Wowza Media Server® 2

User's Guide

Wowza Media Server 2:
User's Guide

Third-Party Information

This document contains links to third-party websites that are not under the control of Wowza Media Systems, Inc. ("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 trademarks of Wowza Media Systems, Inc., and may be registered in the United States or in other jurisdictions including internationally.

Adobe and Flash are registered trademarks of Adobe Systems Incorporated, and may be registered in the United States or in other jurisdictions including internationally.

Silverlight is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

QuickTime, iPhone and iPod 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 trademarks, service marks or trade names of other entities and may be registered in certain jurisdictions including internationally.

Third Party Copyright Notices

Log4j and Mina: Copyright © 2006 The Apache Software Foundation

Java ID3 Tag Library and JLayer 1.0 (classic): Copyright © 1991, 1999 Free Software Foundation, Inc.

Java Service Wrapper: Copyright © 1999, 2006 Tanuki Software, Inc.

Bouncy Castle Crypto API: Copyright © 2000 – 2008, The Legion Of The Bouncy Castle

Table of Contents

Chapter 1

Introduction

What is Wowza Media Server 2?

Wowza Media Server 2 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 Server can deliver content to many popular media players such as Adobe's® Flash® Player, Microsoft's Silverlight® player, Apple's iPhone™ and iPod® touch and Apple's QuickTime® player, among others. Wowza Media Server 2 includes support for many streaming protocols including the Real-Time Messaging Protocol (RTMP), Microsoft Smooth Streaming, Apple HTTP Live Streaming, Real-Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP), MPEG2 Transport Streams (MPEG-TS) and more. It is an alternative to the Adobe Flash Media Server products (FMIS and FMSS), Apple Streaming Server (Darwin) and other media servers.

For the most up to date information, tutorials and tips, visit our online forums at:

http://www.wowzamedia.com/forums

To get started quickly with Wowza Media Server 2 see the Quick Start Guide included with the Wowza Media Server 2 software installer and also available online at:

http://www.wowzamedia.com/resources.html

Real-Time Messaging Protocol (Adobe Flash Player)

Wowza Media Server 2 communicates with the Adobe Flash player using the Real-Time Messaging Protocol (RTMP). Wowza Server can deliver multi-bitrate live and on-demand media, data and remote procedure call information to and from the Flash player using RTMP. It supports media streaming as well as other features such as: Shared Objects, video recording, video chat, remote procedure calls and more. Wowza Media Server 2 supports all video and audio formats that the Flash player supports:

Video

- H.264
- VP6
- SorensonSpark
- Screen Shared codec

Audio

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

Wowza Server supports five variants of the protocol: RTMP, RTMPE (encrypted RTMP), RTMPT (tunneling), RTMPTE (encrypted RTMPT) and RTMPS (RTMPT over SSL). RTMP is the base protocol and is the most efficient and fastest of the five variants. RTMPT is a tunneling variant of the RTMP protocol that can be used to tunnel through firewalls that employ stateful packet inspection. RTMPE and RTMPTE are encrypted variants of the RTMP and RTMPT protocols that secure the data being transmitted between the Flash player and Wowza Media Server. Wowza Server includes bi-directional support for Action Message Format (AMF) AMF3 and AMF0 for data serialization (AMF3 was introduced in Flash Player 9 and ActionScript 3.0).

Apple HTTP Live Streaming (iPhone, iPod touch and QuickTime)

Wowza Media Server 2 can stream multi-bitrate live and video on demand H.264 (baseline level 3.0 or lower), AAC and MP3 content to the iPhone/iPod touch (version 3.0 OS or greater), QuickTime player (version 10 or greater) and Safari browser (version 4.0 or greater) using the Apple HTTP Live Streaming protocol. Apple HTTP Live 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 Server. Wowza Server supports the encrypted version of the Apple HTTP Live Streaming protocol which uses a 128-bit version of the Advanced Encryption Standard (AES-128). Apple HTTP Live Streaming is referred to in the Wowza Server documentation and configuration files as Cupertino Streaming. As of the writing of this document the iPhone and iPod touch devices support the following media formats:

Video

- H.264 (Baseline profile level 3.0 or below)

Audio

- AAC, AAC Low Complexity (AAC LC), High Efficiency AAC v1 (HE-AAC)
- MP3

Note

The iPhone and iPod touch do not support a combination of H.264 video and MP3 audio in the same stream. The iPhone and iPod touch do not support High Efficiency AAC v2.

Microsoft Smooth Streaming (Microsoft Silverlight)

Wowza Media Server 2 can stream multi-bitrate live and video on demand H.264, AAC and MP3 content to the Microsoft Silverlight player using the Smooth Streaming protocol. Microsoft Silverlight is 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 Server so there is no need for an IIS 7 server.

The following media formats can be used when streaming to the Silverlight player using Wowza Server:

Video

- H.264

Audio

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

Real-Time Streaming Protocols (QuickTime, VLC, Mobile Devices, Set-top Boxes)

Wowza Media Server 2 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 MPEG2 Transport Stream protocol (MPEG-TS). This includes players and devices such as QuickTime Player, VideoLAN VLC player, set-top boxes and mobile devices. Wowza Server can also accept incoming streams from encoding devices that use these same protocols. Wowza Server supports RTP and MPEG-TS in and out over UDP as well as Multicast. In addition, Wowza Server supports interleaved RTSP/RTP (which is where the RTP portion of the stream

flows over the RTSP TCP connection) which enables RTSP/RTP to be delivered in network environment that do not allow UDP transmission.

Wowza Server supports the following RTSP, RTP and MPEG-TS specifications:

RTSP	rfc2326
RTP: H.264	rfc3984, QuickTime Generic RTP Payload Format
RTP: AAC	rfc3640, rfc3016, ISO/IEC 14496-3
RTP: MP3	rfc2250
RTP: Speex	rfc5574
MPEG-TS	ISO/IEC 13818-1
MPEG-TS over RTP	rfc2038

Video and Audio Streaming, Recording and Chat

Wowza Media Server 2 can stream live and video on demand content to many different player technologies. Wowza Media Server 2 supports the following video on demand file formats: FLV (Flash Video - .flv), MP4 (QuickTime container - .mp4, .f4v, .mov, .m4v, .mp4a, .3gp, and .3g2) and MP3 content (.mp3). Wowza Server can accept live video and audio streams from encoders that support the following protocols; RTMP, RTSP/RTP, native RTP and MPEG-TS. Wowza Server can record any incoming live stream to either the Flash Video (FLV) or MP4 (QuickTime container) format.

Wowza Media Server 2 can be used to re-stream SHOUTcast and Icecast (MP3, AAC and AAC+) audio streams as well as IP Camera streams (H.264, AAC and MP3) to the supported player technologies. Wowza Server will maintain a single connection back to the original source stream while delivering the stream to multiple players. Wowza Server is also able to forward the embedded SHOUTcast and Icecast metadata such as song title and artist to the Adobe Flash player client as metadata. The SHOUTcast example that ships with Wowza Server illustrates these capabilities.

Wowza Media Server 2 can deliver two-way video, audio and text chat to the Adobe Flash player. This feature can be leveraged to deliver video conferencing applications or two-way messaging applications.

Extending the Server

Wowza Media Server 2 is built using Java technology. The server can be extended by writing custom Java classes that are dynamically loaded at runtime. Server extensions (also referred to as modules) run at the full speed of the server. The server includes a rich API to interact with and control the streaming process. Wowza Server ships with several example server extensions. See the chapter **Extending Wowza Server Using Java** for more detailed information and the **Wowza Media Systems Forums** for many code examples.

Adobe Flash Player Features

Wowza Media Server 2 includes support for two Adobe Flash specific features; Remote Shared Objects (RSO) and bi-directional remote procedure calls. Remote Shared Objects are an extension of ActionScript objects that enables the synchronization of shared object data between

Flash players on the same or different client machines. Shared data is synchronized by the Wowza Server server through an event based synchronization method. RSO's can also be persisted on the server to maintain data across sessions.

Bi-directional remote procedures calls are a way for ActionScript code running in the Flash player to invoke methods and pass data to Wowza Server. Wowza Server can in turn invoke methods and pass data to the Flash player. This enables rich client/server applications to be built using the Flash player and Wowza Server.

Server Architecture

Wowza Media Server 2 is a pure Java server. It is written in Java and can be extended dynamically using custom Java classes. Wowza Server can be deployed in any environment that supports the Java 6 virtual machine or later. Wowza Server is fully 64-bit compliant. It is architected to be highly multi-threaded and can take full advantage of multi-core hardware. All logging is done using the log4j logging component and utilizes the W3C Extended Common Log Format (ECLF).

Wowza Media Server 2 was architected from the ground up to handle multiple streaming protocols. The server side API is designed to make it easy to control the streaming process of each of the supported streaming protocols and player technologies. Streaming is controlled through the creation and configuration of a streaming application. A single application can be configured to simultaneously deliver live or video on demand content to multiple player technologies.

Wowza Media Server 2 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 2 Editions

Wowza Media Server 2 comes in five editions: Developer, Evaluation, Software Subscription, Perpetual and Wowza Media Server 2 for Amazon EC2.

Developer edition	The Developer and the Subscription/Perpetual editions differ only in the number of concurrent connections the server can handle (10 and unlimited respectively), streaming time duration limits on selected protocols (limited and unlimited respectively), and licensing rights (see the Wowza EULA for more information); all other functionality is exactly the same
Evaluation edition	The Evaluation edition provides the same functionality as the Subscription/Perpetual editions but is limited to 30 days of use and other restrictions apply as described in the Evaluation EULA Addendum
Subscription and Perpetual editions	Subscription and Perpetual editions differ only by licensing terms (see the Wowza EULA for more information); all other functionality is exactly the same.
Wowza Media Server 2 for Amazon EC2 edition	Subscription edition but under different licensing terms (see the Wowza EULA and the Wowza Media Server 2 for Amazon EC2 EULA, respectively, for more information). The Wowza Media Server 2 for Amazon EC2 edition is a pre-configured version of Wowza Media Server 2 running in the Amazon Elastic Computing Cloud (EC2) environment (see the following web page for more information: http://www.wowzamedia.com/ec2.php).

Chapter

2

Server Installation

How do I install Wowza Media Server 2?

Wowza Media Server 2 is a small and powerful Java server. Below are the instructions needed to choose the correct version of Java and install and run Wowza Server.

Before Installation

Wowza Media Server 2 is a Java 6 (aka 1.6) application. To run, it requires the installation of a Java 6 or greater runtime environment (JRE). To develop server side applications, a Java Development Kit (JDK) version 6 or later is required. The server also implements a Java Management Extensions (JMX) interface that can be used to manage and monitor the server. One of the more popular JMX consoles is JConsole, which ships with the JDK.

So what does this all mean? If you are developing server side applications or are going to monitor a local or remote Wowza Server, you need to install Java Development Kit version 6 (aka 1.6) or greater. If you are simply deploying Wowza Server for production use, then you need only install a Java runtime environment version 6 (aka 1.6) or greater. We recommend installing the most recent version of the Java JDK or JRE for you platform.

Note

We suggest that you deploy Wowza Media Server 2 under the most recent version of either the Java Development Kit (JDK) or Java Runtime Environment (JRE) available on your platform. On the Windows platform the Java Runtime Environment does not include the **server** runtime environment (which is explained in the tuning instructions). This environment is included with the Java Development Kit. For this reason when running on Windows, we suggest installing the JDK.

Once you have your Java environment installed and configured, you can validate that it is 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 is equal to or greater than 1.6.

Note

The Support section of the Wowza Media Systems website contains additional information and links to help with obtaining the correct Java environment and tools for your platform. You can visit this site at: http://www.wowzamedia.com.

Note

Wowza Media Server 2 on the Windows platform uses the JAVA_HOME environment variable to determine the location of the Java environment under which to run. If you have problems starting Wowza Server on Windows, double check to be sure the JAVA_HOME variable is pointing to a Java 6 (aka 1.6) or greater Java environment. And when making changes or upgrades to your Java environment that may affect the installation path 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 2 is installed using an installer. On Linux, Solaris and other Unix based platforms, the software is installed using a self extracting binary installer. These are available for download at:

http://www.wowzamedia.com/store.html

Windows

To install Wowza Media Server 2 on Windows, double-click the installer file and follow the instructions on the screen. During the installation process you will be asked to enter the product serial number. You cannot proceed with the installation until you have entered a valid serial number. There is information below on how to change your serial number if you need to upgrade your server license.

To uninstall, choose **Uninstall Wowza Media Server** from the **Start>Programs>Wowza Media Server 2** menu.

Mac OS X

To install Wowza Media Server 2 on Mac OS X, mount the disk image (double-click .dmg) file, double-click the installer package (.pkg) file and follow the instructions on the screen. Files will be installed to the following locations.

```
/Applications/Wowza Media Server 2  - server startup/shutdown scripts &
                                      documentation
/Library/WowzaMediaServer           - server application files and
                                      folders: applications, bin, conf,
                                      content, examples, lib and logs
/Library/LaunchDaemons              - background service script
                                      com.wowza.WowzaMediaServer.plist
/Library/Receipts                   - installer receipt file
                                      WowzaMediaServer-2.0.0-
                                      preview7.pkg
```

The first time you run the server in standalone mode you will be asked to enter your serial number. The serial number is stored in the file **/Library/WowzaMediaServer/conf/Server.license**. There is information below on how to change your serial number if you need to upgrade your server license.

To uninstall, throw the following folders and files into the trash.

```
folder:     /Applications/Wowza Media Server 2
folder:     /Library/WowzaMediaServer-2.0.0
symlink:    /Library/WowzaMediaServer
file:       /Library/LaunchDaemons/com.wowza.WowzaMediaServer.plist
file:       /Library/Receipts/WowzaMediaServer-2.0.0.pkg
```

Linux

To install on Linux systems follow the steps below:

Red Hat Package Manager Systems

```
sudo chmod +x WowzaMediaServer-2.0.0.rpm.bin
sudo ./WowzaMediaServer-2.0.0.rpm.bin
```

To uninstall:

```
sudo rpm –e WowzaMediaServer-2.0.0
```

Debian Package Manager Systems

```
sudo chmod +x WowzaMediaServer-2.0.0.deb.bin
sudo ./WowzaMediaServer-2.0.0.deb.bin
```

To uninstall:

```
sudo dpkg –-purge wowzamediaserver
```

You will be asked to agree to the **End User License Agreement**. The package manager will extract and install the files in the **/usr/local/WowzaMediaServer-2.0.0** directory. The server will be installed as the root user. The first time you run the server in standalone mode you will be asked to enter your serial number. The serial number is stored in the file **/usr/local/WowzaMediaServer/conf/Server.license**. There is information below on how to change your serial number if you need to upgrade your server license.

Other Linux and Unix Systems

To install the server on other Linux and Unix based systems, such as Solaris, open a terminal window. Download **WowzaMediaServer-2.0.0.tar.bin** to any directory, and execute the self extracting installer:

```
sudo chmod +x WowzaMediaServer-2.0.0.tar.bin
sudo ./WowzaMediaServer-2.0.0.tar.bin
```

You will be asked to agree to the **End User License Agreement**. The self-extracting installer will install the files in the **/usr/local/WowzaMediaServer-2.0.0** directory. The server will be installed as the root user. The first time you run the server in standalone mode you will be asked to enter your serial number. The serial number is stored in the file **/usr/local/WowzaMediaServer/conf/Server.license**. There is information below on how to change your serial number if you need to upgrade your server license.

To uninstall:

```
cd /usr/local
rm –rf WowzaMediaServer-2.0.0
```

Starting and Stopping the Server

Windows: Standalone

On Windows, Wowza Media Server 2 can be started in standalone mode from the **Start** menu: All **Programs>Wowza Media Server 2**>Wowza Startup/Shutdown.

The server cannot also be started from a DOS command prompt. To do this, open a DOS command prompt and execute the following commands:

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

Windows: Service

To start the server as a Windows service, open the **Settings>Control Panel>Administrative Tools>Services** administrative tool and locate the **Wowza Media Server 2** entry in the list. Next, right click on the entry and select **Start** from the context menu. To stop the server select **Stop** from the same context menu. To configure the service to run each time Windows restarts, select **Properties** from the right click context menu, set **Startup type** to **Automatic** and click the **OK** button to close the dialog.

Note

By default the Windows service is running under the **Local System Account**. This can limit how Wowza Media Server 2 can interact with the underlying operating system. For example you might not be able to connect to Wowza Server using JConsole/JMX or you may have issues streaming content from UNC paths. To address these issues, modify the service to run as a named user in the **Log On** tab of the service properties dialog.

Mac OSX: 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 2** or by opening a **Terminal** window and entering the following commands:

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

Mac OSX: Service

To start the server as a Mac OS X launchd service, open a **Terminal** window and enter:

```
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 then enter the following commands:

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

To stop the server enter:

./shutdown.sh

Linux: Service

To start the server as a Linux service, open a command prompt and enter one of these two commands (it differs based on your Linux distribution):

```
/sbin/service WowzaMediaServer start
```

or

```
/etc/init.d/WowzaMediaServer start
```

To stop the service, enter one of these two 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. Please consult your Linux manual if these instructions do not apply to your Linux distribution.

Note

The Linux services script subsystem does not use the full $PATH definition to determine the location of Linux commands. It uses what is known as the **init** path. This can lead to an issue on Linux distributions where the default installation location for Java cannot be found by applying the **init** path. See this forum post for more information:

http://www.wowzamedia.com/forums/showthread.php?t=1511

Entering a New Serial Number

Wowza Media Server 2 stores serial number information in the following file (on each of the platforms):

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

To change the serial number, edit this file using a text editor and enter the new serial number. Upon next launch of the standalone server, the last four digits of the serial number will be displayed in the console window.

Ports Used For Streaming

Before streaming with Wowza Media Server 2 it is important that you open the following ports on your firewall. The table below represents the defaults ports Wowza Server uses for streaming.

All of these port numbers are configurable through the configuration files described later in this document.

TCP 1935	**RTMP/RTMPT/RTMPE/RTSP-interleaved Streaming**
UDP 6970-9999	**RTP UDP Streaming**
TCP 8084-8085	**JMX/JConsole Monitoring and Administration**
TCP 8086	**Administration**

By default Wowza Media Server 2 is configured to only use TCP port 1935 for streaming. You may want to configure additional ports for streaming such as TCP port 80 for HTTP or RTMPT or TCP port 554 for RTSP streaming. To add an additional ports using a text editor, edit **[install-dir]/conf/VHost.xml** and duplicate the **<HostPort>** entry for port 1935 (be sure to get the entire XML section starting with **<HostPort>** and ending with **</HostPort>**) and change the **<Port>** value to the desired port. Wowza Server cannot share ports with other programs or services. So be sure there are no other programs or services running that share the added ports. Below is a table of common ports used for streaming:

TCP 80	**RTMPT, Smooth Streaming, Cupertino Streaming**
TCP 443	**RTMPS**
TCP 554	**RTSP**

Server Configuration and Tuning

Wowza Media Server 2 is configured through a set of XML, configuration and properties files in the **[install-dir]/conf** folder. These configuration files are read during server startup. The configuration files can be directly edited using a standard text editor. Below is a brief explanation of each of the configuration files:

Server Configuration Files

Server.xml - General Server configuration
VHosts.xml - Define virtual hosts
log4j.properties - Logging configuration

Virtual Host Configuration Files

Authentication.xml - RTSP and HTTP authentication configuration
HTTPStreamers.xml - Cupertino Streaming and Smooth Streaming configuration
LiveStreamPacketizers.xml - HTTP packetization configuration
MediaCasters.xml - MediaCaster (SHOUTcast, Live Repeater…) 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
VHost.xml - Virtual host configuration

Application Configuration Files

Application.xml - Application configuration

The **Configuration Reference** document that accompanies this User's Guide contains detail information on each of these configuration files.

The settings associated with the Java runtime environment, such as the command used to invoke Java and the maximum Java heap size, are controlled through a set of scripts and configuration files. The location of these files differs depending on platform and the method used to invoke the server. Below is a description of each of these files.

bin\ setenv.bat (Windows)

The bin\setenv.bat is invoked when the server is started from the command line. The most important settings in this file are:

```
set _EXECJAVA=java            # Command used to invoke java
set JAVA_OPTS="-Xmx768M"      # Command line options for java command
```

bin\WowzaMediaServer-Service.conf (Windows)

The bin\WowzaMediaServer-Service.conf is the configuration file used when the server is invoked as a Windows service. The most important settings in this file are:

```
wrapper.java.command=java     # Command used to invoke java
wrapper.java.initmemory=3     # Initial Java Heap Size (in MB)
wrapper.java.maxmemory=768    # Maximum Java Heap Size (in MB)
```

/Library/WowzaMediaServer/bin/setenv.sh (Mac OS X)

The bin/setenv.sh is invoked when the server is started in standalone and service mode. The most important settings in this file are:

```
_EXECJAVA=java          # Command used to invoke java
JAVA_OPTS="-Xmx768M"    # Command line options for java command
```

/usr/local/WowzaMediaServer/bin/setenv.sh (Linux)

The bin/setenv.sh is invoked when the server is started in standalone mode. The most important settings in this file are:

```
_EXECJAVA=java          # Command used to invoke java
JAVA_OPTS="-Xmx768M"    # Command line options for java command
```

Note

It is very import that Wowza Server be tuned properly so that it can take best advantage of the available hardware resources. The default tuning of the server is sufficient for application development, but it not ideal for productions use. Without proper tuning, the server under medium to heavy load will run out of resources and will stop working properly. The **General Tuning Guide** resides in the **Wowza Media Systems Forums** so that it can be kept up to date:

http://www.wowzamedia.com/forums/showthread.php?t=1320

Chapter 3

Application Configuration

How do I create and configure an application for streaming?

All streaming in Wowza Media Server 2 is controlled through the creation and configuration of an application. 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 folder:

[install-dir]/applications/myapplication

A single application can be configured to deliver a live or video on demand stream to the Adobe Flash player, the Silverlight player, an Apple iPhone or iPod touch device and an RTSP/RTP based player at the same time (with the exception of video on demand delivery to RTSP/RTP which is not supported in the current version). The **Quick Start Guide** contains basic tutorials with the step by step instructions on how to configure an application for the more common streaming tasks. The remainder of this chapter will cover the details of application configuration. For more detailed configuration information see the **Configuration Reference** document that accompanies this document.

Applications and Application Instances (Application.xml)

As seen above an application is created by creating a named folder in **[install-dir]/application**. The name of the application is the name of the folder. An **Application.xml** file defines the configuration for a given application. An application instance is an instantiation of an application and provides a name space and context for streaming. An application instance is started dynamically and a single application may have multiple named application instances all running at the same time. 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 need to 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 and provides separation and a name space for each room.

Application configuration is defined in an **Application.xml** file. When an application instance is loaded, it looks in the following two 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 located will be used.

Note

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

URL Formats

All streaming in Wowza 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 the Flash player, Silverlight, RTSP/RTP or the iPhone all follow a similar format:

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

Where:
[protocol]:           - streaming protocol (rtmp, rtsp, http …)
[address]:            - address of the server running Wowza 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 the stream name does not contain any path elements and the default application instance name is to be used the URL can be shortened to:

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

Below are example URLs for the different player technologies. This example assumes we are streaming the live video with the stream name **myStream** using the application name **live**.

Adobe Flash Player:
```
Server: rtmp://mycompany.com/live
Stream: myStream
```

Apple iPhone or iPod touch:
```
http://mycompany.com:1935/live/myStream/playlist.m3u8
```

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

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

Now is probably a good time to take a quick look at the default **Application.xml** file. Use a text editor to edit the default **Application.xml** file. The rest of this chapter covers the more commonly configured items in this file.

Stream Types

Wowza Media Server 2 uses named stream types to control the different types of streaming (live, video on demand, chat, remote recording…). Stream types are configured using the **Streams/StreamType** property in **Application.xml**. Stream types are defined in **[install-dir]/conf/Streams.xml**. Below is a list of the stream types and their uses:

Stream Type	Use
default	Video on demand
file	Video on demand
record	Video recording
live	Publish and play live video content (best for one-to-many streaming of live events)
live-lowlatency	Publish and play live video content (best for one-to-one or one-to-few video/audio chat applications)
live-record	Same as **live** in addition content will be recorded
live-record-lowlatency	Same as **live-lowlatency** in addition content will be recorded
shoutcast	Audio re-streaming of a SHOUTcast/Icecast MP3 or AAC+ audio stream
shoutcast-record	Same as **shoutcast** in addition content will be recorded
liverepeater-origin	Publish and play live video content across multiple Wowza Media Server servers in an origin/edge configuration (use to configure origin application)
liverepeater-origin-record	Same as **liverepeater-origin** in addition content will be recorded
liverepeater-edge	Publish and play live video content across multiple Wowza Servers in an origin/edge configuration (use to configure edge application)
liverepeater-edge-lowlatency	Publish and play live video content across multiple Wowza Servers in an origin/edge configuration (use to configure edge application when latency is important)
liverepeater-edge-origin	Publish and play live video content across multiple Wowza Servers in an origin/edge/edge configuration (use to configure an middle-edge application)
rtp-live	Re-streaming of an RTSP/RTP, native RTP or MPEG-TS stream
rtp-live-lowlatency	Re-streaming of an RTSP/RTP, native RTP or MPEG-TS stream when latency is important)
rtp-live-record	Same as **rtp-live** in addition content will be recorded
rtp-live-record-lowlatency	Same as **rtp-live-lowlatency** in addition content will be recorded

Each stream type exposes properties that are used for tuning of the stream type. For example, the stream type definition for **live** and **live-lowlatency** only differ in the tuning which is 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** of the **live-lowlatency** stream type the <**Stream>** section of the **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>
```

HTTPStreamers and LiveStreamPacketizers

The **<HTTPStreamers>** setting in **Application.xml** controls if the streams in the defined application (live or video on demand) are made available for playback to the iPhone/iPod touch and Microsoft Silverlight players. **HTTPStreamers** can contain none, one or more the following values (separated by commas): **cupertinostreaming** and **smoothstreaming**. If the **cupertinostreaming** value is present then stream is available for playback by the iPhone and iPod touch (as well as with an appropriate version of QuickTime/Safari on Mac OS). If the **smoothstreaming** is present then the stream is available for playback by Microsoft Silverlight.

The **<LiveStreamPacketizers>** setting works in a similar fashion but only applies to live streams. It controls how live streams are packetized for delivery to the the HTTP streaming technologies. **LiveStreamPacketizers** can contain none, one or more of the following values (separated by commas):

LiveStreamPacketizers	Description
cupertinostreamingpacketizer	Cupertino: iPhone and iPod touch
smoothstreamingpacketizer	Smooth: Microsoft Silverlight
cupertinostreamingrepeater	Cupertino: Live stream repeater for iPhone/iPod touch
smoothstreamingrepeater	Cupertino: Live stream repeater for Microsoft Silverlight

You would set the packetizer with a repeater value when using the server in an Origin/Edge configuration. This is described later in this document in the section titled **Live Stream Repeater (Multiple Server 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 the modules to be loaded for a given application. Many AddOn Packages provide additional functionality through the use of modules. The details of modules are discussed in the **Server-side Modules** chapter.

A basic module definition looks like this:

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

Each module must have a unique name. The **<Description>** information is for providing a detailed description of the module and is not used in any operations. The **<Class>** item is the full path to the Java class that is providing 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 into a **.jar** file that is copied to the **[install-dir]/lib** folder. The Wowza Server comes with many modules that can be added to the **<Modules>** list to provide additional functionality. See the **Built-in Modules** section for a complete list. You can also use the **Wowza IDE** to develop your own custom modules to provide additional functionality. See the **Extending Wowza Server Using Java** chapter for more information.

Note

The Wowza Integrated Development Environment (Wowza IDE) is a free tool available for download at:

http://www.wowzamedia.com/labs.html

Properties

The default **Application.xml** file contains several different **<Properties>** containers that can be used to add or override property values within Wowza Server. Properties are a list of name/value pairs that provide a means for tuning and modifying the default configuration of the Wowza Server. Properties can also be used server-side as a means to pass data to custom modules from **Application.xml**. You will see in this document, the **Wowza Media Server Forums** and the **Quick Start Guide** references to individual properties. There currently is not a comprehensive document that lists all the available properties. A property definition has the following form:

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

Where **<Name>** is the property name, **<Value>** is the property value and **<Type>** is the property type. Valid property types are: String, Integer, Boolean, Double and Long. It is important when tuning the server to be sure to add properties to the correct container. The

instructions for tuning will always specify which **<Properties>** container a property should be added to for tuning.

Media Types

Media types are not configured in **Application.xml** but are an important part of streaming. Wowza Media Server 2 supports many different media types. Wowza Server can read the following media or file types: **FLV** (Flash Video - .flv), **MP4** (QuickTime container - .mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2 …), **MP3** content (.mp3) and **SMIL** (Synchronized Multimedia Integration Language - .smil). Media types are specified through 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 default media type prefix if none is specified is **flv:**. Below is the table of the supported media type prefixes:

Media type prefix	Description
flv:	Flash Video (default if no prefix specified)
mp4:	QuickTime container
mp3:	MP3 file
id3:	MP3 file (return only ID3 tag information)
smil:	Synchronized Multimedia Integration Language

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

Content Storage

By default Wowza Media Server 2 is setup to stream video on demand content and record to the **[install-dir]/content** folder. You can easily change this behavior by editing an application's **Application.xml** file and changing the value of **Streams/StorageDir**. For example to setup an Application to use an application specific content folder you might change this value to:

${com.wowza.wms.context.VHostConfigHome}/applications/ ${com.wowza.wms.context.Application}/content

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

```
${com.wowza.wms.AppHome}                    - Application home directory
${com.wowza.wms.ConfigHome}                 - Configuration home directory
${com.wowza.wms.context.VHost}              - Virtual host name
${com.wowza.wms.context.VHostConfigHome}    - Virtual host config directory
${com.wowza.wms.context.Application}        - Application name
${com.wowza.wms.context.ApplicationInstance} - Application instance name
```

Chapter 4

Streaming Tutorials

Where do I get step-by-step instructions?

The **Quick Start Guide** and **Wowza Media Server Forums** contain tutorials that include step by step instructions for common streaming scenarios. The **Quick Start Guide** provides basic instructions to get you up and running quickly. The **Wowza Media Server Forums** provide more in-depth and up-to-date instructions and information. Below is a brief description of each of the streaming scenarios with a link to the online forums and quick start guide.

How to play a video on demand file

This tutorial describes how to stream video on demand files.

Quick Start Guide:
How to play a video on demand file

Wowza Media Server Forums (includes multi-bitrate instructions):
How to play a video on demand file

How to publish and play a live stream (RTMP or RSTP/RTP based encoder)

This tutorial describes how to publish and play a live stream when using an encoder that supports either the Real-time Messaging Protocol (RTMP) or the RTSP Announce Method. Examples of encoders that support RTMP publishing are: Telestream Wirecast, On2 Flix Live and Orban Opticodec. Examples of encoders that support the RTSP Announce Method are: Telestream Wirecast and QuickTime Broadcaster.

Quick Start Guide:
How to publish and play a live stream (RTMP or RSTP/RTP based encoder)

Wowza Media Server Forums (includes multi-bitrate instructions):
How to publish and play a live stream (RTMP or RSTP/RTP based encoder)

How to publish and play a live stream (MPEG-TS based encoder)

This tutorial describes how to publish and play a live stream when using an encoder that supports MPEG2 Transport Streams (MPEG-TS). Examples of encoder vendors that sell products that support MPEG-TS publishing are: HaiVision, Digital Rapids and ViewCast.

Quick Start Guide:
How to publish and play a live stream (MPEG-TS based encoder)

Wowza Media Server Forums (includes multi-bitrate instructions):
How to publish and play a live stream (MPEG-TS based encoder)

How to publish and play a live stream (native RTP encoder with SDP file)

This tutorial describes how to publish and play a live stream when using an encoder that supports Real-time Transport Protocol (native RTP). Examples of encoder vendors that sell products that support native RTP publishing are: HaiVision, Digital Rapids, ViewCast and Telestream.

Quick Start Guide:
How to publish and play a live stream (native RTP encoder with SDP file)

Wowza Media Server Forums (includes multi-bitrate instructions):
How to publish and play a live stream (native RTP encoder with SDP file)

How to re-stream video from an IP camera

This tutorial describes how to re-stream and play a live stream from an IP camera.

Quick Start Guide:
How to re-stream video from an IP camera

Wowza Media Server Forums:
How to re-stream video from an IP camera

How to re-stream audio from SHOUTcast/Icecast

This tutorial describes how to re-stream and play a live SHOUTcast or Icecast audio stream.

Quick Start Guide:
How to re-stream audio from SHOUTcast/Icecast

Wowza Media Server Forums:
How to re-stream audio from SHOUTcast/Icecast

How to setup video chat application

This tutorial describes how to setup an application for video chat using the Adobe Flash player.

Quick Start Guide:
How to setup video chat application

Wowza Media Server Forums:
How to setup video chat application

How to setup video recording application

This tutorial describes how to setup an application for remove recording using the Adobe Flash player.

Quick Start Guide:
How to setup video recording application

Wowza Media Server Forums:
How to setup video recording application

Chapter 5

Advanced Configuration Topics

How do I take advantage of Wowza Server's 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 2. Because of this several of these advanced topics will include a brief overview with a link to the AddOn Package. A list of available AddOn packages can be found here:

http://www.wowzamedia.com/packages.html

MediaCasters, Stream Manager and StartupStreams.xml

Wowza Media Server 2 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 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 made available to the player. When the last viewer of the stream stops watching a given stream the MediaCaster system waits a timeout period. If no other players request the stream then the stream is stopped and the stream is no longer available for streaming until another request comes in for the streams.

This methodology works great for the Adobe Flash player and RTSP/RTP streaming where there is no need for advanced packetization. For HTTP Streamers such as Cupertino streaming and Smooth Streaming the pull model does not work. The iPhone requires about 30 seconds of video to be pre-packetized before it can begin playback. Microsoft Silverlight requires three times the key frame duration. For this reason it is necessary to start the stream prior to the stream being ready for streaming to these player technologies. There are two methodologies for starting a stream that uses the MediaCaster system and keeping it running; **Stream Manager** and the **StartupStreams.xml**.

The **Stream Manager** is a web based tool for starting and stopping MediaCaster streams on the fly that is built into Wowza Media Server 2. To startup the Stream Manager do the following:

1. Edit **[install-dir]/conf/admin.password** and enter a new line with a username and password. For example to add the username **myuser** with the password **mypassword** the contents of this file should look like:

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

2. Open a web browser and enter the URL:

   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 startup the stream, select the MediaCaster type, type in the stream name and click, OK. To stop a stream, click the **[stop-receiving-stream]** link next to stream name. You can reset a stream by clicking on the **[reset-receiving-stream]** link.

The second method for starting MediaCaster streams is using the **StartupStreams.xml** file. Stream entries in this file are automatically started when the server is started (or more specifically when a virtual host is started). The **StartupStreams.xml** is a list of application, media caster types and stream names. The format of a single entry is:

```
<StartupStream>
      <Application>live</Application>
      <MediaCasterType>rtp</MediaCasterType>
      <StreamName>rtsp://192.168.1.7:554/mycoolstream.sdp</StreamName>
</StartupStream>
```

Live Stream Repeater (Multiple Server Live Streaming)

The following example illustrates a suggested configuration and implementation for delivering a live media event across multiple Wowza Media Server 2 servers. We will walk through the configuration and deployment of the live stream repeater. The live stream repeater uses multiple Wowza Servers in an origin and edge configuration to deliver live media content across multiple servers. The encoded media content will be delivered to the origin server in the same manner as if you were delivering the content to a single Wowza Server. The player will request the content from an edge server and the edge server will maintain a single connection per-unique stream to the origin. Origin and edge configuration is an application level configuration. A single Wowza Server instance can be configured as an origin for one application and an edge for another.

For this example we will setup a single origin server using the application name **liverepeater**. Here are the steps to configure the origin server:

1. Create a folder named **[install-dir]/applications/liverepeater**.

2. Create a folder named **[install-dir]/conf/liverepeater** and copy the file **[install-dir]/conf/Application.xml** into this new folder.

3. Edit the newly copied **Application.xml** file and make the following changes:

    a. Change the **Streams/StreamType** to **liverepeater-origin**

    b. Change the **LiveStreamPacketizers** to:
    cupertinostreamingpacketizer,smoothstreamingpacketizer

Next, configure each of the edge servers as follows:

1. Create a folder named **[install-dir]/applications/liverepeater**.

2. Create a folder named **[install-dir]/conf/liverepeater** and copy the file **[install-dir]/conf/Application.xml** into this new folder.

3. Edit the newly copied **Application.xml** file and make the following changes:

    a. Change the **Streams/StreamType** to **liverepeater-edge** (you can use the **liverepeater-edge-lowlatency** stream type if low latency is important, this will add extra load to the server).

    b. Change the **LiveStreamPacketizers** to:
    cupertinostreamingrepeater,smoothstreamingrepeater

    c. Add the following property to the **MediaCaster/Properties** container (be sure to get the correct properties container – there are several in the file):

```
<Property>
       <Name>streamTimeout</Name>
       <Value>15000</Value>
       <Type>Integer</Type>
</Property>
```

    d. Uncomment the **Repeater/OriginURL** section and set **OriginURL** to rtmp URL of the origin server. For example if the origin server uses the domain name origin.mycompany.com, this value should be set to:

```
<Repeater>
       <OriginURL>rtmp://origin.mycompany.com</OriginURL>
       <QueryString></QueryString>
</Repeater>
```

For this example let's assume 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, edge3.mycompany.com. Let's also assume that we are going to use the stream name **mycoolevent**. The URLs for the players are as follows (assuming we are streaming off of edge1):

Adobe Flash Player:
Server: rtmp://edge1.mycompany.com/liverepeater
Stream: mycoolevent

Apple iPhone or iPod touch:
http://edge1.mycompany.com:1935/liverepeater/mycoolevent/playlist.m3u8

Microsoft Silverlight:
http://edge1.mycompany.com:1935/liverepeater/mycoolevent/Manifest

RTSP/RTP:
rtsp://edge1.mycompany.com:1935/liverepeater/mycoolevent

It is possible to configure more than one origin server to provide a hot backup in case the main origin server goes down. Let's say the failover origin server has the domain name origin2.mycompany.com. Assuming it is configured in the same manner as the main origin server, you would set the following Repeater/OriginURL in each the edge's Applications.xml files:

```
<Repeater>
       <OriginURL>rtmp://origin.mycompany.com|rtmp://origin2.mycompany.com</OriginURL>
       <QueryString></QueryString>
</Repeater>
```

Basically it's the two connection URLs concatenated together with the pipe (**|**) character. The edge servers will first try to connect to the first origin server, if this fails they will attempt to connect to the second origin server.

This example assumes you are using an encoder in which the stream name is a simple name and not a URL. If you are using an encoder such as an MPEG-TS encoder in which the stream name is not a simple stream name, then you can use **.stream** files on the origin to hide the complex stream names. For example if your complex stream name on the origin is udp://0.0.0.0:10000, use a text editor to create a file in the **[install-dir]/content** folder with the name **mycoolevent.stream** and set the contents to udp://0.0.0.0:10000. You then use **mycoolevent.stream** in place of **mycoolevent** in the URLs above to play the stream.

Note

The **Media Security** AddOn Package describes how to secure the connection between the origin and edge machines using SecureToken.

Note

If you are streaming to the iPhone/iPod touch or Microsoft Silverlight player and are using a non-push based encoder (native RTP or MPEG-TS) then you will need to use the Stream Manager to start and keep the stream running on the origin.

Note

To provide load balancing between the edge servers you can use the dynamic load balancing system referenced in the **Dynamic Load Balancing** section.

Live Stream Recording

The **VideoRecording** example that ships with Wowza Media Server 2 is a specialized way of remote recording of a live stream when using the Adobe Flash player. It uses the **record** stream type and special capabilities built into the Flash player to control the recording process. If you simply want to record an incoming live stream from an encoder, then there are two more general purpose methods to accomplish this; use one of the ***-record** stream types (such as **live-record**) or use the **LiveStreamRecord** AddOn package.

The ***-record** stream types are the simplest method but give you the least amount of control. If you use this method the entire duration of the published stream will be recorded. If the encoder starts and stops, the file will be versioned with a version number and a new file will start. You can control the container format used (FLV or MP4) by specifying a stream name prefix in the encoder. If the prefix **flv:** or not specified, then the stream will be recorded to an FLV container. If the prefix is **mp4:**, then the stream will be recorded to an MP4 (QuickTime) container. Remember that a MP4 container can only record H.264, AAC and MP3 media data. If you are recording using the Flash player the FLV container is the only option.

Another option is the **LiveStreamRecord** AddOn package. This package gives you more control over the recording process. The package includes a server side module that can be used to control the recording process (the source code is included). You can control when the recording starts and stop, the file name and location and the container format as well as other small details. The package is available for download here:

http://www.wowzamedia.com/forums/showthread.php?t=2733

Server-side Publishing (Stream and Publisher classes)

Wowza Media Server 2 includes two methods 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 regarding the Stream class see this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=5490

The Publisher class is a low level publishing API to provide the ability to inject raw compressed video and audio frames into Wowza Server to create a custom live stream. This API is quite new so at this time we do not have a lot of documentation and examples. We hope to have more information as the feature matures. See the Publisher classes server-side API (Javadoc documentation) for the current detailed documentation. There is also an audio example which walks through the process of publishing Speex data to a stream in this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=5717

Dynamic Load Balancing

The Dynamic Load Balancing AddOn package provides a method for dynamically load balancing between multiple Wowza Server edge servers. The edge servers communicate with one or more load balancing Wowza Server. You can then connect to the load balancing server to get the currently least loaded edge server. You can download the package from this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=4637

Media Security (SecureToken, authentication and encryption)

The MediaSecurity AddOn Package provides a set of server-side modules and methodologies for protecting streaming to the different player technologies. It includes a detailed tutorial for protecting streaming using SecureToken, authentication and encryption. You can download the package from this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=1281

Chapter 6

Adobe Flash Streaming and Scripting

What can I do with Wowza Server and the Adobe Flash player?

Wowza Media Server 2 includes additional features that are only applicable to the Adobe Flash player. When using Wowza Server with the Adobe Flash player, Wowza Server is much more then just a streaming server - it is an application server. It provides features such as shared objects, video chat, remote recording and bi-directional remote procedures calls. This chapter covers all of these topics.

Streaming Basics

We will start with the most basic code needed to play a live or video on demand stream in Flash. Let assume we have followed the instructions in the **How to play a video on demand file** tutorial in the Quick Start Guide and we have an application with the name **vod** that is setup for video on demand streaming. Do the following in Adobe Flash CS3 or CS4:

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

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

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

		ns.bufferTime = 3;

		video1.attachNetStream(ns);

		ns.play("mp4:Extremists.m4v");
	}
}
nc.addEventListener(NetStatusEvent.NET_STATUS, ncOnStatus);

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

7. Select **Control->Test Movie** from the menu.

You should be streaming the Extremists.m4v example file. This is the most basic ActionScript 3.0 code needed for live or video on demand playback. If you quickly inspect the code you can see how simple it is. We create a NetConnection object for streaming, add an event listener so that we get notified when the connection to Wowza Server is established and when we are notified of a successful connection we create a NetStream object and begin playback of the stream.

The **SimpleVideoStreaming** and **LiveVideoStreaming** example players that ship with Wowza Server take this example a little further. These example players support progress bars, pause, stop and full screen. Inspecting the code for these two examples is a good next step for learning how to stream. The VideoChat and VideoRecording examples are great starting point to learn how to publish video and audio using the built-in **Camera** and **Microphone** objects.

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. There are many options. We are going to cover a few of the more popular options Adobe FLVPlayback component, JW Player and FlowPlayer.

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 fullscreen. It comes with Adobe Flash CS3 and Adobe Flash CS4. From time to time the component is updated. It is best to keep your Adobe Flash software up to date to be sure you are running the most recent version. The nice thing about this component is that it can be integrated into your own custom Flash code.

JW Player is pre-built Flash based player offered by Long Tail Video. It includes a rich set of features such as playlists, skinning and ad serving. It is fully supported and there is a commercial option. It also includes a built-in version of the Wowza SecureToken security mechanism. You can download it from here:

http://www.longtailvideo.com

There are instructions on how to use it with Wowza Server in these forum posts:

http://www.wowzamedia.com/forums/showthread.php?t=182

http://www.wowzamedia.com/forums/showthread.php?t=1665

Another option is FlowPlayer which 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. You can download it from here:

http://flowplayer.org

There are instructions on how to use it with Wowza Server in this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=710

Bi-directional Remote Procedure Calls

Wowza Media Server 2 supports bi-direction remote procedure calls to and from the Adobe Flash player. What this means is from the Flash player you can call a server-side Java method and pass data to the Wowza Server and from the Wowza Server you can call a client-side ActionScript method and pass data to the Flash player. This is very useful when building rich Internet applications.

Calls from the Flash player to Wowza Server are performed using method:

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

For example, to call the server-side method **doSomething** and pass two parameters **value1** and **value2** and receive a single return value, the ActioniScript 3.0 client-side code looks like this (we will cover the server-side code for this method later in this document):

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

Receiving method calls from Wowza Server are done by adding handler methods/functions to the client object that is attached to the NetConenction object. For example to add the handler method **onSomethingHappended** that receives two string parameters **value1** and **value2**, the ActionScript 3.0 code looks like this:

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

We will go into more detail on the programming model in the chapter; **Extending Wowza Server Using Java**.

Remote Shared Objects

Wowza Media Server 2 supports Adobe Flash remote shared objects. Remote shared objects are a means for sharing data between Wowza Server and multiple Flash players. Each Flash player that subscribes to a shared object will be notified of updates to the shared object data. Shared object data can be changed client-side by a Flash player or server-side through the Wowza Server ISharedObject API. Below is an example of the ActionScript 3.0 code needed to create a remote shared object and set a value:

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

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

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

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

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

nc.addEventListener(NetStatusEvent.NET_STATUS, ncOnStatus);

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

The **RemoteSharedObjects** example that ships with Wowza Server is a more complete remote shared object example. We will go into more detail on the programming model in the chapter; **Extending Wowza Server Using Java.**

Chapter 7

Server-side Modules and Extensions

What is a server-side module and what server-side functionality ships with Wowza Media Server 2?

Much of the functionality delivered by Wowza Media Server 2 is done through server-side modules and HTTPProviders. Server-side modules are Java classes that are configured on a per-application basis and are loaded at application instance startup and provide much of the functionality needed to control the streaming process. HTTPProviders are Java classes that are configured on a per-virtual host basis and are light-weight HTTP servers that can be used to query server information. In this chapter we discuss each of these methods of extending Wowza Server and the built-in Java classes that are immediately available for use. In the next chapter we discuss how to create your own server-side extensions.

Sever-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 are callable from the Adobe Flash player. It is these methods that provide the play, publish, seek, pause and resume functionality needed to control the Flash player streaming process. Server-side modules can also be used to control iPhone/iPod touch, Microsoft Silverlight and RTSP/RTP streaming process as well. The details of how the API works are in the next chapter.

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:

```
<Modules>
	<Module>
		<Name>base</Name>
		<Description>Base</Description>
		<Class>com.wowza.wms.module.ModuleCore</Class>
	</Module>
	<Module>
		<Name>properties</Name>
		<Description>Properties</Description>
		<Class>com.wowza.wms.module.ModuleProperties</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 the **Built-in Server-side Module** section below. Creating custom server-side modules is covered in the next chapter.

HTTPProviders

HTTPProviders are mini HTTP servers that can be used to extend the functionality of Wowza Server and are configured on a per-port basis in **[install-dir]/conf/VHost.xml**. An individual HTTPProvider can be username and password protected. Multiple HTTPProviders can be attached to a single port and the HTTPProvider is selected based on request filter. An example HTTPProvider 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 to the class that implements the IHTTPProvider interface. The **RequestFilters** is a pipe (**|**) separated list of filters that control when this provider will be invoked based on the HTTP request path. For example, this above request filter will only be invoked if the path portion of the HTTP request URL starts with **streammanager**. The AuthenticationMethod is the authentication method used to control access to this HTTPProvider. Valid values are **admin-digest** and **none**. The **admin-digest** authentication method uses digest authentication (a challenge/response system to authenticate user – passwords are never sent in clear text) to control access to the HTTPProvider. The usernames and passwords for admin-digest authentication are stored in the file **[install-dir]/conf/admin.password**. The **none** method allows all access.

Creating custom HTTPProviders is covered in the next chapter.

Built-in Server-side Modules

Below is a list of each of the built-in server-side modules along with a brief description of the functionality that is provided. For detailed information on each of the methods provided in a module see the server-side API reference.

ModuleCore – (com.wowza.module.ModuleCore)

The ModuleCore module represents the server-side implementation of the Adobe Flash **NetConnection**, **NetStream** and **SharedObject** objects. It is required that this module be included by all applications for the server to operate properly. This module contains several additional server side methods that are highlighted here:

Function call	Description
`setStreamType(streamType:String);` `getStreamType();`	Returns and sets the default stream type for this client connection.
`setRepeaterOriginUrl(originUrl:String);` `getRepeaterOriginUrl();`	Returns and sets the live stream repeater origin URL to use for this connection.
`getStreamLength(streamName:String);` `getStreamLength(streamNames:Array);`	For video on demand streaming it returns the duration of the stream in seconds. If an array of stream names is passed in an array of durations is returned.
`getClientID();`	Returns the client ID for this client connection.
`getReferrer();`	Get the referrer from the onConnect method.
`getPageUrl();`	Get 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.

`FCSubscribe(streamName, [mediaCasterType]);` `FCUnSubscribe(streamName);`	When using the live stream repeater to lock and unlock a stream on the edge during streaming. This method is useful when doing dynamic streaming to lock all bitrate renditions of a live stream on an edge server to be sure they are available when a switch is made between bitrate renditions.

ModuleProperties - (com.wowza.module.ModuleProperties)

The ModuleProperties module gives the Flash player client code access to application specific properties (name, value pairs) that are attached to the objects in the server object hierarchy.

Function call	Description
`setApplicationProperty(name:String, value:String);` `getApplicationProperty(name:String);`	Returns and sets properties attached to this client's Application object.
`setAppInstanceProperty(name:String, value:String);` `getAppInstanceProperty(name:String);`	Returns and sets properties attached to this client's Application Instance object.
`setClientProperty(name:String, value:String);` `getClientProperty(name:String);`	Returns and sets properties attached to this client's object.
`setStreamProperty(streamId:Number, value:String);` `getStreamProperty(streamId:Number);`	Returns and sets properties attached to a NetStream object. NetStream objects are identified by StreamId which can be returned to the client by making a call to getLastStreamId() directly following a call to "new NetStream(nc)".

ModuleClientLogging - (com.wowza.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 the Flash player client:

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

Is the same as a server side call to:

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

ModuleFastPlay - (com.wowza.module.ModuleFastPlay)

The ModuleFastPlay enables fast forward, fast rewind and slow motion play back of static flash video. Fast play is configured by making a call to netStream.call("setFastPlay", null, multiplier, fps, direction) before each call to netStream.play, netStream.pause(false), netStream.seek. To turn off fast play simply make a call to netStream.play, netStream.pause(false), netStream.seek without first making a call to **setFastPlay**.

```
setFastPlay(multiplier:Number, fps:Number, direction: Number);
```

multiplier the speed at which to play the movie. To play a movie at 4x normal speed, set this value to 4.0. To play a movie in slow motion, set this value to a value less than one. For example to playback at quarter speed, set this value to 0.25.

fps the frames per second for the resultant video stream. During fast play the server will discard video frames as needed to try to maintain this frame rate. For slow motion (multipliers less than 1) this value is ignored.

Note

Fast play does not work properly with H.264/HE-AAC content.

Note

Remember that Flash video is made up of a series of key-frames and progressive-frames (D and P frames). During the fast play process the server is going to throw out mostly progressive-frames in favor of key-frames. Key-frames tend to be much larger than progressive-frame. Because of this you will want to specify a frames-per-second rate that is significantly lower than the movie's frame rate to maintain a reasonable bandwidth. So for a movie that normally plays at 30 fps a setting of 10fps is about right for fast play.

direction the direction of play. A value of 1 for forward and -1 for reverse.

During fast play the time value returned by NetStream.time needs to be shifted and scaled to reflect the current playback position in the movie. Each time fast play is initiated the NetStream object receives an onStatus(statusObj) event. Wowza Media Server has extended the statusObj to include information about the current fast play settings. The following properties have been added to the statusObj:

isFastPlay boolean that is true if fast play is on and false if not.

fastPlayMultiplier the multiplier specified in the call to setFastPlay.

fastPlayDirection the direction specified in the call to setFastPlay

fastPlayOffset the offset used to calculate the true location in the video stream.

With this information you can calculate the current playback position by executing the following calculation:

```
var inc:Number;
var time:Number;

inc = ((NetStream.time*1000)-fastPlayOffset)*fastPlayMultiplier;
time = (fastPlayOffset + (fastPlayDirection>0?inc:-inc))/1000;
```

Note

The example **FastPlayVideoStreaming** in the examples folder is a great starting point for discovering how to use fast play.

Note

When using the **file** or **default** stream type, fast play is not supported when a media playlist contains more than one entry.

ModuleFLVPlayback - (com.wowza.module.ModuleFLVPlayback)

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

Built-in HTTPProviders

Below is a list of each of the built-in HTTPProviders along with a brief description.

HTTPServerVersion - (com.wowza.wms.http.HTTPServerVersion)

HTTPServerVersion returns the Wowza Server version and build number. It is the default HTTPProvider on port 1935.

HTTPCrossdomain - (com.wowza.wms.http.HTTPCrossdomain)

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

HTTPClientAccessPolicy - (com.wowza.wms.http.HTTPClientAccessPolicy)

HTTPClientAccessPolicy serves up the Microsoft Silverlight **clientaccesspolicy.xml** file when present in **[install-dir]/conf** folder.

HTTPStreamManager - (com.wowza.wms.http.HTTPStreamManager)

HTTPStreamManager is the Stream Manager HTTPProvider that is available through administrative port 8086 (http://[wowza-ip-address]:8086/streammanager).

HTTPServerInfoXML - (com.wowza.wms.http.HTTPServerInfoXML)

HTTPServerInfoXML return detailed server and connection information in XML format and is available through administrative port 8086 (http://[wowza-ip-address]:8086/serverinfo).

HTTPConnectionInfo - (com.wowza.wms.http.HTTPConnectionInfo)

HTTPConnectionInfo return detailed connection information in XML format and is available through administrative port 8086 (http://[wowza-ip-address]:8086/connectioninfo).

HTTPConnectionCountsXML - (com.wowza.wms.http.HTTPConnectionCountsXML)

HTTPConnectionCountsXML return connection information in XML format and is available through administrative port 8086 (http://[wowza-ip-address]:8086/connectioncounts).

Chapter

8

Extending Wowza Server Using Java

How do I extend the Wowza Server server?

Wowza Media Server 2 can easily be extended by writing Java classes that are loaded dynamically by the server. There are several integration points that can be used to extend the server; custom server-side modules, HTTPProviders and listeners. We will explore each of these integration points and provide a quick example. We provide a free integrated development environment called the Wowza IDE that can be used to extend the functionality of the server. You can download it from here:

http://www.wowzamedia.com/labs.html

It is probably best to download and install the Wowza IDE first before reading this chapter. The included documentation will walk you through the creation of your first custom server-side module. It will point you back to this chapter for more information. Consult the **Server-side API Guide** for detailed information on the available APIs. There is also a wealth of knowledge and code snippets online in the **Wowza Media Server Forums**.

Custom Module Classes

Server-side modules are Java class that are configured on a per-application basis and are dynamically created at application instance startup. Typically module classes are bound into **.jar** files that are located in the **[install-dir]/lib** folder. Modules can leverage any available 3rd party libraries or built-in Java functionality as long as the dependent **.jar** files are copied into 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 will 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.

```
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");
	}
}
```

Next, to add this module to an application configuration edit **Application.xml** for the application and add the following **<Module>** entry for this module to the end of the **<Modules>** list:

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

Each module must have a **<Name>** that is unique in that **<Modules>** list. The **<Description>** information is for providing a detailed description of the module and is not used in any operations. The **<Class>** item is the full path to the Java class that is providing 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 Flash, Microsoft Silverlight, iPhone/iPod touch and RTSP. Event methods are defined by the following interfaces:

```
IModuleOnApp
IModuleOnConnect
IModuleOnStream
IModuleOnHTTPSession
IModuleOnRTPSession
IModuleOnHTTPCupertinoStreamingSession
IModuleOnHTTPSmoothStreamingSession
IModuleOnHTTPCupertinoEncryption
```

All event methods defined in all modules are invoked when an event occurs. What this means is that if two modules implement the **onAppStart** event method, then both modules **onAppStart** methods will be invoked when a new application instance is created. Module methods are invoked starting at the top of the **<Modules>** list defined in **Application.xml**. So the first

<Modules> entry in the list will be called first and it will work its way up to the last item in the list. Below are each of the event method interfaces and their corresponding event methods.

IModuleOnApp

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

onAppStart: Invoked when an application instance is started
onAppStop: Invoked when an application instance is stopped

IModuleOnConnect

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

onConnect: Invoked when a Flash player connects to an application instance
onDisconnected: Invoked when a Flash player disconnect from an application instance
onConnectAccept: Invoked when a Flash player connection is accepted
onConnectReject: Invoked when a Flash player connection is refused

IModuleOnStream

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

onStreamCreate: Invoked when a new IMediaStream object is created
onStreamDestroy: Invoked when a IMediaStream object is closed

Note

The **onStreamCreate** event method is invoked before **play** or **publish** has been called for this IMediaStream object. For this reason the IMediaStream object does not have a name. See the IMediaStreamActionNotify2 interface to implement a server listener that is invoked when actions occur on this IMediaStream object.

IModuleOnHTTPSession

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

onHTTPSessionCreate: Invoked when HTTP streaming session(Cupertino or Smooth) created
onHTTPSessionDestroy: Invoked when HTTP streaming session(Cupertino or Smooth) closed

IModuleOnRTPSession

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

onRTPSessionCreate: Invoked when RTP session created
onRTPSessionDestroy: Invoked when RTP session closed

IModuleOnHTTPCupertinoStreamingSession

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

onHTTPCupertinoStreamingSessionCreate: Invoked when Cupertino session created
onHTTPCupertinoStreamingSessionDestroy: Invoked when Cupertino session closed

IModuleOnHTTPSmoothStreamingSession

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

onHTTPSmoothStreamingSessionCreate: Invoked when Smooth session created
onHTTPSmoothStreamingSessionDestroy: Invoked when Smooth session closed

IModuleOnHTTPCupertinoEncryption

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

onHTTPCupertinoEncryptionKeyRequest: Invoked when encryption key request is made for Cupertino streaming
onHTTPCupertinoEncryptionKeyCreateVOD: Invoked when encryption key is created for video on demand stream
onHTTPCupertinoEncryptionKeyCreateLive: Invoked when encryption key is created for live stream

Custom Methods

Custom methods are public mehods that you wish to expose to the Adobe Flash player through calls to the client-side interface NetConnection.call() or are 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 following argument signature (IClient, RequestFunction, AMFDataList params). Only public methods with this signature will be available to be called from the Flash player.

Processing for custom methods is different than that of event methods. When a given method such as **play** is invoked from the Flash player, only the last module in the **<Modules>** list that defines that custom method will be invoked. For example, the **ModuleCore** module defines the method **play** which is invoked when NetStream.play(streamName) is called from the 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 will be invoked rather than the **play** method defined in ModuleCore. If in your implementation of **play** you wish to invoke the **play** method of the next module up the list that precedes your module, you call **this.invokePrevious(client, function, params)**. Wowza Server will search up the module list and find the next module that implements the **play** method and it will 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 that implement that method above them in the **<Modules>** list.

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

```
package com.mycompany.module;

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

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

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

onCall

The onCall method is a catch-all for any methods that are undefined by custom methods. The interface for this method is defined in the **IModuleOnCall** interface class. The onCall method functions the same as an event method in that all onCall methods defined in all modules will be called. Example:

```
package com.mycompany.module;

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

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

Adobe Flash Player and Custom Methods

Parameters passed from the Adobe Flash player client to Wowza Server need to be marshaled to Java primitive and object types. The com.wowza.wms.module.ModuleBase class includes a number of 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. Consult the server API javadocs and the **Server Side Coding** example for more detailed information. Below is a simple example of converting three incoming parameters:

```
package com.mycompany.module;

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

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

A custom method called from the Adobe Flash player may return a single result value. This value must be converted to an Action Message Format (AMF) object to be understood by the Flash player. These value types can include simple types like strings, integers and booleans as well as more complex types like objects, arrays or arrays of objects. The com.wowza.wms.module.ModuleBase class includes a number of helper functions for returning simple types. For more complex types the com.wowza.wms.amf package contains an API for object creation and conversion. Consult the server API javadocs and the **Server Side Coding** example for more detailed information. Below is a simple example of three methods returning simple value types:

```
package com.mycompany.module;

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

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

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

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

Adobe Flash Player and Server to Client Calls

A custom method can call a function in Adobe Flash player directly by invoking the IClient.call() method. The client call can return a single variable that will be received by the server by creating a result object that implements the com.mycompany.module.IModuleCallResult interface. The IClient.call() method has two forms:

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

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

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

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

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

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

```
package com.mycompany.module;

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

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

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

Logging

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

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

Java Management Extensions (JMX)

All modules instantiated for a given application instance will be made available through the Java Management Extension's (JMX) Interface. The path to the modules section in the MBean interface is:

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

All public methods and properties (wrapped in Java Bean get/set methods) will be made available through the **Instance** object found within 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. So what this means is that your custom modules are instantly made available through the Wowza Server administration interface without an 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.

HTTPProvider Classes

HTTPProviders are Java classes that are mini Java servlets that can be used to add an HTTP interface to Wowza Server. They are configured on a per-port basis in **[install-dir]/conf/VHost.xml** (configuration is covered in the previous chapter). Below is a simple HTTPProvider that returns the server version:

```
package com.mycompany.wms.http;

import java.io.*;

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

public class HTTPServerVersion extends HTTProvider2Base
{
	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 HTTPProviders is encapsulated in the HTTProvider2Base base class. Your HTTPProvider if it extends this class only needs to implement the onHTTPRequest method. Below are a few interesting code snippets to aid in HTTPProvider development:

Get HTTP request URL
```
String path = super.getPath(req, false);
```

Get HTTP request header value
```
String headerValue = req.getHeader(headerName);
```

Set HTTP response header value
```
resp.setHeader(headerName, headerValue);
```

Set HTTP response status
```
resp.setResponseCode(404);
```

There are several more complex and interesting examples of HTTPProviders in the online **Wowza Media Server Forums**.

Event Listeners

There are many points within the Wowza Media Server 2 object hierarchy where event listeners can be added. Event listeners are classes that implement a notifier interface and are notified of specifc 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 is notified each time an application instance is started or stopped. Below are specifics on the more interesting and useful listener interfaces:

Server Listener (IServerNotify2)

Sever listeners are notified of the life cycle of the server and are a great place to invoke and attach functionality that you would like to make avaible while Wowza Server is running. Examples are web services or SOAP interface and a web server or HTTP interface. Below is a simple server listener:

```
package com.mycompany.wms;

import com.wowza.wms.server.*;

public class MyServerListener implements IServerNotify2
{
	public void onServerCreate(IServer server)
	{
		System.out.println("onServerCreate");
	}

	public void onServerConfigLoaded(IServer server)
	{
		System.out.println("onServerConfigLoaded");
	}

	public void onServerInit(IServer server)
	{
		System.out.println("onServerInit");
	}

	public void onServerShutdownStart(IServer server)
	{
		System.out.println("onServerShutdownStart");
	}

	public void onServerShutdownComplete(IServer server)
	{
		System.out.println("onServerShutdownComplete");
	}
}
```

Once compiled, bound into a **.jar** file and placed in the **[install-dir]/lib** folder this server listener can be invoked by adding an entry to the **<ServerListeners>** list in **[install-dir]/conf/Server.xml**:

```
<ServerListener>
	<BaseClass>com.mycompany.wms.MyServerListener</BaseClass>
</ServerListener>
```

Virtual Host Listener (IServerNotify2)

Virtual host listeners are notified of the life cycle of virtual host. Below is a simple virtual listener:

```
package com.mycompany.wms;

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

public class MyVHostListener implements IVHostNotify
{
	public void onVHostCreate(IVHost vhost)
	{
		System.out.println("onVHostCreate: "+vhost.getName());
	}

	public void onVHostInit(IVHost vhost)
	{
		System.out.println("onVHostInit: "+vhost.getName());
	}

	public void onVHostShutdownStart(IVHost vhost)
	{
		System.out.println("onVHostShutdownStart: "+vhost.getName());
	}

	public void onVHostShutdownComplete(IVHost vhost)
	{
		System.out.println("onVHostShutdownComplete: "+vhost.getName());
	}

	public void onVHostClientConnect(IVHost vhost, IClient inClient,
			RequestFunction function, AMFDataList params)
	{
		System.out.println("onVHostClientConnect: "+vhost.getName());
	}

}
```

Once compiled, bound into a **.jar** file and placed in the **[install-dir]/lib** folder this virtual host listener can be invoked by adding an entry to the **<VHostListeners>** list in **[install-dir]/conf/Server.xml**:

```
<VHostListener>
	<BaseClass>com.mycompany.wms.MyVHostListener</BaseClass>
</VHostListener>
```

MediaStream Listeners (IMediaStreamActionNotify2)

MediaStream listeners receive play, publish, pause… events for an Adobe Flash MediaStream object. Below is a simple MediaStream listener:

```
package com.mycompany.wms;

import com.wowza.wms.amf.*;
import com.wowza.wms.stream.*;

public class MyMediaStreamListener implements IMediaStreamActionNotify2
{
	public void onMetaData(IMediaStream stream, AMFPacket metaDataPacket)
	{
		System.out.println("onMetaData");
	}

	public void onPauseRaw(IMediaStream stream, boolean isPause,
		double location)
	{
		System.out.println("onPauseRaw");
	}

	public void onPause(IMediaStream stream, boolean isPause,
		double location)
	{
		System.out.println("onPause");
	}

	public void onPlay(IMediaStream stream, String streamName,
		double playStart, double playLen, int playReset)
	{
		System.out.println("onPlay");
	}

	public void onPublish(IMediaStream stream, String streamName,
		boolean isRecord, boolean isAppend)
	{
		System.out.println("onPublish");
	}

	public void onSeek(IMediaStream stream, double location)
	{
		System.out.println("onSeek");
	}

	public void onStop(IMediaStream stream)
	{
		System.out.println("onStop");
	}

	public void onUnPublish(IMediaStream stream, String streamName,
		boolean isRecord, boolean isAppend)
	{
		System.out.println("onUnPublish");
	}
}
```

Once compiled, bound into a **.jar** file and placed in the **[install-dir]/lib** folder this MediaStream listener can be invoked by creating an instance of this object and attaching it to an IMediaStream object. You might do this in an **onStreamCreate** event method like this:

```
public void onStreamCreate(IMediaStream stream)
{
	stream.addClientListener(new MyMediaStreamListener());
}
```

Chapter 9

Server Administration

How do I setup, manage, and deploy Wowza Media Server 2?

Wowza Media Server 2 is a small and powerful Java server. It is 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 Wowza Server custom applications since the server can be started and stopped quickly and server log messages can be seen immediately in the console window. Running the server as a system service is most often used for server deployment where the server needs to continue to run even after you log off the machine or be automatically started when the server is rebooted.

Configuring SSL and RTMPS

Wowza Media Server 2 supports Secure Socket Layer (SSL) and RTMPS (RTMP over SSL) streaming protection. SSL is a technology which allows web browsers and web servers to communicate over a secured connection. This means that the data being sent and received is encrypted in both directions. You can get an SSL certificate from a certificate authority or you can create a certficiate your self. The instructions below will walk you through the steps to create a self signed SSL certificate for use with Wowza Server. If you would like to obtain an SSL certificate from a certificate authority follow the steps in this forum post:

http://www.wowzamedia.com/forums/showthread.php?t=6464

Below are the steps to create a self-signed SSL certificate using the keytool application that comes with the Java JDK. To get started, install the Java JDK and be sure the **bin** folder of your JDK installation has been added to your PATH variable. If the PATH variable is configured correctly, you should be able to open a command prompt and execute the command **keytool**. This should return the command reference for the keytool command. Once you have the keytool command up and running, proceed to the following steps to create a self-signed SSL certificate:

1. Open a command prompt and change directory to **[install-dir]/conf**
2. Execute the following command:

   keytool -genkey -alias wowza -keyalg RSA -keystore ssl.mycompany.com.cert

3. You will then be prompted to answer the following questions. Below are sample responses assuming the domain name that you wish this certificate to be tied to is **ssl.mycompany.com**:

```
[Enter keystore password]
 password
[What is your first and last name]
 ssl.mycompany.com
[What is the name of your organizational unit]
 Web Department
[What is the name of your organization]
 My Company Name
[What is the name of your City or Locality]
 Cincinnati
[What is the name of your State or Province]
 Ohio
[What is the two-letter country code for this unit]
 US
[Enter key password for <password>]
 password
```

Once complete, you will see a file named **ssl.mycompany.com.cert** in the **[install-dir]/conf** folder. This is the certificate file. To configure TCP port to use this certificate, edit **[install-dir]/conf/VHost.xml** and make the following changes:

1. Uncomment the **<HostPort>** definition for port **443** that is just following the comment **<!-- 443 with SSL -->** (be sure to remove the comment before **<HostPort>** and after **</HostPort>**).

2. Set the value **SSLConfig/KeyStorePath** to:

${com.wowza.wms.context.VHostConfigHome}/conf/ssl.mycompany.com.cert

3. Set the **SSLConfig/KeyStorePassword** to key store password entered above.

TCP port 443 is now protected by SSL and RTMPS. You will need to setup a domain name entry for the domain chosen above and all communications using port 443 will need to done using SSL or RTMPS and the domain name specified in the certificate.

Self-signed certificates (this not does not apply to certificates from certificate authorities) do not work on the Mac (OSX) when using Adobe Flash to stream over RTMPS without first installing the certificate in the Keychain and setting its trust level to Always Trust. To extract the certificate and install in the OSX Keychain, do the following:

1. Extract the certificate from the keystore using the following command and copy the ssl.mycompany.com.crt file to the Mac:

```
keytool -export -alias wowza -file ssl.mycompany.com.crt -keystore
ssl.mycompany.com.cert
```

2. Open the **Applications > Utilities > Keychain Access** utility and select the **Certificates** category

3. Drag and drop the **ssl.mycompany.com.crt** onto this utility

4. Right click on the **ssl.mycompany.com** entry in the list and select **Get Info**

5. Open the **Trust** section of the info dialog and set **When using this certificate** to **Always Trust**

These steps need to be followed on any machine that is going to use RTMPS to play a stream that is protected using a self-signed certificate. Obviously this is not the preferred way to stream using RTMPS. It is better to obtain a signed certificate from a trusted certificate authority. With a trusted certificate these additional steps are not necessary.

There are two methods of doing RTMPS streaming when using the Adobe Flash player. The default method leverages tunneling (RTMPT over SSL) which can be slow and causes additional server load. The second method is RTMP over SSL which performs better. You can enable this mode by setting the **NetConnection.proxyType** to **"best"** before calling **NetConnection.connect**. The code looks like this:

```
var nc:NetConnection = new NetConnection();
nc.proxyType = "best";
nc.connect("rtmps://ssl.mycompany.com/myapplication");
```

Logging

Wowza Media Server 2 uses the apache.org log4j library 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 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. The log files are written to the following folder:

```
[install-dir]/logs
```

Wowza Media Server logging can generate the following logging fields:

date	Date of log event
time	Time of log event
tz	Time zone of log event
x-event	Log event (see table below)
x-category	Log event category (server, vhost, application, session, stream)
x-severity	Log event severity (DEBUG, INFO, WARN, ERROR, FATAL)
x-status	Status of log event (see table below)
x-ctx	Extra data about the context of the log event
x-comment	Extra comment about the log event
x-vhost	Name of the virtual host from which the event was generated
x-app	Name of the application from which the event was generated
x-appinst	Name of the application instance from which the event was generated
x-duration	Time in seconds that this event occurred within the lifetime of the x-category object
s-ip	IP address on which the server received this event
s-port	Port number on which the server received this event
s-uri	Full connection string on which the server received this event
c-ip	Client connection IP address
c-proto	Client connection protocol (rtmp, rtmpe, rtmpt(HTTP-1.1), rtmpte(HTTP-1.1), rtmps(HTTP-1.1), http (cupertino), http (smooth))
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
c-client-id	Client ID number assigned by the server to the connection
cs-bytes	Total number of bytes transferred from client to server (accumulative)
sc-bytes	Total number of bytes transferred from server to client (accumulative)
x-stream-id	Stream ID number assigned by server to the stream object
x-spos	Position in milliseconds within the media stream
cs-stream-bytes	Total number of bytes transferred from client to server for stream x-stream-id (accumulative)
sc-stream-bytes	Total number of bytes transferred from server to client for stream x-stream-id (accumulative)
x-sname	Name of stream x-stream-id
x-sname-query	Query parameters of stream x-stream-id
x-file-name	Full file path of stream x-stream-id
x-file-ext	File extension of stream x-stream-id
x-file-size	File size in bytes of stream x-stream-id
x-file-length	File length in seconds of stream x-stream-id
x-suri	Full connection string for stream x-stream-id (including query parameters)
x-suri-stem	Full connection string for stream x-stream-id (excluding query parameters)
x-suri-query	Query parameter for connection string
cs-uri-stem	Full connection string for stream x-stream-id (excluding query parameters)
cs-uri-query	Query parameter for stream x-stream-id

Wowza Media Server generates the following logging events:

comment	Comment
server-start	Server start
server-stop	Server shutdown
vhost-start	Virtual host start
vhost-stop	Virtual host shutdown
app-start	Application instance start
app-stop	Application instance shutdown
connect-pending	Connection pending approval by application and license manager
connect	Connection result
connect-burst	Connection accepted in burst zone
disconnect	Client (session) disconnected from server
play	Play has started
pause	Play has paused
unpause	Play has unpaused /resumed
seek	Seek has occurred
setstreamtype	Client call to netConnection.call("setStreamType", null, "[streamtype]");
setbuffertime	Client call to NetStream.setBufferTime(secs) logged in milliseconds
stop	Play has stopped on a stream
create	Media or data stream created
destroy	Media or data stream destroyed
publish	Start stream publishing
unpublish	Stop stream publishing
record	Start stream recording
recordstop	Stop stream recording
announce	RTSP Session Description Protocol (SDP) ANNOUNCE

Wowza Media Server generates the following logging status values:

100	Pending or waiting (for approval)
200	Success
302	Rejected by application with redirect information
400	Bad request
401	Rejected by application
413	Rejected by license manager
500	Internal error

Wowza Server logging is configured in the conf/log4j.properties properties file. There are many logging configuration options made available by the log4j logging system. The remainder of this section will cover the basic options for enabling and disabling different logging fields, events and categories. Below is an example of a basic log4j.properties file for Wowza Server.

```
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.File=${com.wowza.wms.ConfigHome}/logs/wowzamediaserver_access.log
log4j.appender.serverAccess.layout=com.wowza.wms.logging.ECLFPatternLayout
log4j.appender.serverAccess.layout.Fields=x-severity,x-category,x-event;date,time,c-client-id,c-ip,c-
port,cs-bytes,sc-bytes,x-duration,x-sname,x-stream-id,sc-stream-bytes,cs-stream-bytes,x-file-size,x-
file-length,x-ctx,x-comment
log4j.appender.serverAccess.layout.OutputHeader=true
log4j.appender.serverAccess.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 this file sets the logging level to INFO and defines three appenders; stdout, serverAccess, serverError. Setting the logging level to INFO configures the logging mechanism such that it will only log events with a severity of INFO or greater. The logging severity in ascending order is: DEBUG, INFO, WARN, ERROR and FATAL. To log all events set the logging level to DEBUG.

Next, we configure each of the appenders. The important properties in this section are:

Field	Comma delimited list of fields to log
OutputHeader	Boolean value (true/false) that instructs the logging system to write out a W3C Extended Common Log Format header each time the server is started.
QuoteFields	Boolean value (true/false) that instructs the logging system to surround all field data in double quotes
Delimiter	The delimiter character to use between field values. Valid values are **tab**, **space** or the actual delimiter character.
CategoryInclude	Comma separated list of logging categories. Only log events with the specified categories will be logged.
CategoryExclude	Comma separated list of logging categories. Only log events whose category is not in this list will be logged.
EventInclude	Comma separated list of logging events. Only log events with the specified event name will be logged.
EventExclude	Comma separated list of logging categories. Only log events whose event name is not in this list will be logged.

These properties allow you to control the way the log information is formatted and filtered. For more detailed information on how to configure the log4j specific properties such as log file rolling and additional log appender types visit the apache.org website at http://logging.apache.org/log4j.

Wowza Media Server 2 can also be configured to generate logs on a per-virtual host and per-application basis. These configurations are included but commented out at the bottom of the default [install-dir]/conf/log4j.properties files. The first commented out section includes configuration for per-application logging. The second commented out section includes configuration for per-virtual host logging. To turn either of these features on, simply remove the comments (**#** sign at the beginning of each of the lines) from the section. The per-virtual host logging will generate log files using the following directory structure:

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

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

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

This method of log file generation can be very useful if you plan on offering the Wowza Media Server 2 as a shared service to several customers.

Logging to a Database

Wowza Media Server 2 can be configured to log information to a database. This is a very useful feature if you wish to get real-time statistics. See this online forum post for detailed instructions:

http://www.wowzamedia.com/forums/showthread.php?t=6037

Run Server as Named User

The default installation of Wowza Media Server 2 on Linux and Mac OS X will install and run the server as the **root** user. If you would like to run the server as a user other than **root**, you can follow these instructions to create a new user and configure the server to run as that new user.

Note

For security reasons, most Linux and Unix distributions do not allow user's other than the **root** user to bind to port numbers less than 1024. If you plan on running Wowza Server on a lowered numbered ports such as 80 (HTTP), 443 (RTMPS, HTTPS) and/or 554 (RTSP) then the server will need to continue to run as the **root** user.

Linux

First, we are going to create a new user and group named **wowza**.

```
groupadd wowza
useradd -g wowza wowza
passwd wowza
```

Next, we are going to change ownership and permissions on Wowza Server installation files.

```
cd /usr/local
chown wowza:wowza WowzaMediaServer
chown –R wowza:wowza WowzaMediaServer-2.0.0
chmod –R 775 WowzaMediaServer-2.0.0
rm –f /var/run/WowzaMediaServer.pid
rm –f /var/run/WowzaMediaServer.lock
```

Finally, we are going to change the command that is used to start the server so that it is run as the new **wowza** user. Change directory to the **/usr/local/WowzaMediaServer/bin** directory. Edit the standalone startup script **startup.sh** and prepend **sudo –u wowza** to the 24th line. It should now be:

```
sudo –u wowza $_EXECJAVA $JAVA_OPTS -Dcom.wowza.wms.AppHome=
      "$WMSAPP_HOME" -Dcom.wowza.wms.ConfigHome=
      "$WMSCONFIG_HOME" -cp
      $WMSAPP_HOME/bin/wms-bootstrap.jar
      com.wowza.wms.bootstrap.Bootstrap start
```

You will also need to edit the service startup script **wms.sh** and make the same change to line 24. Now both the standalone startup script and the service startup script will start the server as the user **wowza**.

Mac OS X

First, we are going to create a new user named **wowza**. Open the **Accounts** systems preferences panel. Unlock the add user functionality by clicking on the lock icon in the lower left hand corner of the panel (you will be asked to enter your administrative password). Click the **+** button below the list of users to add a new user. Enter the following values and click the **Create Account** button:

```
Name:              wowza
Short Name:        wowza
Passord:           [enter a password]
Verify:            [enter a password]
```

Next, we are going to change the permissions on Wowza Server installation files. Open a **Terminal** window and enter the following commands:

```
cd /Library
sudo chown wowza:admin WowzaMediaServer
sudo chown –R wowza:admin WowzaMediaServer-2.0.0
```

Finally, we are going to change the command that is used to start the server so that it is run as the new **wowza** user. Change directory to the **/Library/WowzaMediaServer/bin** directory. Edit the standalone startup script **startup.sh** and prepend **sudo –u wowza** to the 24th line. It should now be:

```
sudo –u wowza $_EXECJAVA $JAVA_OPTS -Dcom.wowza.wms.AppHome=
      "$WMSAPP_HOME" -Dcom.wowza.wms.ConfigHome=
      "$WMSCONFIG_HOME" -cp
      $WMSAPP_HOME/bin/wms-bootstrap.jar
      com.wowza.wms.bootstrap.Bootstrap start
```

Now when you start the server in standalone and service mode it will run as user **wowza**. You can verify this by executing the **ps –ja** command in a **Terminal** window while the server is running.

Note

For more up to date security information visit the **Useful Code** section of the **Wowza Media Systems Forums** at http://www.wowzamedia.com/forums/.

Chapter 10

Server Management Console and Monitoring

How do I manage and monitor Wowza Media Server 2?

Wowza Media Server 2 can be managed and monitored through a Java Management Extensions (JMX) interface. JMX is a standards-based technology for exposing components of a Java application through a unified object interface. This interface can then be consumed by open source and commercial monitoring tools such as HP OpenView, OpenNMS (http://www.opennms.org), JConsole and VisualVM (http://visualvm.dev.java.net).

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. Please consult your vendor's documentation.

Note

A good place to learn more about the Java Management Extension (JMX) standard is from the Sun website (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/).

Local Management Using JConsole

Wowza 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 by using the JConsole applet that ships with the Java Development Kit (JDK) of most popular VMs. This tool is usually located in the bin folder of your Java JDK installation. By default the 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, first start Wowza Media Server (either by running it as a service or standalone from a command prompt). Next, run JConsole. In JConsole you should see a list of the currently running Java virtual machines that are

exposing a JMX interface. Wowza Server will be listed as **com.wowza.wms.bootstrap.Bootstrap start**. Select this item and click the **Connect** button.

Note

On 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 is not supported on a FAT file system that provide insufficient access controls. The workaround is to setup remote monitoring. See the **Remote Management** section below, to learn how to configure the remote JMX interface.

From here you can explore the different tab panels that are part of JConsole. Wowza Media Server 2 management objects are located under the **MBean** tab in the **WowzaMediaServer** group. The JMX objects are organized based on the configured virtual hosts, applications and applications instances. Monitoring objects will be created and deleted on the fly as applications, application instances, client connections and streams are created and deleted from the server.

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 Server from a remote computer. Both the JVM and Wowza Server include remote JMX interfaces. It is only necessary to configure one of these remote interfaces to enable remote monitoring. It is suggested that you use the Wowza Server remote interface since it is more easily configured and can be properly exposed through hardware or software based firewalls. The following two sections describe the configuration process.

Wowza Media Server built-in JMX interface configuration

The remote JMX interface built into Wowza Media Server 2 can be configured through the **JMXRemoteConfiguration** and **AdminInterface** sections of the **[install-dir]/conf/Server.xml** file. This section contains the following settings:

JMXRemoteConfiguration - Enable, IpAddress, RMIServerHostName, RMIConnectionPort, RMIRegistryPort

The **Enable** setting is a boolean value that can either be **true** or **false** and is the main switch to turn on and off the remote JMX interface. The default value is **false**. Setting this value to **true** (with no further modifications to the other settings), will turn on the remote JMX interface with authentication. The default username/password is admin/admin and the URL for invocation in JConsole or VisualVM is:

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

The **IpAddress** and **RMIServerHostName** work together to properly expose the JMX interface to the network. In general **IpAddress** should be set to the internal ip address of the server running Wowza Media Server and **RMIServerHostName** should be set to the external ip address or domain name of the machine. For example, if the server running Wowza 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 **RMIConnectionPort** and **RMIRegistryPort** settings control the TCP ports used to expose the RMI connection and RMI registry interfaces. These values only need to be changed if Wowza Server reports port conflicts upon startup. The default values for these settings are 8084 and 8085 respectively. The **RMIConnectionPort** corresponds to the first port number in the connection URL and the **RMIRegistryPort** to the second.

The **IpAddress**, **RMIConnectionPort** and **RMIRegistryPort** effect the connection URL in the following way:

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

If the remote JMX interface is enabled, Wowza Server upon startup will log the URL of the currently configured JMX interface. 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 **RMIConnectionPort** and **RMIRegistryPort** settings.

JMXRemoteConfiguration - Authenticate, PasswordFile, AccessFile

The **Authenticate** setting is a boolean value that can either be **true** or **false** and is the main switch to turn on and off remote JMX interface authentication. The **PasswordFile** and **AccessFile** settings are the full path to the JMX password and access files.

The password file is a text file with one line per user. Each line contains a username followed by a space followed by a password. The access file contains one line per user. Each line contains a username followed one of two access permission identifiers; **readwrite** or **readonly**. A sample password file **jmxremote.password** and sample access file **jmxremote.access** can be found in the conf directory of the installation. These files define three named users:

```
admin (password admin)        – access readwrite
monitorRole (password admin)  - access readonly
controlRol (password admin)   - access readwrite
```

Note

Some Java Runtime Environments require that both the password and access files have read only privileges. On Linux, this can be achieved by setting the permissions on the 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 either be **true** or **false** and is the switch to turn on and off remote JMX interface over SSL. SSL configuration can get quite involved. The following online documentation describes the process for enabling SSL with JMX: http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html#gdemv.

AdminInterface/ObjectList

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

```
Server                - Server level connection and performance info and notifications
VHost                 - Information about currently running virtual hosts
VHostItem             - Details of currently configured virtual hosts
Application           - Application level connection and performance info
ApplicationInstance   - Application Instance level connection and connection info
Module                - Details of currently loaded modules
MediaCaster           - Details of media caster objects (ie, live stream repeater)
Client                - Details of each connected Flash session
MediaStream           - Details of each individual server side NetStream object
SharedObject          - Details of currently loaded shared objects
Acceptor              - Details of currently running host ports or TCP ports
IdleWorker            - Details of currently running idle workers
```

Exposing **Client**, **MediaStream** and/or **SharedObject** information can add significant load to the server and to the JMX interface. You will most likely want to turn off this level of detail for deployed solutions.

JVM built-in JMX interface configuration

The remote JMX interface built into the Java Virtual Machine can be configured through the Wowza Media Server start scripts. The following scripts in the **bin** folder can be edited to enable remote JMX monitoring

```
startup.bat                      – Windows standalone startup script
WowzaMediaServer-Service.conf    - Windows service config script
startup.sh                       - Linux/Mac OS X standalone startup
                                   script
wms.sh                           - Linux/Mac OS X service startup script
```

Each of these scripts contain commented out configuration parameters that can be used to configure the remote interface. A detailed description of the process for configuring the remote interface can be found at http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html.

Below are the settings that are used to configure remote connections.

```
-Djava.rmi.server.hostname=192.168.1.7
-Dcom.sun.management.jmxremote.port=1099
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.password.file=jmxremote.password
-Dcom.sun.management.jmxremote.access.file=jmxremote.access
```

-Dcom.sun.management.jmxremote.port=[port-number]
The remote port that the JMX service will listen on for remote connections. Be sure to open up this port on any firewalls between the server and the remote client.

-Dcom.sun.management.jmxremote.ssl=[true,false]
Boolean value that turns on and off remote SSL connections. Default is true. If set to true you must properly install and configure server side digital certificates. A detailed description of the procedure for installing and configuring digital certificates can be found at: http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html#SSL_enabled.

-Dcom.sun.management.jmxremote.authenticate=[true,false]
-Dcom.sun.management.jmxremote.password.file=[path-to-password-file]
-Dcom.sun.management.jmxremote.access.file=[path-to-access-file]
These three settings control remote JMX authentication. To turn off authentication set com.sun.management.jmxremote.authenticate to false. To enable authentication set com.sun.management.jmxremote.authenticate to true and configure the password and access files as defined below.

The password file is a text file with one line per user. Each line contains a username followed by a space followed by a password. The access file contains one line per user. Each line contains a username followed one of two access permission identifiers; **readwrite** or **readonly**. A sample password file **jmxremote.password** and sample access file **jmxremote.access** can be found in the conf directory of the installation. These files define three named users:

```
admin (password admin)        – access readwrite
monitorRole (password admin)  - access readonly
controlRol (password admin)   - access readwrite
```

Before configuring your server for authentication, you will want to change the default usernames and passwords.

Many virtual machines require that these files have read-only file permissions. On Windows the file must be located outside the C:\Program File folder and the file permissions can be set using the cacls command. To setup authentication on Windows, do the following:

1. Create a folder at the root of your C: drive named **WowzaMediaServerJMX**.
2. Copy the **[install-dir]/conf/jmxremote.access** and **[install-dir]/conf/jmxremote.password** into this new folder.
3. Open a DOS command shell, change directory to **C:\WowzaMediaServerJMX**, and run the following **cacls** command on the two files:

```
cacls jmxremote.password /P [username]:R
cacls jmxremote.access /P [username]:R
```

Where **[username]** is the user running the java process or service.

4. Update the jmxremote settings to reflect the new location:

```
-Dcom.sun.management.jmxremote.password.file=C:\WowzaMediaServerJMX\jmxremote.password
-Dcom.sun.management.jmxremote.access.file=C:\WowzaMediaServerJMX\jmxremote.access
```

On Linux and Mac OS X there is no need to move the files from their default location. Simply change the file permissions using chmod. Below is an example:

```
chmod 600 jmxremote.password
chmod 600 jmxremote.access
```

-Djava.rmi.server.hostname=[hostname/ip-address]
Server host name or ip address. This setting is often required if the server either has multiple ip addresses or if the hostname for the server resolves to different ip address based on how the server is being accessed (inside and outside a firewall or router space).

Note

When running Wowza Media Server 2 as a Windows service, the JMX interface will not be available unless the service is running as a named user. To configure the service to run as a named user, go to **Settings>Control Panel>Administrative Tools>Services** and right click on the **Wowza Media Server** service and select **Properties**. Next, click on the **Log On** tab, change the **Log on as** radio to **This account** and enter a user name and password for a local user.

Remote Management

Remote Management Using JConsole

JConsole can also be used to monitor a remote Wowza Server. Once you configured the remote JMX interface as described above, run JConsole. Enter the remote JMX interface URL into the **Remote Process** field. The default remote JMX interface URL for the Wowza Media Server 2 built-in JMX interface is:

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

The default remote JMX interface URL for the JVM built-in JMX interface is:

```
service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
```

Finally, enter your user name and password into the provided fields and click the **Connect** button. You should now be connected to the remote server and able to view the JMX hierarchy.

Remote Management Using VisualVM

Another great tool for monitoring Wowza Media Server 2 over JMX is VisualVM. VisualVM can be downloaded from the following location:

http://visualvm.dev.java.net

Once you get it installed and running, it is best to install the MBean plugin. To do this select the **Plugins** command from the **Tools** menu. In the **Available Plugins** tab put a check mark next to the **VisualVM-MBean** plugin and click the **Install** button. Once you get this plugin installed it will provide similar information to JConsole. You can select **Add JMX Connection** from the **File** menu to add your Wowza Media Server 2 to the **Applications** list.

Object Overview

This section describes the more important top level objects that can be used to monitor the server's performance and uptime. This section will not cover each and every object that is exposed by the server. These objects are available under the **WowzaMediaServer** object in the MBean 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 on 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 on 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 will display 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 to that section of the server. For example the Connections object under a particular Application object will display the current clients 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 pertaining to the connection limits and connection bursting capabilities of Wowza Media Server. Wowza Media Server 2 can generate the following notification events:

```
com.wowza.wms.connect.WarningServerLicenseLimit      - connection accepted in
                                                       bursting zone (warning)
com.wowza.wms.connect.ErrorServerLicenseLimit        - connection refused due
                                                       due to license limit
com.wowza.wms.connect.WarningVHostLimit              - connection refused due
                                                       to virtual host limit
```

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. Notification events can be viewed in JConsole by navigating to the **MBean** tab, opening the **WowzaMediaServer** group and selecting the **ServerNotification** object. Next, select the **Notifications** tab and click the **Subscribe** button. All events will display as new rows in the **Notifications** list. Only events that occur after you subscribe to the notifications will be displayed.

Chapter 11

Virtual Hosting

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

Wowza Media Server 2 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

Below is a description of the VHosts.xml file in the conf directory that is used to define a virtual host.

VHosts.xml

The VHosts.xml configuration file is used to define each of the virtual host environments. Below is a description of each of the items that 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. The contents of this directory will be described below.

VHosts/VHost/ConnectionLimit

The maximum number of simultaneous connections this virtual host can support. If this value is zero the virtual host can have an unlimited number of simultaneous connections.

Typical Configuration

Let's jump in and look at a typical VHosts.xml file for a virtual host environment that contains two virtual hosts: **vhost1** and **vhost2**.

```
<Root>
	<VHosts>
		<VHost>
			<Name>vhost1</Name>
			<ConfigDir>/home/vhosts/vhost1</ConfigDir>
			<ConnectionLimit>0</ConnectionLimit>
		</VHost>
		<VHost>
			<Name>vhost2</Name>
			<ConfigDir>/home/vhosts/vhost2</ConfigDir>
			<ConnectionLimit>0</ConnectionLimit>
		</VHost>
	</VHosts>
</Root>
```

The directory structure for these two virtual hosts would be the following:

```
[/home/vhosts]
            [vhost1]
                  [applications]
                  [conf]
                        Application.xml
                        Authentication.xml
                        HTTPStreamers.xml
                        LiveStreamPacketizers.xml
                        MediaCasters.xml
                        MediaReaders.xml
                        MediaWriters.xml
                        MP3Tags.xml
                        RTP.xml
                        StartupStreams.xml
                        Streams.xml
                        VHost.xml
                        admin.password
                        publish.password
                  [content]
                  [keys]
                  [logs]
            [vhost2]
                  [applications]
                  [conf]
                        Application.xml
                        Authentication.xml
                        HTTPStreamers.xml
                        LiveStreamPacketizers.xml
                        MediaCasters.xml
                        MediaReaders.xml
                        MediaWriters.xml
                        MP3Tags.xml
                        RTP.xml
                        StartupStreams.xml
                        Streams.xml
                        VHost.xml
                        admin.password
                        publish.password
                  [content]
                  [keys]
                  [logs]
```

Note

See the logging section for instructions on how to configure per virtual host logging.

The process for virtual host configuration is very 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 an application, conf and logs directory. Each virtual host gets its own set of configuration files.

It is very important to note that Wowza Server only supports ip-address/port based virtual hosting and does not support domain named based virtual hosting. What this means is that in VHost.xml each virtual host must define HostPort entries with unique ip-address and port combinations that do not conflict with other virtual hosts defined on a given server. The following combinations represent valid vhost 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>
```

Through the JMX interface and the VHosts.xml configuration file virtual hosts can be added, modified and deleted on the fly without stopping and restarting the server. The virtual host operations can be accessed through JConsole. First, with the server running start JConsole and select the **MBean** tab. Open the **WowzaMediaServer** group and select the **Server** object. The virtual host operations are found under the **Operations** tab. There are three operations of interest:

```
startVHost              - start an individual vhost by name
stopVHost               - stop an individual vhost by name
reloadVHostConfig       - reload the VHosts.xml configuration file
```

To add a new virtual host without restarting the server, edit **VHosts.xml** add a new virtual host definition and copy and configure a new set of configuration files as described above. Next, open JConsole and navigate to the **Server** object and click the **reloadVHostConfig** to reload the **VHosts.xml** file. Finally, enter the name of the new virtual host into the text entry box next to the **startVHost** button and click the button. The new virtual host will be started immediately.

Chapter 12

Examples & AddOn Packages

What do each of these examples do and where do I get AddOn Packages?

Wowza Media Server 2 ships with many examples that highlight the functionality of the server. This chapter describes each of these examples and provides. All Adobe Flash examples are implemented using ActionScript 3.0. For most Flash examples, there is also an ActionScript 2.0 implementation provided in the **clientAS2** folder and an Adobe Flex version in the examples **clientFlex** folder. Older Flash players may only support ActionScript 2.0.

Wowza Media Systems also provide several AddOn Packages that extend and enhance the functionality of Wowza Media Server. An up to date list of AddOn Packages can found here:

http://www.wowzamedia.com/packages.html

Note

In the root folder of each example is a README.txt that contains any extra installation steps that are necessary to make the example function.

SimpleVideoStreaming

This example includes a video on demand player for Adobe Flash and Microsoft Silverlight. It includes source code for Adobe Flash CS3 or greater, Adobe Flex 3 or greater and Microsoft Silverlight 3 or greater. It utilizes the **file** stream type.

FastPlayVideoStreaming

This is an Adobe Flash example that illustrates how to use the ModuleFastPlay module. Fast play is a simple fast forward, fast rewind and slow motion streaming playback mechanism.

LiveVideoStreaming

This is an Adobe Flash example that illustrates how to setup and playback live video. It utilizes the **live** stream type.

NativeRTPVideoStreaming

This is an Adobe Flash example that illustrates how to setup and playback live video from a native RTP source. It utilizes the **rtp-live** stream type.

VideoChat

This is an Adobe Flash example that illustrates how to implement video chat between two users. It utilizes the **live-lowlatency** stream type and uses the Camera and Microphone objects to obtain video and audio content. The example can either stream video and audio data between two client connections or loop the data back to itself.

VideoRecording

This is an Adobe Flash example that illustrates how to implement client to server video recording. It utilizes the **record** stream type and uses the Camera and Microphone objects to obtain video and audio content.

TextChat

This is an Adobe Flash example that illustrates how to implement a simple text chat application.

SHOUTcast

This is an Adobe Flash xample that illustrates how re-stream SHOUTcast MP3 or AAC+ audio data through Wowza Media Server 2. It utilizes the **shoutcast** stream type.

RemoteSharedObjects

This is an Adobe Flash example 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. To see the data synchronization in action, try opening two instances of the example. While you make changes in one instance you will see the data update in the other.

ServerSideModules

This example is referenced by the Wowza IDE: User's Guide and is a good starting point to learn how to create your first custom server side module.

MediaSecurity

Wowza Media Systems provides a media security package that includes SecureToken and RTMP Authentication functionality as well as a document that covers other methods of securing Wowza Media Server 2. To obtain the latest version of this package visit the following Wowza Media Server forum thread:

http://www.wowzamedia.com/forums/showthread.php?t=1281

BWChecker

This is an Adobe Flash example that provides a means for testing the bandwidth between individual Flash client connections and Wowza Media Server 2. It includes both a debugging tool that can be used to interactively test bandwidth as well as Flash code that you can embed into your Flash application.

LoadBalancer

Wowza Media Systems provides a dynamic load balancing package that you can add to the Wowza Media Server 2. To obtain the latest version of this package visit the following Wowza Media Server forum thread:

http://www.wowzamedia.com/forums/showthread.php?t=4637