Media Server 3.6.4 or by opening a terminal window and entering the following commands:

cd /Library/WowzaMediaServer/bin
./startup.sh

To stop the server, open another terminal window and enter the following commands:

cd /Library/WowzaMediaServer/bin
./shutdown.sh

Note

When you run the server in standalone mode for the first time, you'll be asked to enter your license key from your approval email in a terminal window. After you enter your license key in the terminal window, it will be stored in the /Library/WowzaMediaServer/conf/Server.license file.

Service

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

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

To stop the service, enter:

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

Linux

Standalone

To start the server in standalone mode on Linux, open a command shell and then enter the following commands:
To stop server, open a command shell and enter:

cd /usr/local/WowzaMediaServer/bin
./shutdown.sh

**Note**

When you run the server in standalone mode for the first time, you'll be asked to enter your license key from your approval email in a terminal window. After you enter your license key in the terminal window, it will be stored in the


**Service**

To start the server as a Linux service, open a command prompt and then enter one of the following commands (the commands differ based on your Linux distribution):

/sbin/service WowzaMediaServer start

-or-

/etc/init.d/WowzaMediaServer start

To stop the service, enter one of the following commands:

/sbin/service WowzaMediaServer stop

-or-

/etc/init.d/WowzaMediaServer 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.

**Note**

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)](link).
Entering a New License Key

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

Trial, Daily, and Monthly Edition subscribers will have a single license key while Perpetual Edition users may have more than one key to enable purchased AddOns.

To add a license key, open the Server.license file in a text editor and enter each new license key on a new line. When the standalone server is restarted, the new license will be in effect. The licenses are additive, so when adding additional licenses, be sure to retain the original license information in the file 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 console window.

The following is an example Server.license file for a Perpetual Edition user with a Wowza Media Server license key, two Wowza Transcoder AddOn license keys, and one Wowza nDVR AddOn license key:

| SVRP3-LaGpC-zrTD9-F4Y3S-a9bR2-h5t3C |
| TRN23-Ry6qe-4mT8J-yKj2W-4N5sH-27d3a |
| TRN13-y9Cj2-kneqT-2zjHp-GadzB-N6fwa |
| DVRA3-k3r3R-nzxCB-ypjs5-Sk3y9-ahFdF |

Ports Used for Streaming

Before streaming with Wowza Media Server, you should open ports on your firewall. The following table shows the default ports that Wowza Media Server 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 8084-8085 | JMX/JConsole monitoring and administration |
| TCP 8086 | Administration |
| UDP 6970-9999 | RTP UDP streaming |

By default, Wowza Media Server 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, in a text editor, edit [install-dir]/conf/VHost.xml and add the additional ports to the <Port> list (this list is comma-delimited). Wowza Media Server 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 | Apple HTTP Live Streaming (HLS), Adobe HTTP Dynamic Streaming (HDS), |
Server Configuration and Tuning

Wowza Media Server is configured through a set of XML configuration and properties files in the [install-dir]/conf folder.

**Note**

It's very important that Wowza Media Server be tuned properly so that it can take best advantage of available hardware resources. The default tuning of the server is sufficient for application development, but not for production use. Without proper tuning, the server under medium to heavy load will run out of resources and stop working properly. For more information about how to tune Wowza Media Server, see [Performance Tuning](#).

**Configuration Files**

The configuration files are read during server startup. They can be directly edited using a standard text editor.

**Server Configuration Files**

- **Server.xml** - General server configuration
- **VHosts.xml** - Virtual hosts definition
- **log4j.properties** - Logging configuration

**Virtual Host Configuration Files**

- **Authentication.xml** - RTSP and HTTP authentication configuration
- **CEACaptionConversion.xml** - CEA-608 closed captioning configuration
- **DVR.xml** - nDVR base configuration
- **HTTPStreamers.xml** - Cupertino (Apple HLS), Smooth (Smooth Streaming), and San Jose (Adobe HDS) streaming configuration
- **LiveStreamPacketizers.xml** - HTTP packetization configuration
- **LiveStreamTranscoders.xml** - Transcoder base configuration
- **MediaCasters.xml** - MediaCaster restreaming configuration
- **MediaReaders.xml** - File format reader configuration
- **MediaWriters.xml** - File format writer configuration
- **MP3Tags.xml** - MP3 ID3 tag naming
- **RTP.xml** - RTP and MPEG-TS packetization configuration
- **StartupStreams.xml** - Streams started at virtual host startup
- **Streams.xml** - Stream type configuration
TimedTextProviders.xml - Closed-captioning configuration
VHost.xml - Virtual host configuration
VHosts.xml - Virtual hosts configuration

Application Configuration Files
Application.xml - Application configuration

For more information about the configuration files, see the Wowza Media Server Configuration Reference.

Scripts

When Wowza Media Server starts, it runs a set of scripts that control the settings associated with the Java runtime environment, such as the command used to invoke Java and the maximum Java heap size. The location of these files differs depending on the platform and the method that's used to start Wowza Media Server. For more information about how to tune the settings for Java runtime environment for best performance, see Performance Tuning.

Run Server as Named User

The default installation of Wowza Media Server on Linux and Mac OS X installs and runs the server as the root user. If you want to run the server as a different user, follow the instructions in Run Wowza Media Server as Named User (Linux and OS X) to create a new user and configure the server to run as that new user.

Note

For security reasons, most Linux and Unix distributions only allow the root user to bind to port numbers less than 1024. If you plan on running Wowza Media Server on a lower-numbered ports such as 80 (HTTP), 443 (HTTPS, RTMPS), or 554 (RTSP), then the server must continue to run as the root user.

Upgrading from a Previous Release

Before upgrading to Wowza Media Server 3.6.4, the previously installed version should be uninstalled. For more information about how to upgrade from an earlier version of Wowza Media Server to Wowza Media Server 3.6.4, see the Wowza Media Server Upgrade Guide.
Co-Existence of Multiple Wowza Media Server Instances

You can run multiple instances of the same version of Wowza Media Server on one computer instead of running one large instance with multiple virtual hosts. An instance is defined as a single copy of a running Wowza Media Server. 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 involves 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 (Daily or Monthly) or 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 Media Server on one computer.

**Note**

You can install different versions of Wowza Media Server for testing and comparison purposes. Only one version should be running at any time to avoid network port conflicts and all installed Wowza Media Server versions must be properly licensed. For more information, see How to install different versions of Wowza Media Server on one computer.

Patch Updates

In between stable, production releases, the Wowza Team periodically produces development builds in the form of patches. This allows users to test the latest releases of Wowza Media Server software and give feedback, and to get early access to new features. Specific instructions for installing patches are provided in a README.txt file that's included in the patch archive file. For more information, see Software Updates.
Application Configuration

How do I create and configure an application for streaming?

All streaming in Wowza Media Server® is controlled through the creation and configuration of applications. An application is defined simply by creating a folder in the [install-dir]/applications folder. For example, to create a new application named myapplication, create the following folder:

[install-dir]/applications/myapplication

A single application can be configured to deliver a live or video on demand stream to Adobe® Flash® Player, Microsoft® Silverlight®, Apple® iOS devices (iPhone®, iPad®, or iPod touch®) or Apple TV® digital media extender, Roku® and Amino set-top boxes, and RTSP/RTP-based players (including 3GPP smartphones and tablets, and Android™ devices) at the same time. 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 Media Server Configuration Reference.

Applications and Application Instances (Application.xml)

An application is created by creating a named folder in [install-dir]/application. The application name is the same as the folder name. An Application.xml file defines the configuration 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_. Multiple application instances are more commonly used in video chat and video conferencing scenarios where you must create multiple rooms for streaming. In this case, an application
instance is used to separate streaming into rooms. Each room is a separate application instance, which provides separation and a namespace for each room.

Application configuration is defined in an Application.xml file. 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.

**Note**

It's a common mistake to put the Application.xml file in the [install-dir]/applications/[application] folder. All configuration files for Wowza Media Server and its applications should be located in the [install-dir]/conf/[application] folder.

## URL Formats

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

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

- **[protocol]**: streaming protocol (http, rtmp, rtsp, and so on)
- **[address]**: address of the server running Wowza Media Server
- **[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 contain 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 being streamed.

**Adobe Flash Player (RTMP)**

**Server:** rtmp://mycompany.com/live  
**Stream:** myStream
Adobe Flash Player (Adobe HTTP Dynamic Streaming)
http://mycompany.com:1935/live/myStream/manifest.f4m

Apple iPhone, iPad, or iPod touch (Apple HTTP Live Streaming)
http://mycompany.com:1935/live/myStream/playlist.m3u8

Microsoft Silverlight (Microsoft Smooth Streaming)

DASH Clients (MPEG-DASH Streaming)

RTSP/RTP
rtsp://mycompany.com:1935/live/myStream

Now is probably a good time to take a quick look at the default [install-dir]/conf/Application.xml file. The rest of this chapter describes the more commonly configured items in this file.

Stream Types

Wowza Media Server uses named stream types to control the different types of streaming (live, video on demand, chat, remote recording, and so on.). Stream types are configured using the Streams/StreamType property in Application.xml. Stream types are defined in [install-dir]/conf/Streams.xml. The following table shows the stream types and their uses.

<table>
<thead>
<tr>
<th>Stream type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chat</td>
<td>Live video chat</td>
</tr>
<tr>
<td>default</td>
<td>Video on demand</td>
</tr>
<tr>
<td>file</td>
<td>Video on demand</td>
</tr>
<tr>
<td>live</td>
<td>Publish and play live video content (best for one-to-many streaming of live events)</td>
</tr>
<tr>
<td>live-lowlatency</td>
<td>Publish and play live video content (best for one-to-one or one-to-few video/audio chat applications)</td>
</tr>
<tr>
<td>live-record</td>
<td>Same as live—in addition content is recorded</td>
</tr>
<tr>
<td>live-record-lowlatency</td>
<td>Same as live-lowlatency—in addition content is recorded</td>
</tr>
<tr>
<td>liverepeater-edge</td>
<td>Publish and play live video content across multiple Wowza Media Servers in an origin/edge configuration (used to configure edge application)</td>
</tr>
<tr>
<td>liverepeater-edge-lowlatency</td>
<td>Publish and play live video content across multiple Wowza Media Servers in an origin/edge configuration (used to configure edge application when latency is important)</td>
</tr>
<tr>
<td>liverepeater-edge-origin</td>
<td>Publish and play live video content across multiple Wowza Media Servers in an origin/edge configuration (used to configure edge application when latency is important)</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>liverepeater-origin</td>
<td>Publish and play live video content across multiple Wowza Media Server servers in an origin/edge configuration (used to configure a middle-edge application)</td>
</tr>
<tr>
<td>liverepeater-origin-record</td>
<td>Same as liverepeater-origin—in addition content is recorded</td>
</tr>
<tr>
<td>record</td>
<td>Video recording</td>
</tr>
<tr>
<td>rtp-live</td>
<td>Re-streaming of RTSP/RTP, native RTP, or MPEG-TS streams</td>
</tr>
<tr>
<td>rtp-live-lowlatency</td>
<td>Re-streaming of RTSP/RTP, native RTP, or MPEG-TS streams when latency is important</td>
</tr>
<tr>
<td>rtp-live-record</td>
<td>Same as rtp-live—in addition content is recorded</td>
</tr>
<tr>
<td>rtp-live-record-lowlatency</td>
<td>Same as rtp-live-lowlatency—in addition content is recorded</td>
</tr>
<tr>
<td>shoutcast</td>
<td>Audio re-streaming of SHOUTcast/Icecast MP3 or AAC+ audio streams</td>
</tr>
<tr>
<td>shoutcast-record</td>
<td>Same as shoutcast—in addition content is recorded</td>
</tr>
</tbody>
</table>

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. Properties defined in `[install-dir]/conf/Streams.xml` for a given stream type can be overridden on a per-application basis by defining new values in the `Streams/Properties` container in `Application.xml`. For example, to change the `flushInterval` property of the `live-lowlatency` stream type, the `<Streams>` container in `Application.xml` should look like this:

```
<Streams>
    <StreamType>live-lowlatency</StreamType>
    <StorageDir>${com.wowza.wms.context.VHostConfigHome}/content</StorageDir>
    <KeyDir>${com.wowza.wms.context.VHostConfigHome}/keys</KeyDir>
    <LiveStreamPacketizers></LiveStreamPacketizers>
    <Properties>
        <Property>
            <Name>flushInterval</Name>
            <Value>30</Value>
            <Type>Integer</Type>
        </Property>
    </Properties>
</Streams>
```

### HTTP Streamers and Live Stream Packetizers

The `<HTTPStreamers>` section in `Application.xml` controls if the streams in the defined application (live or video on demand) are available for playback to Apple iOS devices, Microsoft Silverlight, and HTTP streaming to Adobe Flash Player. `HTTPStreamers` can contain none, one, or more than one of the following values (separated by commas):
HTTPStreamer | Description
---|---
cupertinostreaming | Enables live and video on demand content to be streamed 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 Roku and Amino set-top boxes and some brands of smart TVs, using the Apple HTTP Live Streaming (HLS) protocol.
smoothstreaming | Enables live and video on demand content to be streamed to Silverlight using the Microsoft Smooth Streaming protocol.
sanjosestreaming | Enables live and video on demand content to be streamed to Flash Player using the Adobe HTTP Dynamic Streaming (HDS) protocol.
dvrchunkstreaming | Enables live and video on demand content to be streamed from Wowza Media Server (origin) to Wowza Media Server (edge).
mpegdashstreaming | Enables live and video on demand content to be streamed to DASH clients using the HTTP protocol.

Live streams coming into Wowza Media Server must be packaged (packetized) before they can be delivered to clients using HTTP streaming protocols such as Apple HLS, Adobe HDS, and Smooth Streaming. The `<Streams>/LiveStreamPacketizers` list 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 stream from a single Wowza Media Server to clients or from an origin Wowza Media Server to an edge Wowza Media Server when using the live repeater mechanism in an origin/edge configuration. *LiveStreamPacketizers* can contain none, one, or more than one of the following streaming packetizers.

<table>
<thead>
<tr>
<th>LiveStreamPacketizers</th>
<th>Description</th>
</tr>
</thead>
</table>
cupertinostreamingpacketizer | Enables Apple HLS live streaming to iOS-based devices. |
sanjosestreamingpacketizer | Enables Adobe HDS live streaming to Flash Player. |
smoothstreamingpacketizer | Enables Microsoft Smooth Streaming to Silverlight. |
dvrsreamingpacketizer | nDVR packetizer for use with Wowza nDVR AddOn. |
mpegdashstreamingpacketizer | Enables MPEG-DASH streaming to DASH clients. |

A packetizer is set with a repeater value on an edge Wowza Media Server in an origin/edge configuration. Repeater packetizers are used when using the live repeater mechanism to deliver a live stream from an origin Wowza Media Server to an edge Wowza Media Server. *LiveStreamPacketizers* on the edge Wowza Media Server can contain none, one, or more than one of the following repeater packetizers.

<table>
<thead>
<tr>
<th>LiveStreamPacketizers</th>
<th>Description</th>
</tr>
</thead>
</table>
cupertinostreamingrepeater | Enables Apple HLS live stream repeater for iOS-based devices. |
sanjosestreamingrepeater | Enables Adobe HDS live stream repeater for Flash Player.
smoothstreamingrepeater | Enables Microsoft Smooth Streaming live stream repeater for Silverlight.
dvrstreamingrepeater | nDVR live stream repeater for use with Wowza nDVR AddOn.

For more information about how to configure and implement the live stream repeater mechanism for delivering a live media event across multiple Wowza Media Servers in and origin/edge configuration, 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 Media Server in an origin/edge configuration. For more information, see the Wowza nDVR User's Guide.

**Wowza Timed Text Configuration**

The `<TimedText>` container in the `Application.xml` file serves to configure Timed Text (closed captioning) for a Wowza application. For more information, see Closed Captioning Overview.

**Wowza Transcoder AddOn and Wowza nDVR AddOn Configurations**

The `<Transcoder>` and `<DVR>` containers in the `Application.xml` file serve to configure an application to use Wowza Transcoder AddOn and Wowza nDVR AddOn respectively. For more information, see the Wowza Media Server 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 `Application.xml`, 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.
A basic module definition looks like this:

```xml
<Module>
  <Name>base</Name>
  <Description>Base</Description>
  <Class>com.wowza.wms.module.ModuleCore</Class>
</Module>
```

Each module must have a unique `<Name>`. The `<Description>` property is for providing a detailed description of the module and isn't used in any operations. The `<Class>` property is the full 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 that's copied to the `[install-dir]/lib` folder. The Wowza Media Server 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 Wowza IDE to develop your own custom modules to provide additional functionality. For more information, see Extending Wowza Media Server Using Java.

**Note**

To download the free Wowza® Integrated Development Environment (Wowza IDE) tool, see the Wowza Developers webpage.

### Properties

The default `Application.xml` file contains several different `<Properties>` containers that can be used to add or override property values within Wowza Media Server. Properties are a list of name/value pairs that provide a means for tuning and modifying the default configuration of Wowza Media Server. Properties can also be used server-side as a means to pass data to custom modules from `Application.xml`.

A property definition has the following form:

```xml
<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: String, Integer, Boolean, Double, and Long. It's important when tuning the server to make sure to add properties to the correct container. Tuning instructions always specify which `<Properties>` container a property should be added to for tuning.
Media Types

Media types aren't configured in Application.xml but are an important part of streaming. Wowza Media Server supports many media types. Wowza Media Server can read the following media or file types:

- FLV (Flash Video - .flv)
- MP3 content (.mp3)
- MP4 (QuickTime container - .mp4, .f4v, .mov, .m4a, .m4v, .mp4a, .mp4v, .3gp, .3g2, etc.)
- 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. The media type prefix defaults to flv: if none is specified. The following table shows the supported media type prefixes.

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

The media type prefix is also used to control the file container that's used to record live video. When publishing video, if the mp4: media type 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 or if no prefix is specified, an FLV or Flash Video container is used.

Synchronized Multimedia Integration Language (.smil) files provide a means to specify a group of live streams or video on demand 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 video on demand, 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 template and are available for playback using the `ngrp:` media type prefix.

Wowza Media Server 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 stream name prefix `amlst:` to use a set of Java objects that describe the adaptive bitrate stream (an API-based MediaList). When the Wowza Media Server reads a SMIL file, it creates a MediaList and passes it back to the streaming provider. This API provides a means for intercepting these requests and creating the MediaList dynamically in a Wowza Media Server module. For more information, see How to use Java API calls to resolve SMIL file requests (AMLST).

**Content Storage**

By default Wowza Media Server is configured to stream video on demand content from (and record to) the `[install-dir]/content` folder. You can change this behavior by editing an application's `Application.xml` file and changing the value of `Streams/StorageDir`. For example, to configure an application to use an application-specific content folder, you may change this value to:

```xml
<StorageDir>${com.wowza.wms.context.VHostConfigHome}/applications/${com.wowza.wms.context.Application}/content</StorageDir>
```

Using this setting, content is streamed from the `[install-dir]/applications/[application]/content` folder, where `[application]` is the application's name. The `Streams/StorageDir` setting supports the following variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>${com.wowza.wms.AppHome}</td>
<td>Application home directory</td>
</tr>
<tr>
<td>${com.wowza.wms.ConfigHome}</td>
<td>Configuration home directory</td>
</tr>
<tr>
<td>${com.wowza.wms.context.VHost}</td>
<td>Virtual host name</td>
</tr>
<tr>
<td>${com.wowza.wms.context.VHostConfigHome}</td>
<td>Virtual host configuration directory</td>
</tr>
<tr>
<td>${com.wowza.wms.context.Application}</td>
<td>Application name</td>
</tr>
<tr>
<td>${com.wowza.wms.context.ApplicationInstance}</td>
<td>Application instance name</td>
</tr>
</tbody>
</table>
Advanced Configuration Topics

How do I take advantage of Wowza Media Server 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 Wowza Media Server®.

MediaCasters, Stream Manager, and StartupStreams.xml

Wowza Media Server includes a system for re-streaming called MediaCaster®. The MediaCaster system is used for re-streaming IP camera streams (RTSP/RTP streams), SHOUTcast/Icecast streams, and native RTP encoders. The MediaCaster system pulls a stream from a stream source and makes it available for streaming to the different player technologies supported by Wowza Media Server. This system works on demand—when the first request comes in 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, then the stream is stopped and is no longer available for streaming until another request comes in for it.

This methodology works great for the Adobe® Flash® player (RTMP) and for RTSP/RTP streaming where advanced packetization isn't required. For HTTP Streamers such as Cupertino (Apple HLS), Smooth (Microsoft Smooth Streaming), and San Jose (Adobe HDS), the on-demand startup model doesn't work. An iOS device requires about 30 seconds of video to be pre-packetized before it can begin playback. Microsoft Silverlight clients require three times the key frame duration. Therefore, it's necessary to start the stream prior to the stream being ready for streaming to these player technologies. The methodologies for starting a stream that uses the MediaCaster system and keeping it running are Stream Manager and the [install-dir]/conf/StartupStreams.xml configuration file.
Stream Manager is a web-based tool that's built into Wowza Media Server and is used for starting and stopping MediaCaster streams on the fly. To start Stream Manager, do the following:

1. In a text editor, open `[install-dir]/conf/admin.password` and enter a new line with a user name and password. For example, to add the user name `myuser` and the password `mypassword`:

```# Admin password file (format [username][space][password])
#username password
myuser mypassword```

2. Open the following URL in a web browser:

   `http://[wowza-ip-address]:8086/streammanager`

To start a stream, click on the `[start-receiving-stream]` link under the application to which you want to start the stream, select the MediaCaster type, type in the stream name and then click OK.

To stop a stream, click the `[stop-receiving-stream]` link next to stream name.

To reset a stream, click the `[reset-receiving-stream]` link.

The second method for starting MediaCaster streams is to use 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). `StartupStreams.xml` contains a list of applications, media caster types, and stream names. The format of a single entry is:

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

For example:

```<StartupStream>
   <Application>live</Application>
   <MediaCasterType>rtp</MediaCasterType>
   <StreamName>mpegs.stream</StreamName>
</StartupStream>```

The valid media caster types are:

- `rtp`
- `rtp-record`
- `shoutcast`
- `shoutcast-record`
- `liverepeater (origin/edge)`
Server-side methods can also be used to start and stop streams using the MediaCaster system:

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

For more information about these methods, see Wowza Media Server Server-Side API.

**Live Stream Repeater (Origin/Edge Live Streaming)**

This section recommends a configuration and implementation methodology (live stream repeater) for delivering a live media event across multiple Wowza® servers. A live stream repeater uses multiple Wowza Media 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 Media 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 Media Server 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 the application name `liverepeater`. To configure the origin server, do the following:

1. Create a folder named `[install-dir]/applications/liveorigin`.
2. Create a folder named `[install-dir]/conf/liveorigin` and copy the file `[install-dir]/conf/Application.xml` to this folder.
3. In a text editor, make the following changes in the `Application.xml` file:
   a. Change the `Streams/StreamType` value to `liverepeater-origin`.
   b. Change the `Streams/LiveStreamPacketizers` value to `cupertinostreamingpacketizer,smoothstreamingpacketizer,sanjosestreamingpacketizer`

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

1. Create a folder named `[install-dir]/applications/liveedge`.
2. Create a folder named `[install-dir]/conf/liveedge` and copy the file `[install-dir]/conf/Application.xml` to this folder.
3. In a text editor, make the following changes in the `Application.xml` file:
   a. Change the `Streams/StreamType` value to `liverepeater-edge`. (You can use `liverepeater-edge-lowlatency` if low latency is important; however, this will add extra load to the server.)
b. Change the Streams/LiveStreamPacketizers value to 
cupertinostreamingrepeater,smoothstreamingrepeater,sanjosestreamingrepeater.

c. Set OriginURL to the URL of the origin server using the WOWZ™ protocol URL 
prefix (wowz://). For example, if the origin server uses the domain name 
origin.mycompany.com, the value would be:

```xml
<Repeater>
  <OriginURL>wowz://origin.mycompany.com/liveorigin</OriginURL>
  <QueryString></QueryString>
</Repeater>
```

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 Flash Player (RTMP)**

Server: rtmp://edge1.mycompany.com/liveedge
Stream: mycoolevent

**Adobe Flash Player (Adobe HDS)**

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

**Apple iPhone, iPad, or iPod touch (Apple HLS)**

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

**Microsoft Silverlight (Smooth Streaming)**

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

**RTSP/RTP**

rtsp://edge1.mycompany.com:1935/liveedge/mycoolevent

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 set the following Repeater/OriginURL value in the Applications.xml files on each edge 
server:

```xml
<Repeater>
  <OriginURL>wowz://origin.mycompany.com/liveedge|wowz://origin2.mycompany.com 
  /liveedge</OriginURL>
  <QueryString></QueryString>
</Repeater>
```
The **Repeater/OriginURL** value is basically the two connection URLs concatenated by a pipe (|) character. 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 a text editor to create a file named `mycoolevent.stream` in the `[install-dir]/content` folder 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.

---

**Note**

The WOWZ™ protocol is a TCP-based messaging protocol in Wowza Media Server 3.5 (and later) and is used for server-to-server communication. It's enabled by default. If one of the Wowza Media Servers in the origin/edge configuration isn't running Wowza Media Server 3.5 (or later), an RTMP connection is established between the servers instead.

---

**Note**

You can secure the connection between Wowza Media Servers in and origin/edge configuration by using **SecureToken**. For more information, see the [Media Security Overview](#).

---

**Note**

If you stream to an iOS device or Silverlight, or if you use Flash HTTP to stream to Flash Player, and you use a non-push based encoder (native RTP or MPEG-TS), then you must use **Stream Manager** to start the stream on the origin server and keep it running. Streams don't need to be kept running on edge servers.

---

**Note**

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

The WebcamRecording example that's included with Wowza Media Server 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. If you just want to record an incoming live stream from an encoder, you can use one of the *-record stream types (such as live-record) or use the Live Stream Record feature in Wowza Media Server.

The *-record stream types are the easiest to use but give you the least amount of control. If you use this method, the entire duration of the published stream is recorded. If the encoder starts and stops, the file is versioned with a version number and a new file is started. You can control the container format used (FLV or MP4) by specifying a stream name prefix in the encoder. If you specify the flv: prefix, the stream is recorded to an FLV container; if you specify the mp4: prefix, the stream is recorded to an MP4 (QuickTime) container. Remember that an MP4 container can only record H.264, AAC, and MP3 media data. If you're recording with Flash Player, the FLV container is the only option.

The Live Stream Record feature in Wowza Media Server gives you more control over the recording process. This feature enables you to control the live stream recording process using a web-based user interface, HTTP URL queries, or programmatically. 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. A developer user interface is also included, which shows all current live streams and enables you to control when the recording starts and stops, the file name and locations, the container format, and other details. For more information, see How to record live streams (HTTPLiveStreamRecord).

Server-side Publishing (Stream and Publisher classes)

Wowza Media Server includes two classes for doing server-side publishing; the Stream class and the Publisher class. The Stream class is a high-level server-side API for mixing live and video on demand content on the fly into a single destination stream. It provides the ability to 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 provides the ability to inject raw compressed video and audio frames into Wowza Media Server 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.
Adobe Flash Streaming and Scripting

What can I do with Wowza Media Server and Adobe Flash Player?

Wowza Media Server® 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, Wowza Media Server is much 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 procedures 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:

```javascript
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")
    {
        ns = new NetStream(nc);
        ns.addEventListener(NetStream, NetStreamEvent.OPEN, onVideoStream, false, 0);
        ns.open(ChanName, Channal, "?play=true", null);
    }
}
```
7. On the Control menu, select Test Movie.

You should be streaming the sample.mp4 example 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 the Wowza Media Server is established. When we're notified of a successful connection, we create a NetStream object and begin playback of the stream.

The LiveVideoStreaming and VideoOnDemandStreaming example players that are installed with Wowza Media Server 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 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, 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 Media Server, see the following support articles:

- How to use LongTail JW Player with Wowza Media Server
- 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.

For more information about how to use Flowplayer with Wowza Media Server, see How to use Flowplayer with Wowza Media Server.

Adobe Media Playback player

The Adobe 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 Adobe Flash Media Playback player with Wowza Media Server.

Bi-directional Remote Procedure Calls

Wowza Media Server 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 Media Server. 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 Media Server. These features are available when using the RTMP protocol.

Calls from Flash Player to Wowza Media Server are performed using the following method:

NetConnection.call(methodName, resultObj, params...)
For example, to call the server-side method `doSomething`, pass the parameters `value1` and `value2`, and receive a single return value, the ActionScript 3.0 client-side code looks like this:

```actionscript
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 Media Server is done by adding handler methods/functions to the client object that's attached to the `NetConnection` object. For example, to add the handler method `onSomethingHappened` that receives two string parameters `value1` and `value2`, the ActionScript 3.0 code looks like this:

```actionscript
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 Media Server Using Java](#).

### Remote Shared Objects

Wowza Media Server supports Adobe Flash remote shared objects (RSOs), which enable data-sharing between a Wowza Media Server 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 Wowza Media Server 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 Media Server `ISharedObject` API. The following example shows the ActionScript 3.0 code that's needed to create a remote shared object and set a value:

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

function ncOnStatus(infoObj:NetStatusEvent)
{
    trace("ncOnStatus: "+infoObject.info.code);
    if (infoObject.info.code == "NetConnection.Connect.Success")
```
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 Media Server?

Much of the functionality delivered by Wowza Media Server® 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 server information. This chapter reviews these methods for extending Wowza Media Server and the built-in Java classes that are immediately available for use.

For more information about the programming model that you can use to create your own server-side extensions, see Extending Wowza Media Server 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. For the most part, server-side modules 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. Server-side modules can also be used to control Apple iPhone/iPad/iPod touch, Microsoft Silverlight, Adobe HDS, and RTSP/RTP streaming. For more information about how the API works, see Extending Wowza Media Server Using Java.
Server-side modules are configured by adding `<Module>` entries to the `<Modules>` list in an application's `Application.xml` file. The default `<Modules>` list looks like this:

```xml
<Modules>
  <Module>
    <Name>base</Name>
    <Description>Base</Description>
    <Class>com.wowza.wms.module.ModuleCore</Class>
  </Module>
  <Module>
    <Name>logging</Name>
    <Description>Client Logging</Description>
    <Class>com.wowza.wms.module.ModuleClientLogging</Class>
  </Module>
  <Module>
    <Name>flvPlayback</Name>
    <Description>FLVPlayback</Description>
    <Class>com.wowza.wms.module.ModuleFLVPlayback</Class>
  </Module>
</Modules>
```

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 Media Server Using Java.

**Built-in Server-side Modules**

This section briefly describes the server-side modules that are built-in with Wowza Media Server. For more information about the methods that are provided in a module, see the Wowza Media Server 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.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setStreamType(streamType:String);</code></td>
<td>Returns and sets the default stream type for this client connection.</td>
</tr>
<tr>
<td><code>getStreamType();</code></td>
<td></td>
</tr>
<tr>
<td><code>setRepeaterOriginUrl(originUrl:String);</code></td>
<td>Returns and sets the live stream repeater origin URL to use for this connection in an origin/edge configuration.</td>
</tr>
<tr>
<td><code>getRepeaterOriginUrl();</code></td>
<td></td>
</tr>
<tr>
<td><code>getStreamLength(streamName:String);</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>getStreamLength(streamNames:Array);</code></td>
<td></td>
</tr>
<tr>
<td><code>getClientID();</code></td>
<td>Returns the client ID for this client</td>
</tr>
</tbody>
</table>
connection.

getReferrer();

Gets the referrer from the onConnect method.

getPageUrl();

Gets the pageUrl from the onConnect method.

getVersion();

Returns the server name and version.

getLastStreamId();

Returns the ID number of the last NetStream object that was created by this client.

FCSSubscribe(streamName, [mediaCasterType]);

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.

FCUnSubscribe(streamName);

FCPublish(streamName);

FCUnpublish(streamName);

Called to tell the Wowza Media Server 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.

HTTP Providers

HTTP Providers are mini HTTP servers that can be used to extend Wowza Media Server 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.streammanager.HTTPStreamManager</BaseClass>
  <RequestFilters>streammanager*</RequestFilters>
  <AuthenticationMethod>admin-digest</AuthenticationMethod>
</HTTPProvider>

The **BaseClass** property is the full 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 **streammanager** (for example, http://[wowza-ip-address]:8086/streammanager). **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 cleartext) 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 Media Server Using Java](mailto:extending-wowza-media-server-using-java).

**Built-in HTTP Providers**

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

  Serves up 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)
  Serves up the Adobe Flash **crossdomain.xml** file when present in the [install-dir]/conf folder.

- **HTTPLiveStreamRecord** - (com.wowza.wms.livestreamrecord.http.HTTPLiveStreamRecord)
  Serves up the Live Stream Record web-based user interface that's available through administrative port 8086 (http://[wowza-ip-address]:8086/livestreamrecord).
- **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).

- **HTTPServerVersion** - (com.wowza.wms.http.HTTPServerVersion)
  Returns the Wowza Media Server version and build number. It's the default HTTP Provider on port 1935.

- **HTTPStreamManager** -
  (com.wowza.wms.http.streammanager.HTTPStreamManager)
  The Stream Manager HTTP Provider that's available through administrative port 8086 (http://[wowza-ip-address]:8086/streammanager).

- **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])
Extending Wowza Media Server Using Java

How do I extend Wowza Media Server?

Wowza Media Server® 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 Media Server includes a rich set of APIs to interact with and control the streaming process. See the Wowza Media Server 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 IDE, which is used to extend Wowza Media Server functionality. The downloadable 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 [install-dir]/lib folder. Modules can leverage third-party libraries or built-in Java functionality if the dependent .jar files are copied to the [install-dir]/lib folder. Modules are added to an application configuration by adding a <Module> entry to the <Modules> list in the application's Application.xml file.
Let's start by creating our first module. It'll have two methods: onAppStart and doSomething. The onAppStart method is an event method and the doSomething method is a custom method. The details of event methods and custom methods will be discussed later.

```java
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, add the following `<Module>` entry for this module to the end of the `<Modules>` list in the application's Application.xml configuration file:

```xml
<Module>
    <Name>MyModule</Name>
    <Description>This is MyModule</Description>
    <Class>com.mycompany.module.MyModule</Class>
</Module>
```

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 full path to the Java class that provides the module's functionality. We combine 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: Adobe® HDS (San Jose), Apple HLS (Cupertino), Microsoft Smooth Streaming, Dynamic Adaptive Streaming over HTTP (MPEG-DASH), and RTSP. Event methods are defined by the following interfaces:

- IModuleOnApp
- IModuleOnConnect
- IModuleOnStream
- IModuleOnHTTPSession
- IModuleOnRTPSession
- IModuleOnHTTPCupertinoStreamingSession
- IModuleOnHTTPSmoothStreamingSession
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 `<Modules>` entries are listed in `Application.xml`. 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**

```java
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.

**IModuleOnConnect**

```java
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.
- **onDisconnected**: 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**

```java
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 `Media Stream Listeners (IMediaStreamActionNotify3)` for more information about how to implement a server listener that's invoked when actions occur on this `IMediaStream` object.
IModuleOnHTTPSession

```java
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

```java
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

```java
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

```java
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

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

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

**IModuleOnHTTPCupertinoEncryption**

```java
public void onHTTPCupertinoEncryptionKeyRequest(
    HTTPStreamerSessionCupertino httpSession, IHTTPRequest req,
    IHTTPResponse resp);
```

```java
public void onHTTPCupertinoEncryptionKeyCreateVOD(
    HTTPStreamerSessionCupertino httpSession, byte[] encKey);
```

```java
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**

```java
public void onHTTPSmoothStreamingPlayReadyCreateVOD(
    HTTPStreamerSessionSmoothStreamer httpSession,
    PlayReadyKeyInfo playReadyKeyInfo);
```

```java
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.

**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 `<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 Media Server will search upwards in the <Modules> list, find the next module that implements the play method, and then 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:

```java
package com.mycompany.module;
import com.wowza.wms.module.*;
import com.wowza.wms.client.*;
import com.wowza.wms.request.*;
import com.wowza.wms.amf.*;
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:
Adobe Flash Player and Custom Methods

Parameters passed from Adobe Flash Player to Wowza Media Server 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 server API Javadocs ([install-dir]/documentation/serverapi) and the Server Side Coding example ([install-dir]/examples/ServerSideModules).

The following example shows how to convert three incoming parameters:

```java
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)
    {
        getLog().info("onCall: "+handlerName);
    }
}
```

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 server API Javadocs ([install-dir]/documentation/serverapi) and the Server Side Coding example ([install-dir]/examples/ServerSideModules).
The following example shows how to return simple value types from three methods:

```java
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:

```java
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:

```java
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:

```java
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:

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

Java Management Extensions (JMX)

All modules instantiated for a given application instance are made available through the Java Management Extension (JMX) Interface. More information about JMX can be found at the JMX Technology Home Page.

The path to the modules section in the MBean interface is:

```
WowzaMediaServerPro/VHosts/[vHostName]/Applications/[applicationName]/
  ApplicationInstance/[applicationInstanceName]/Modules
```

All public methods and properties (wrapped in Java Bean Get/Set methods) are made available through the `Instance` object found in each module definition. If you want to exclude a method or property from the JMX interface, import the `com.wowza.util.NoMBean` class and add the `@NoMBean` annotation to your method definition. This makes your custom
modules instantly available through the Wowza Media Server administration interface without any additional programming. All property values can be inspected, properties with `get[property-name]` accessors can be changed, and methods with simple Java types can be invoked through JConsole or VisualVM.

**HTTP Provider Classes**

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

```java
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.setStatusCode(404);
```

More complex and interesting HTTP Providers examples can be found on the [HTTP Providers](http://www.wowza.com) webpage.

**Event Listeners**

You can add event listeners to many points in the Wowza Media Server 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 in Wowza Media Server](http://www.wowza.com).
Server Administration

**How do I configure, manage, and deploy Wowza Media Server?**

Wowza Media Server® is a powerful Java server. It's configured through a set of XML files. The server can be run standalone from a command shell or installed as a system service. Running the server standalone is best for developing custom Wowza Media Server 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 Media Server 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).

**Note**

- If you want to get an SSL certificate from Wowza for use with Wowza Media Server 3.0 or later, 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 Media Server 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, Wowza Media Server 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 [install-dir]/logs.

For more information about log messages, scenarios that may cause these messages, and suggestions for resolution, see How to troubleshoot using Wowza Media Server log messages.

Logging Fields

Wowza Media Server 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                                                   |
### Logging Events

Wowza Media Server can generate the following logging events.

<table>
<thead>
<tr>
<th>Event name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>announce</td>
<td>RTSP Session Description Protocol (SDP) ANNOUNCE</td>
</tr>
<tr>
<td>app-start</td>
<td>Application instance start</td>
</tr>
<tr>
<td>app-stop</td>
<td>Application instance shutdown</td>
</tr>
<tr>
<td>comment</td>
<td>Comment</td>
</tr>
<tr>
<td>connect</td>
<td>Connection result</td>
</tr>
<tr>
<td>connect-burst</td>
<td>Connection accepted in burst zone</td>
</tr>
<tr>
<td>connect-pending</td>
<td>Connection pending approval by application and license manager</td>
</tr>
<tr>
<td>create</td>
<td>Media or data stream created</td>
</tr>
<tr>
<td>decoder-audio-start</td>
<td>Audio decoding has started for a transcoded stream</td>
</tr>
<tr>
<td>decoder-audio-stop</td>
<td>Audio decoding has stopped for a transcoded stream</td>
</tr>
<tr>
<td>decoder-video-start</td>
<td>Video decoding has started for a transcoded stream</td>
</tr>
</tbody>
</table>

*Copyright © 2006 - 2014 Wowza Media Systems, LLC. All rights reserved.*
Logging Status Values

Wowza Media Server can generate the following logging status values.

<table>
<thead>
<tr>
<th>Status value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Pending or waiting (for approval)</td>
</tr>
<tr>
<td>200</td>
<td>Success</td>
</tr>
<tr>
<td>302</td>
<td>Rejected by application with redirect information</td>
</tr>
<tr>
<td>400</td>
<td>Bad request</td>
</tr>
<tr>
<td>401</td>
<td>Rejected by application</td>
</tr>
<tr>
<td>413</td>
<td>Rejected by license manager</td>
</tr>
<tr>
<td>500</td>
<td>Internal error</td>
</tr>
</tbody>
</table>

Logging Configuration

Logging for Wowza Media Server 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 Media Server:

```properties
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.layout.QuoteFields=false
log4j.appender.stdout.layout.Delimeter=space

# Access appender
log4j.appender.serverAccess=org.apache.log4j.DailyRollingFileAppender
log4j.appender.serverAccess.DatePattern='.'yyyy-MM-dd
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.layout.QuoteFields=false
log4j.appender.serverAccess.layout.Delimeter=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/wowzamediaserver_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.Delimeter=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.

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

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](https://log4j.org).

Wowza Media Server 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:

```bash
[install-dir]/logs/[vhost]/[application]/wowzamediaserver_access.log
[install-dir]/logs/[vhost]/[application]/wowzamediaserver_error.log
[install-dir]/logs/[vhost]/[application]/wowzamediaserver_stats.log
```

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

```bash
[install-dir]/logs/[vhost]/wowzamediaserver_access.log
[install-dir]/logs/[vhost]/wowzamediaserver_error.log
[install-dir]/logs/[vhost]/wowzamediaserver_stats.log
```

This method for generating log files can be very useful if you want to offer Wowza Media Server as a shared service to several customers.
Server Management Console and Monitoring

How do I manage and monitor Wowza Media Server?

Wowza Media Server® can be managed and monitored through a Java Management Extensions (JMX) interface. JMX is a standards-based technology for exposing Java application components through a unified object interface. This interface can then be consumed by open source and commercial monitoring tools such as HP OpenView, OpenNMS, JConsole, and VisualVM.

For more information about the JMX interface, see the JMX Technology Homepage.

For more information about the JMX standard, see the Java Management Extensions (JMX) webpage.

Note

Most Java Runtime Environment (JRE or JVM) vendors require that you install the full Java Development Kit (JDK) to get the JConsole management and monitoring application. See your vendor's documentation for more information.

Local Management Using JConsole

Wowza Media Server exposes a rich set of objects for monitoring the server. The Java virtual machine also exposes a set of JMX objects that can be used to monitor the virtual machine. The easiest way to view these objects is to use the JConsole applet that ships with the Java Development Kit (JDK) of most popular VMs. This tool is usually located in the bin folder that's created by the JDK installation.
By default, startup.bat and startup.sh are configured to expose the JMX object interface to a locally running copy of JConsole. To view the JMX interface, start Wowza Media Server, and then run JConsole. In JConsole, you should see a list of the currently running Java virtual machines that expose a JMX interface. Wowza Media Server will be listed as `com.wowza.wms.bootstrap.Bootstrap start`. Select this item, and then click Connect.

You can then explore the different tab panels that are included in JConsole. Wowza Media Server management objects are located under the MBeans tab in the WowzaMediaServerPro group. JMX object organization is based on the configured virtual hosts, applications, and applications instances. Monitoring objects are created and deleted on the fly as applications, application instances, client connections, and streams are created and deleted from the server.

**Note**

In Windows, for security reasons, local monitoring and management is only supported if your default Windows temporary directory is on a file system that supports setting permissions on files and directories (for example, on an NTFS file system). It's not supported on a FAT file system that provides insufficient access controls. The workaround is to configure remote monitoring. For more information about how to configure the remote JMX interface, see Remote Management.

**Remote JMX Interface Configuration**

By default, the startup and service scripts are configured to only expose the JMX interface to a locally running monitoring application. You can also configure a remote JMX interface for monitoring Wowza Media Server from a remote computer. Both the JVM and Wowza Media Server include remote JMX interfaces. It's only necessary to configure one of these remote interfaces to enable remote monitoring. We recommend that you use the Wowza Media Server remote interface because it's easier to configure and can be properly exposed through hardware or software based firewalls.

**Wowza Media Server built-in JMX interface configuration**

The remote JMX interface that's built-in with Wowza Media Server can be configured through the JMXRemoteConfiguration and AdminInterface containers in the [install-dir]/conf/Server.xml file. This section describes the configuration settings:

- JMXRemoteConfiguration - Enable, IpAddress, RMIServerHostName, RMIConnectionPort, RMIRegistryPort

The Enable setting is a Boolean value that can be either true or false. It's the "switch" to enable and disable the remote JMX interface. The default value is false. Setting this value to...
true (with no further modifications to the other settings) enables the remote JMX interface with authentication. The default user name/password is admin/admin. The URL for invocation in JConsole or VisualVM is:

```
```

The IpAddress and RMIServerHostName settings work together to properly expose the JMX interface to the network. In general, the IpAddress value should be set to the internal IP address of the Wowza Media Server and the RMIServerHostName value should be set to the external IP address or domain name of the machine. For example, if the Wowza Media Server is behind a network-translated IP address (NAT), such that the internal IP address of the server is 192.168.1.7 and the external IP address is 40.128.7.4, the two settings should be as follows:

```
<IpAddress>192.168.1.7</IpAddress>
<RMIServerHostName>40.128.7.4</RMIServerHostName>
```

With this configuration, you would use the following URL to connect to the JMX interface:

```
service:jmx:rmi://40.128.7.4:8084/jndi/rmi://40.128.7.4:8085/jmxrmi
```

The RMICconnectionPort and RMIRegistryPort settings control the TCP ports that are used to expose the RMI connection and RMI registry interfaces. These values should only be changed if Wowza Media Server reports port conflicts during startup. The default values for these settings are 8084 and 8085 respectively. The RMICconnectionPort corresponds to the first port number in the connection URL and the RMIRegistryPort to the second.

The IpAddress, RMICconnectionPort, and RMIRegistryPort settings effect the connection URL in the following way:

```
service:jmx:rmi://[RMIServerHostName]:[RMICconnectionPort]/jndi/rmi://[RMIServerHostName]:[RMIRegistryPort]/jmxrmi
```

If the remote JMX interface is enabled, Wowza Media Server will log the URL of the currently configured JMX interface when it starts. This is probably the most reliable way to determine the JMX URL to use to connect to the server.

To enable remote JMX monitoring through software- or hardware-based firewalls, open TCP communication for the two ports defined by the RMICconnectionPort and RMIRegistryPort settings.

JMXRemoteConfiguration - Authenticate, PasswordFile, AccessFile

The Authenticate setting is a Boolean value that can be either true or false. It's the "switch" to enable and disable remote JMX interface authentication. The PasswordFile and AccessFile settings are the full path to the JMX password and access files respectively.

The password file is a text file with one line per user. Each line contains a user name followed by a space followed by a password. The access file contains one line per user. Each line
contains a user name followed by the **readwrite** or **readonly** access permission identifier. A sample password file (**jmxremote.password**) and a sample access file (**jmxremote.access**) can be found in the conf directory of the Wowza Media Server installation.

These files define three named users:

<table>
<thead>
<tr>
<th>User</th>
<th>Password</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
<td>readwrite</td>
</tr>
<tr>
<td>monitorRole</td>
<td>admin</td>
<td>readonly</td>
</tr>
<tr>
<td>controlRole</td>
<td>admin</td>
<td>readwrite</td>
</tr>
</tbody>
</table>

**Note**

Some Java Runtime Environments (JREs) require that both the password and access files have read-only privileges. On Linux operating systems, this can be achieved by setting the permissions on both files to **600**:  
chmod 600 conf/jmxremote.access  
chmod 600 conf/jmxremote.password

**JMXRemoteConfiguration** - **SSLSecure**

The **SSLSecure** setting is a Boolean value that can be either **true** or **false**. It's the "switch" to enable and disable the remote JMX interface over SSL. SSL configuration can be quite involved. For more information about how to enable SSL with JMX, see Using SSL.

**AdminInterface/ObjectList**

The **AdminInterface/ObjectList** setting is a comma-separated list of object types that you can expose through the JMX interface. This list can contain any number of the following items:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Server-level connection and performance info and notifications</td>
</tr>
<tr>
<td>VHost</td>
<td>Information about currently running virtual hosts</td>
</tr>
<tr>
<td>VHostItem</td>
<td>Details of currently configured virtual hosts</td>
</tr>
<tr>
<td>Application</td>
<td>Application-level connection and performance info</td>
</tr>
<tr>
<td>ApplicationInstance</td>
<td>Application Instance-level connection and performance info</td>
</tr>
<tr>
<td>Module</td>
<td>Details of currently loaded modules</td>
</tr>
<tr>
<td>MediaCaster</td>
<td>Details of media caster object (live stream repeater)</td>
</tr>
<tr>
<td>Client</td>
<td>Details of each connected Flash session</td>
</tr>
<tr>
<td>MediaStream</td>
<td>Details of each individual server-side NetStream object</td>
</tr>
<tr>
<td>SharedObject</td>
<td>Details of currently loaded shared objects</td>
</tr>
<tr>
<td>Acceptor</td>
<td>Details of currently running host ports or TCP ports</td>
</tr>
<tr>
<td>IdleWorker</td>
<td>Details of currently running idle workers</td>
</tr>
</tbody>
</table>

**Exposing Client, MediaStream, and/or SharedObject** information can add significant load to the server and to the JMX interface. You'll most likely want to turn off this level of detail in your deployed solution.

**Note**
When running Wowza Media Server as a Windows service, the JMX interface isn't available unless the service is running as a named user. To configure the service to run as a named user, do the following:

1. Open the Services console (Start > Control Panel > Administrative Tools > Services).
2. Right-click the Wowza Media Server service, and then select Properties.
3. On the Log On tab, change the Log on as option to This account, and then enter a user name and password for a local user.

Remote Management

Remote Management Using JConsole

JConsole can be used to monitor a remote Wowza Media Server. After you configure the remote JMX interface (see Remote JMX Interface Configuration), run JConsole and then enter the remote JMX interface URL in the Remote Process field. The default remote JMX interface URL for the JMX interface that's built-in with Wowza Media Server is:

```
```

Enter your user name and password into the provided fields and then click Connect. This will enable you to connect to the remote server and view the JMX hierarchy.

Remote Management Using VisualVM

VisualVM is another great tool for monitoring Wowza Media Server over JMX. VisualVM can be downloaded from the VisualVM webpage.

After you get VisualVM installed and running, it's best to install the MBean plugin. To do this, select the Plugins command from the Tools menu. Then on the Available Plugins tab, put a checkmark next to the VisualVM-MBean plugin and click Install. The MBean plugin provides similar information to JConsole. You can select Add JMX Connection from the File menu to add your Wowza Media Server to the Applications list.

Object Overview

This section describes the more important top-level objects that can be used to monitor Wowza Media Server performance and uptime. This section doesn't cover every object that's exposed by the server. These objects are available under the WowzaMediaServer object in the MBeans section of JConsole and VisualVM.
Server

The server object contains information about when the server was started and how long it has been running.

VHosts

The VHosts collection includes information about each of the running virtual hosts. From here, you get access to each of the running applications and applications instances. At each level of the hierarchy (Server, VHost, Application, ApplicationInstance), you can get detailed information about the number of connections (Connections object) and the input/output performance (IOPerformance object).

IOPerformance

The Server exposes IOPerformance objects at many different levels of the object hierarchy. These objects can be used to monitor server performance and throughput at that section of the server. For example, the IOPerformance object under a particular VHost displays the throughput of that particular virtual host.

Connections

The Server exposes Connections objects at many different levels of the object hierarchy. These objects can be used to monitor client connections at that section of the server. For example, the Connections object under a particular Application object displays the current number of clients that are connected to that particular application.

VHost/[vHostName] - HandlerThreadPool, TransportThreadPool

The HandlerThreadPool and TransportThreadPool objects expose information about each of the worker thread pools that are owned by each of the virtual hosts. You can use this object to monitor thread usage and load.
ServerNotifications

The **ServerNotifications** object publishes notification events related to the connection limits and connection bursting capabilities of Wowza Media Server. Wowza Media Server can generate the following notification events:

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.wowza.wms.connect.WarningServerLicenseLimit</td>
<td>connection accepted in bursting zone (warning)</td>
</tr>
<tr>
<td>com.wowza.wms.connect.ErrorServerLicenseLimit</td>
<td>connection refused due to license limit</td>
</tr>
<tr>
<td>com.wowza.wms.connect.WarningVHostLimit</td>
<td>connection refused due to virtual host limit</td>
</tr>
</tbody>
</table>

The body of the JMX notification message is a string with information about the virtual host, application, application instance, client ID, IP address, and referrer that generated the event. To view notification events in JConsole, navigate to the MBeans tab, open the WowzaMediaServer group, and then select the ServerNotification object. Then click Subscribe on the Notifications tab. All events are displayed as new rows in the Notifications list. Only events that occur after you subscribe to notifications are displayed.
Virtual Hosting

How do I let multiple users share my server running Wowza Media Server?

Wowza Media Server® can be configured to run multiple virtual host environments. Each of these virtual host environments has its own set of configuration files, application folders, and log files. This enables a single server to serve multiple users in separate environments. By default, the server is configured with a single virtual host named _defaultVHost_.

Configuration Files

This section describes the VHosts.xml configuration file in the Wowza Media Server conf directory. The VHosts.xml file is used to define each of the virtual host environments. The following items are required to define a virtual host.

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

Typical Configuration

A typical VHosts.xml file for a virtual host environment contains two virtual hosts: vhost1 and vhost2:

```xml
<Root>
  <VHosts>
    <VHost>
      <Name>vhost1</Name>
      <ConfigDir>/home/vhosts/vhost1/ConfigDir</ConfigDir>
      <ConnectionLimit>0</ConnectionLimit>
    </VHost>
  </VHosts>
</Root>
```
The directory structure for the virtual hosts in the above example would be:

```
[/home/vhosts]
  [vhost1]
  [applications]
  [conf]
  Application.xml
  Authentication.xml
  CEACaptionConversion.xml
  DVR.xml
  HTTPStreamers.xml
  LiveStreamPacketizers.xml
  LiveStreamTranscoders.xml
  MediaCasters.xml
  MediaReaders.xml
  MediaWriters.xml
  MP3Tags.xml
  RTP.xml
  StartupStreams.xml
  Streams.xml
  TimedTextProviders.xml
  VHost.xml
  admin.password
  publish.password
  [content]
  [keys]
  [logs]

[vhost2]
[applications]
[conf]
  Application.xml
  Authentication.xml
  CEACaptionConversion.xml
  DVR.xml
  HTTPStreamers.xml
  LiveStreamPacketizers.xml
  LiveStreamTranscoders.xml
  MediaCasters.xml
  MediaReaders.xml
  MediaWriters.xml
  MP3Tags.xml
  RTP.xml
  StartupStreams.xml
  Streams.xml
  TimedTextProviders.xml
  VHost.xml
  admin.password
  publish.password
  [content]
  [keys]
  [logs]
```
The virtual host configuration process is simple. Virtual hosts are defined in the `VHosts.xml` file in the conf directory. Each virtual host gets its own configuration directory structure that contains application, conf, and logs directories. Each virtual host gets its own set of configuration files.

It’s important to note that Wowza Media Server only supports IP address/port-based virtual hosting. It doesn't support domain name-based virtual hosting. In `VHost.xml`, each virtual host must define `HostPort` entries with unique IP address and port combinations that don’t conflict with other virtual hosts that are defined on a given server. The following combinations represent valid virtual host port configurations:

```
vhost1:
  <HostPort>
    <IpAddress>192.168.1.2</IpAddress>
    <Port>1935</Port>
  </HostPort>

vhost2:
  <HostPort>
    <IpAddress>192.168.1.2</IpAddress>
    <Port>1936</Port>
  </HostPort>

-or-

vhost1:
  <HostPort>
    <IpAddress>192.168.1.2</IpAddress>
    <Port>1935</Port>
  </HostPort>

vhost2:
  <HostPort>
    <IpAddress>192.168.1.3</IpAddress>
    <Port>1935</Port>
  </HostPort>
```

Virtual hosts can be added, modified, and deleted on the fly, without stopping and restarting the server, through the JMX interface and the `VHosts.xml` configuration file. To access virtual host operations through JConsole (with the server running), start JConsole, select the `MBeans` tab, open the `WowzaMediaServer` group, and select the `Server` object. Virtual host operations can be found in the `Operations` group. The following operations are of special interest:

- `startVHost` - start an individual virtual host by name
- `stopVHost` - stop an individual virtual host by name
- `reloadVHostConfig` - reload the `VHosts.xml` configuration file
To add a new virtual host without restarting the server, in `VHosts.xml`, add a new virtual host definition and then copy and configure a new set of configuration files as described above. Next, open JConsole and do the following:

1. Click the **MBeans** tab.
2. In the Contents pane, navigate to and expand the **Server** object.
3. Expand **Operations** and click **reloadVHostConfig**.
4. In the **Operation invocation** page, click the **reloadVHostConfig** button to reload the `VHosts.xml` file.
5. In the Contents pane, click **startVHost**.
6. In the **Operation invocation** page, enter the name of the new virtual host in the textbox next to the **startVHost** button, and then click the button. The new virtual host will be started immediately.
Examples & AddOn Packages

What do the examples do and where can I get AddOn Packages?

Wowza ships with examples that highlight the server functionality. Wowza also provides AddOn packages that extend and enhance Wowza Media Server functionality. This chapter describes the available examples and AddOns.

Examples

Wowza Media Server includes 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. This section describes the examples that are included in the Wowza Media Server installation.

Note

All Adobe® Flash® examples are implemented using ActionScript 3.0.

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.

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

SHOUTcast

This Adobe Flash example illustrates how to re-stream SHOUTcast MP3 or AAC+ audio data through Wowza Media Server. 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.

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.

WebcamRecording

This Adobe Flash example illustrates how to implement Wowza Media Server'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.

AddOn Packages

Wowza provides AddOn packages that extend and enhance Wowza Media Server functionality. Some of the AddOns are built-in with Wowza Media Server while others must be downloaded and installed. Premium AddOn packages require you to purchase a specific license, depending on the Server Edition being used while other AddOn packages are free of charge. This section briefly describes the available AddOns.
Note

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

Premium AddOns

Wowza Transcoder

Wowza Transcoder AddOn is a premium AddOn package that provides the ability to ingest a live stream (channel) and transrate it to multiple outbound bitrates. It can also decode the video and audio from an incoming channel, and then re-encode the stream into multiple bitrates, frame sizes, and profiles to suit the desired playback devices. Transcoding of multiple channels is possible on the same server instance. For more information about how to license and use this AddOn, see the Wowza Transcoder AddOn webpage.

Wowza nDVR

Wowza nDVR AddOn is a premium AddOn package that provides the ability to record a live stream into a cache on Wowza Media Server while allowing users to play or pause the live stream, rewind to a previously recorded point, or resume viewing at the current live point. Customization is possible through XML configurations and the available APIs. For more information about how to license and use this AddOn, see the Wowza nDVR AddOn webpage.

Wowza DRM

Wowza DRM AddOn is a premium AddOn package that provides encryption key exchange integration with third-party Digital Rights Management (DRM) platforms. Using these keys, Wowza Media Server performs on-the-fly encryption for live and video on demand content. For live workflows, per-stream encryption is available with the ability to rotate keys. For on-demand workflows, per-asset and per-session encryption is available with the ability to rotate keys. Live and video on demand key rotation support is also available for Apple HTTP Live Streaming (HLS). For more information about how to license and use this AddOn, see the Wowza DRM AddOn webpage.

Note

Currently, integration is supported for the following Key Management Systems:

- BuyDRM™ KeyOS™. Supports Microsoft® PlayReady® protected Apple HLS and Microsoft Smooth Streaming playback with BuyDRM players on iOS-based devices (iPhone/iPad) and Windows® Phone devices.
• EZDRM. Supports Smooth Streaming playback with Silverlight clients.
• Verimatrix®. Supports VCAS™ protected Apple HLS playback with ViewRight® clients on iOS-based devices, Android™ devices, PCs, and set-top boxes and PlayReady® protected Smooth Streaming playback with Smooth Streaming clients.

**Note**

Wowza Media Server has APIs that enable encryption schemes for on-the-fly encryption of live and on-demand 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). In addition, Wowza Media Server has an API that enables on-the-fly encryption of live and on-demand 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](#)

**Free AddOns**

**Wowza StreamLock**

Wowza StreamLock™ AddOn is a security option for network encryption from Wowza®. It provides near-instant provisioning of free 256-bit Secure Sockets Layer (SSL) certificates to verified Wowza users for use with Wowza Media Server. 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 (Daily and Monthly) and Perpetual licensees running Wowza Media Server 3.0 or later. It's not available in the Trial and Developer editions of Wowza Media Server. 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 Media 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 edger servers. The edge servers communicate with one or more load-balancing Wowza Media 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 (ModuleGeoIPLock).

GoCoder

The Wowza® GoCoder™ app is a live audio and video application for Apple iPod touch® (iOS 4.0 or later) and iPhone® (3GS or later) that allows content providers to encode live content right from their iOS device and deliver it to Wowza Media Server in real time over 4G, 3G, and Wi-Fi systems. For more information, see How to use Wowza GoCoder video broadcasting iOS app with Wowza Media Server.

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.

MediaCache

This AddOn package is a read-through caching mechanism that enables scaling of video on demand streams. For more information, see How to get MediaCache AddOn (scale video on demand streaming).

Push Publishing

This AddOn package has an API that enables streams to be pushed from a Wowza Media Server running Wowza Media Server 3.5 (or later) to downstream Wowza Media Servers.
running Wowza Media Server 3.5 (or later) using the WOWZ™ protocol (Wowza messaging protocol). It also enables streams to be pushed from a Wowza Media Server running Wowza Media Server 3.5 (or later) to downstream Wowza Media Servers (all versions), Adobe® Media Servers, and 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 get Push Publishing AddOn (push to CDNs and other services).

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.
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.

<table>
<thead>
<tr>
<th>Tutorial name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to set up video on demand streaming</td>
<td>Describes how to configure an application to stream video on demand (VOD) content to all supported player technologies.</td>
</tr>
<tr>
<td>How to set up live streaming using an RTMP-based encoder</td>
<td>Describes how to publish a live stream from RTMP-based encoders to Wowza Media Server® and how to configure an application to deliver the live stream to all supported player technologies.</td>
</tr>
<tr>
<td>How to set up live streaming using an RTSP/RTP-based encoder</td>
<td>Describes how to publish a live stream from RTSP/RTP-based encoders to Wowza Media Server and how to configure an application to deliver the live stream to all supported player technologies.</td>
</tr>
<tr>
<td>How to set up live streaming using a native RTP encoder with SDP file</td>
<td>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.</td>
</tr>
<tr>
<td>How to publish and play a live stream (MPEG-TS based encoder)</td>
<td>Describes how to use a live encoder that publishes a stream using the MPEG-2 Transport Stream (MPEG-2 TS) protocol to</td>
</tr>
<tr>
<td><strong>How to set up and run Wowza Transcoder AddOn for live streaming</strong></td>
<td>Describes how to configure an application to use Wowza Transcoder AddOn.</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>How to set up and run Wowza nDVR for live streaming</strong></td>
<td>Describes how to configure an application to use Wowza nDVR AddOn.</td>
</tr>
<tr>
<td><strong>How to configure a live stream repeater</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>How to re-stream video from an IP camera</strong></td>
<td>Describes how to re-stream and play a live stream from an IP camera.</td>
</tr>
<tr>
<td><strong>How to re-stream audio from SHOUTcast/Icecast</strong></td>
<td>Describes how to re-stream and play live SHOUTcast® or Icecast® audio streams.</td>
</tr>
<tr>
<td><strong>How to set up live video recording</strong></td>
<td>Describes how to configure an application for video recording using Flash Player.</td>
</tr>
<tr>
<td><strong>How to set up live video chat</strong></td>
<td>Describes how to configure an application for video chat using Flash Player.</td>
</tr>
<tr>
<td><strong>Closed Captioning Overview</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>MPEG-DASH Overview</strong></td>
<td>Describes how to configure video on-demand and live applications in Wowza Media Server to deliver streams to Dynamic Adaptive Streaming over HTTP (DASH) clients. Wowza Media Server 3.6 adds basic support for MPEG-DASH streaming in a preview version of MPEG-DASH technology. Not all of the features that are defined in the MPEG-DASH standard are currently supported.</td>
</tr>
<tr>
<td><strong>Wowza DRM AddOn Overview</strong></td>
<td>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</td>
</tr>
<tr>
<td>Microsoft Smooth Streaming delivery to a wide range of devices including set-top boxes (STBs), smart TVs, and smartphones and tablets.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td><strong>Media Security Overview</strong></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>