Wowza Media Server® Pro

User's Guide

Wowza Media Server Pro: User's Guide

®

Version 1.7

Table of Contents

Chapter

1

Introduction

What is the Wowza Media Server Pro?

Wowza Media Server Pro is an interactive RTMP server for streaming video, audio and data content to and from the Adobe® Flash® Player client, executing remote procedure calls and supporting remote shared objects. It is an alternative to the Adobe Flash Media Server product (FMIS and FMSS). In addition, Wowza Media Server Pro supports the Real-Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP) and MPEG Transport Streams (MPEG-TS) for incoming streaming of H.264/HE-AAC content. Wowza Media Server Pro is a powerful and extensible Java based server that can be deployed on any platform that supports the Java 5 (aka 1.5) or later virtual machine.

Server Capabilities

Wowza Media Server Pro communicates with the Flash player client over the RTMP protocol. It enables a wide range of multimedia and interactive Flash applications. The Wowza Pro server supports flash media streaming, H.264/ACC media streaming, MP3 audio streaming, video chat and video recording. It also supports the server side component of remote shared objects. Wowza Media Server Pro enables you to implement custom application interfaces (custom modules) that are callable directly from the Flash player.

Real-Time Messaging Protocol (RTMP)

The Real-Time Messaging Protocol (RTMP) is the protocol that Wowza Media Server Pro uses to communicate with the Flash player client. Wowza Media Server Pro 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 Pro. Wowza Media Server Pro 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).

Real-Time Streaming/Transport Protocols (RTSP/RTP)

Wowza Media Server Pro supports the Real-Time Streaming Protocol (RTSP) and Real-time Transport Protocol (RTP) for incoming streaming of H.264/HE-AAC content. This enables the use of non-RTMP based live media encoders with Wowza Pro. These streams are then translated (not transcoded) as needed for delivery to the Flash player client. For details on supported RTSP/RTP encoders visit the Wowza Media Server Forums at http://www.wowzamedia.com/forums and choose the "Live Encoders" forum.

MPEG Transport Streams (MPEG-TS)

Wowza Media Server Pro supports MPEG Transport Streams (ISO 13818-1, H.222.0, MPEG-TS) for incoming streaming of H.264/HE-AAC content. This enables the use of non-RTMP based live media encoders with Wowza Pro. These streams are then translated (not transcoded) as needed for delivery to the Flash player client. For details on supported MPEG-TS encoders visit the Wowza Media Server Forums at http://www.wowzamedia.com/forums and choose the "Live Encoders" forum.

Video and Audio Streaming

Wowza Media Server Pro can stream video and audio content in Flash video format (.flv), H.264/ HE-AAC media format (.f4v, .mp4, .m4a, .mov, .mp4v, .3gp, and .3g2) and MP3 format (.mp3) to the Flash player client. The server supports the streaming of all variants of video, audio and metadata that can be stored in a Flash video file, H.264/HE-AAC content stored in an MP4 (Quicktime container) file and audio and MP3 metadata stored in an MP3 audio file.

Wowza Media Server Pro can also be used to re-stream SHOUTcast and Icecast MP3 and AAC+ audio streams to the Flash player client. Wowza Pro will maintain a single connection back to the source SHOUTcast or Icecast server for each unique audio channel. Wowza Pro is also able to forward the embedded metadata such as song title and artist to the Flash player client. The SHOUTcast examples that ships with Wowza Pro illustrates these capabilities.

Remote Shared Objects (RSO)

Wowza Media Server Pro implements the server component of Remote Shared Objects. Remote Shared Objects are an extension of ActionScript objects that enables the sharing of object data between Flash movies on the same or different client machines. Shared data is synchronized by the server through an event based synchronization method. RSO's can also be persisted on the server to maintain data across client sessions.

Custom Modules (Remote Procedure Calls)

Wowza Media Server Pro can be extended by coding custom modules which are directly callable by the Flash player client. Custom modules are implemented in Java and are dynamically linked into the server at runtime. Custom modules are a replacement for FMIS's server-side scripting capabilities.

Custom modules can also be used to extend the streaming capabilities of the server. These modules have full access to the video and audio stream at the packet level for both packets entering and leaving the server. This level of access enables deep integration with other media servers or media delivery systems such as SHOUTcast, PBX phone systems, video surveillance systems and many others.

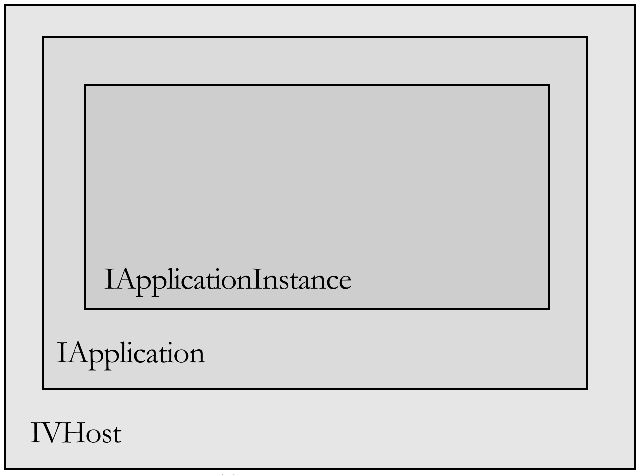
The custom module interface can also be used to integrate Wowza Media Server Pro with other servers or backend APIs directly through Java, Service-Oriented Architecture Protocol (SOAP), remote procedure calls (RPC), the Java Native Interface (JNI) and Java Database Connectivity (JDBC).

Server Architecture and Hierarchy

Wowza Media Server Pro is a pure Java server. It is written in Java 5 (aka 1.5) and may be extended dynamically using custom modules. Wowza Pro can be deployed in any environment that supports the Java 5 virtual machine or later. Wowza Pro is implemented to be tight, small and embeddable. Much of the base functionality is encapsulated in modules (jar files) that can be omitted if that functionality is not being utilized to trim the overall footprint and secure the application.

All logging for the application is done using the log4j logging component and full access is given to the configuration properties file. By default the server is setup to log to both the server console in a stripped down format as well as to log files using the W3C Extended Common Log Format (ECLF).

At the top of the object hierarchy for the server is a virtual host (IVHost) object. Each virtual host contains a set of application (IApplication) objects and each application contains a set of application instance (IApplicationInstance) objects. All client (IClient) objects, media streams (IMediaStreams) objects and remote shared object (IRemoteSharedObject) objects are children of an application instance object.



Top level object hierarchy

From the Flash player client, an application connects to a specific IVHost/IApplication/IApplicationInstance object through the NetStream connection url. For example:

```
var nc:NetConnection;
nc = new NetConnection();
nc.connect("rtmp://wowza.mycompany.com/myapplication/myinstance");
```

The first part of the url determines the protocol variant that is to be used (rtmp, rtmpt, rtmpe, rtmpte or rtmps). The domain portion "wowza.mycompany.com" determines to which virtual host to connect. The domain portion can also contain a port number in the form [domain]:[port] ("wowza.mycompany.com:80"). The "myapplication" portion specifies the application name and "myinstance" specifies the application instance name. The application instance name can be omitted. If the instance name is omitted it will connect to the "_definst_" application instance.

Note

If the port number is omitted in the connection string, the given protocol variant's default port number is used. The protocol variants have the following default port numbers; rtmp & rtmpe: 1935, rtmpt & rtmpte: 80 and rtmps: 443.

Once connected to a specific application instance, a client side application can create or connect to video and audio streams by creating a new NetStream object. It can connect to or create a new remote shared object using the SharedObject.getRemote() interface and can call remote procedures in a custom module using the NetConnection.call() interface. Examples of how this is done are presented later in this document.

Wowza Pro Server Editions

Wowza Media Server Pro comes in five editions: Pro10, Pro Unlimited, Pro Unlimited with MPEG-TS, Software Subscription and Wowza Pro Unlimited with MPEG-TS for Amazon EC2. The Pro10 and Pro Unlimited editions differ only in the number of concurrent connections the server can handle (10 and unlimited respectively) and licensing rights (see the Wowza Pro EULA for more information); all other functionality is exactly the same. The Pro Unlimited and Pro Unlimited with MPEG-TS editions differ only in the addition of support for incoming H.264 streams via MPEG-TS to the Pro Unlimited with MPEG-TS edition and licensing rights (see the Wowza Media Server Pro EULA for more information); all other functionality is exactly the same. The Pro Software Subscription and Pro for EC2 editions have the exact same functionality as the Pro Unlimited with MPEG-TS edition and provide different licensing terms (see the Wowza Pro EULA and the Wowza Media Server Pro Unlimited for Amazon EC2 EULA, respectively, for more information). The Wowza Pro Unlimited with MPEG-TS for Amazon EC2 edition is a pre-configured version of Wowza Pro Unlimited with MPEG-TS 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 Administration

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

Wowza Media Server Pro is a small and powerful Java application. 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 Media Server Pro 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 there server needs to continue to run even after you log off the machine or be automatically started when the server is rebooted. This chapter explains how to administer Wowza Media Server Pro.

Before Installation

Wowza Media Server Pro is a Java 5 (aka 1.5) application. To run, it requires the installation of a Java 5 or greater runtime environment with the exception of the software subscription licensed version which requires Java 6 (aka 1.6) or greater. To develop server side applications, a Java Development Kit (JDK) version 5 or later is required. The server also implements a JMX management and monitoring interface that requires a JMX based console on any machine that is going to be used to monitor the Wowza Pro server. One of the more popular JMX consoles is JConsole, which ships with many Java vendor's JDKs. You can also monitor the server using the JMX perspective that ships with the Wowza IDE. The Java Development Kit also includes the "server" runtime environment. The "server" runtime environment is a better choice when running Wowza Pro in a production environment.

So what does this all mean? If you are developing server side applications, are deploying the server in a production environment or are going to monitor a local or remote Wowza Pro server on a machine, you need to install Java Development Kit version 5 (aka 1.5) or greater (Java 6 if subscription license). If you are simply deploying Wowza Media Server Pro on a machine, then you need only install a Java runtime environment version 5 (aka 1.5) or greater (Java 6 if subscription license).

Note

We suggest that you deploy Wowza Pro under the most recent version of either the Java Development Kit (JDK) or Java Runtime Environment (JRE) available on your platform.

Note

If running Wowza Pro under the Java Development Kit (JDK) environment, see the notes in the following forums threads for more information on how to configure Wowza Pro to use the "server" runtime environment:

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

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 1.5.

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 Pro 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 Pro on Windows, double check to be sure the JAVA_HOME variable is pointing to a Java 5 (aka 1.5) or greater Java environment. Also, 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 the Wowza Pro server is installed using an installer. On Linux, Solaris and other Unix based platforms, the software is installed using a self extracting binary installer.

Windows

To install Wowza Media Server Pro 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.

To uninstall, choose "Uninstall Wowza Media Server Pro" from the "Start>Programs>Wowza Media Server Pro" menu.

Mac OS X

To install Wowza Media Server Pro 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 Pro 1.7.2     - server startup/shutdown scripts
                                               & documentation
/Library/WowzaMediaServerPro                   - server application files and
                                               folders: applications, bin, conf,
                                               content, examples, lib and logs
/Library/LaunchDaemons                         - background service script
                                               com.wowza.WowzaMediaServerPro.plist
/Library/Receipts                              - installer receipt file
                                               WowzaMediaServerPro-1.7.2.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/WowzaMediaServerPro/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 Pro 1.7.2
folder:     /Library/WowzaMediaServerPro-1.7.2
symlink:    /Library/WowzaMediaServerPro
file:       /Library/LaunchDaemons/com.wowza.WowzaMediaServerPro.plist
file:       /Library/Receipts/WowzaMediaServerPro-1.7.2.pkg
```

Linux

To install on Linux systems follow the steps below:

Red Hat Package Manager Systems

```
sudo chmod +x WowzaMediaServerPro-1.7.2.rpm.bin
sudo ./WowzaMediaServerPro-1.7.2.rpm.bin
```

To uninstall:

```
sudo rpm –e WowzaMediaServerPro-1.7.2-ga
```

Debian Package Manager Systems

```
sudo chmod +x WowzaMediaServerPro-1.7.2.deb.bin
sudo ./WowzaMediaServerPro-1.7.2.deb.bin
```

To uninstall:

```
sudo dpkg –-purge wowzamediaserverpro
```

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/WowzaMediaServerPro-1.7.2" 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/WowzaMediaServerPro/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 "WowzaMediaServerPro-1.7.2.tar.bin" to any directory, and execute the self extracting installer:

```
sudo chmod +x WowzaMediaServerPro-1.7.2.tar.bin
sudo ./WowzaMediaServerPro-1.7.2.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/WowzaMediaServerPro-1.7.2" 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/WowzaMediaServerPro/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 WowzaMediaServerPro-1.7.2
```

Default TCP and UDP Ports

Before streaming with Wowza Pro it is important that you open the following ports on your firewall. The table below represents the defaults ports Wowza Pro uses for streaming. All of these port numbers are configurable through the configuration files described later in this document.

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

Starting and Stopping the Server

Windows: Standalone

On Windows, Wowza Media Server Pro can be started and stopped from a DOS command prompt, from the "Start" menu or from the Windows "Services" administrative tool. To start the server from a DOS command prompt, open a DOS command prompt. Change directory ("cd") to the "bin" directory of the server installation. The default location for this folder is:

```
cd %WMSAPP_HOME%\bin
```

To start the server, type in ".\startup.bat" and hit return. The startup script will open a new console window that contains all the server logging statements. To shutdown the server close the console window by clicking on the close box or by issuing the ".\shutdown.bat" command from the DOS command prompt. The server can also be started and stopped from the "Start" menu using the "Server Startup" and "Server Shutdown" menu items in the "Programs>Wowza Media Server Pro" program group.

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 Pro" 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 Pro can interact with the underlying operating system. For example you might not be able to connect to Wowza Pro 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 Pro 1.7.2" or by opening a "Terminal" window and entering the following commands:

```
cd /Library/WowzaMediaServerPro/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.WowzaMediaServerPro.plist
```

To stop the service, enter:

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

Linux: Standalone

On Linux and Mac OS X the server can either be started from a command shell or run as a service. To start the server from command shell, enter the following commands:

```
cd /usr/local/WowzaMediaServerPro/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 WowzaMediaServerPro start
```

or

```
/etc/init.d/WowzaMediaServerPro start
```

To stop the service, enter one of these two commands:

```
/sbin/service WowzaMediaServerPro stop
```

or

```
/etc/init.d/WowzaMediaServePro 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 Pro stores serial number information in the following file (on each of the platforms):

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

To change the serial number, edit this file 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.

Server Configuration

The server is configured through a set of XML, configuration and properties files in the "conf" folder of the main applications folder. These configuration files are read during server startup. The configuration files can be directly edited using a standard text editor.

Note

For up to date tuning information see the "General Tuning Instructions" forum thread:

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

Server.xml

The Server.xml configuration file is used to configure the server container environment.

CommandInterface/HostPort – DomainName or IpAddress and Port

The ip address and port used for the command interface to the server. The command interface is a direct socket connection interface that is used by the "BootStrap" class to shutdown and restart a running Wowza Pro server. For secure deployment of the server it may be desirable to omit this section of the Server.xml file. If omitted the server will function properly but will no longer respond to shutdown and restart commands.

JMXRemoteConfiguration, AdminInterface

Configuration for the remote Java Management Extensions (JMX) interface. See the "Server Management Console and Monitoring" chapter for more information.

UserAgents

A "|" (pipe) delimited list of browser user agents that when encountered are interpreted as RTMPT/RTMPTE/RTMPTS connections.

TransportThreadPool/PoolSize, HandlerThreadPool/PoolSize

TransportThreadPool/PoolSize and HandlerThreadPool/PoolSize defines the maximum size of the server level threads in the transport and handler thread pools. The transport thread pool is used to read/write data from the transport sockets. The handler thread pool is used to process incoming messages. The Server level thread pools are only used if a virtual host's thread pool size is set to 0. This server level thread pool is also used to process the shutdown command. For this reason it should never be set to a value less than 10.

RTP/ DatagramStartingPort

RTP/DatagramStartingPort is lowest UDP port value assigned to incoming UDP streams. Ports are assigned starting and this value incrementing by 1. The most common value for RTSP/RTP based servers is 6970. If you plan on supporting RTSP/RTP, native RTP or MPEG-TS streams it is best to open up UDP ports 6970-9999.

ServerListeners/ServerListener - BaseClass

ServerListeners is a list of Java classes that are loaded by the Wowza Pro server at server initialization and notified of events during the server lifecycle. These custom classes can be used to extend the server to add functionality such as a SOAP interface or integration with a servlet container. Consult the com.wowza.wms.server.IServerNotify2 class in the Wowza Media Server Pro Server Side API documentation for details.

VHostListeners/VHostListener - BaseClass

VHostListeners is a list of Java classes that are loaded by the Wowza Pro server at server initialization and notified of events during the server lifecycle. These custom classes can be used to monitor the starting and stopping of virtual hosts and can be used to rewrite the connection information on per connection basis. Consult the com.wowza.wms.vhost.IVHostNotify class in the Wowza Media Server Pro Server Side API documentation for details.

VHosts.xml

The VHosts.xml configuration file is used to define virtual host environments. By default the server ships with a single virtual host environment named _defVHost_. A complete description of this configuration file can be found in the "Virtual Hosting" chapter of this document.

VHost.xml

The VHost.xml configuration file is used to control the overall workings of a virtual host. It is used to set server ports and ip addresses as well as to configure the thread pool size. Below is a description of each of the settings in the VHost.xml file.

HostPortList/HostPort – DomainName or IpAddress, Port and SSLFactoryClass

The list of ip addresses and ports the server is going to listen on for incoming connections. You can also provide the SSL class that is used to provide SSL handshake and encryption services. There are four child elements that are used to define a host port: "DomainName", "IpAddress", "Port" and "SSLFactoryClass". The "DomainName" and "IpAddress" are mutually exclusive. If "DomainName" is specified the server will use DNS lookup to determine the ip address the server will use for this connection. If a "DomainName" or "IpAddress" of "*" (asterisk) is specified the server will listen on all local ip addresses for incoming connections. A non-SSL connection can accept RTMP, RTMPE, RTMPT , RTMPTE, RTSP and HTTP connections. An SSL connection can only accept RTMPS connections.

HostPortList/HostPort/ProcessorCount

This is the number of threads used to service incoming requests over this socket connection. See the "General Tuning Instructions" forum thread for up to date tuning suggestions.

HostPortList/HostPort/SocketConfiguration – ReuseAddress, ReceiveBufferSize, SendBufferSize, KeepAlive and AcceptorBackLog

This section is the detailed configuration of the socket connection that is created by this HostPort definition at runtime. It is through these settings that you can tune the performance of the socket connections that will be used to send data into and out of the Wowza Pro server. The SendBufferSize and ReceiveBufferSize are the two most important settings in this group. They define the size of the memory buffers used during data transfer over the socket connection. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

The ReuseAddress and KeepAlive settings should both be set to true and are only provided for completeness.

The AcceptorBackLog setting controls the maximum number of TCP connection requests that can be pending before new connection requests are refused. The Wowza Pro server will respond to TCP connection requests as quickly as possible. This value should not be set to a value less than 50. It can be set to a value of -1 which will allow the operating system to control the value (this is not always the best idea, some platforms will then use a very small value for this which will greatly increase connection times).

HostPortList/HostPort/HTTPProvider – BaseClass and Properties

This section references a custom Java class that will be used to service incoming HTTP requests over this HostPort. The Wowza Pro server ships with three HTTPProvider classes:

```
com.wowza.wms.http.HTTPServerVersion      Returns version number
com.wowza.wms.http.HTTPConnectionInfo     Returns connection info
com.wowza.wms.http.HTTPServerInfoXML      Returns detailed info in XML
```

The HTTPServerVersion class returns in HTML the current server version. The HTTPConnectionInfo class returns the current number of connections to the server in the form "server=#". This class can be used to provide load balancing information to the Flash client. The "HTTPServerInfoXML" class returns detailed connection information in XML. Consult the com.wowza.wms.http. IHTTPProvider class in the Wowza Media Server Pro Server Side API documentation for details on how to create your own HTTPProvider class.

TransportThreadPool/PoolSize, HandlerThreadPool/PoolSize

TransportThreadPool/PoolSize and HandlerThreadPool/PoolSize defines the maximum size of the virtual host level threads in the transport and handler thread pools. The transport thread pool is used to read/write data from the transport sockets. The handler thread pool is used to process incoming messages. If the pool size is set to zero for a given thread pool type, the server level thread pool of the same type will be used for this virtual host. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

IdleWorkers – WorkerCount, CheckFrequency

IdleWorkers/WorkerCount controls the number of threads being used to generate idle events. IdleWorkers/CheckFrequency is the time in milliseconds between checking to see if a client has been idle for Client/IdleFrequency. The IdleWorkers/CheckFrequency should be at least four times smaller than the Client/IdleFrequency. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

NetConnections – ProcessorCount, IdleFrequency

NetConnections/ProcessorCount is the number of threads used to service outgoing connections between Wowza Pro servers. NetConnections/IdleFrequency is the time in milliseconds between NetConnection idle events. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

NetConnection/SocketConfiguration – ReuseAddress, ReceiveBufferSize, SendBufferSize and KeepAlive

This section is the detailed configuration of the socket connections used between Wowza Pro servers. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

HTTPTunnel/KeepAliveTimeout

This is the keep alive time for RTMPT, RTMPTE and RTMPS connections.

Client - ClientTimeout, IdleFrequency

Client/ClientTimeout is the time in milliseconds the server will wait before shutting down a non-responding client connection. Client/IdleFrequency is the time in milliseconds between idle events. For basic video on demand streaming a value of 250 milliseconds will provide the best reliability versus performance ratio. For live streaming a value of between 125 and 250 milliseconds is more desirable. It will increase the frequency at which media data is sent to the Flash client. If you adjust this value, be sure to also adjust the IdleWorkers/IdleFrequency to a value that is at least four times smaller.

RTP/ DatagramConfiguration – ReuseAddress, ReceiveBufferSize, TrafficClass and MulticastTimeout, [Unicast|Multicast][Incoming|Outgoing]/ProcessorCount

This section is the detailed configuration of the UDP sockets connections used between Wowza Pro and RTP based encoders. The ReceiveBufferSize is the two most important settings in this group. It defines the size of the memory buffers used during data transfer over the socket connection. See the "General Tuning Instructions" forum thread for up to date tuning suggestions for these settings.

The ReuseAddress, TrafficClass and MulticastTimeout settings only provided for completeness.

The "ProcessorCount" values associated with "UnicastIncoming", "UnicastOutgoing", "MulticastIncoming" and "MulticastOutgoing" control the number of threads used to service UDP connections associated with RTP and MPEG-TS streaming. These values are not used at this time and are here for future capabilities that will be added to future versions of the Wowza Pro software.

Application/ApplicationTimeout
The time in milliseconds the server will wait before shutting down an application to which no clients are connected. A value of zero will keep applications running until the virtual host is shutdown.

Application/PingTimeout
The RTMP protocol includes a connection ping mechanism. This timeout is the maximum time in milliseconds Wowza Pro will wait for a ping response from a client.

Application/ValidationFrequency
If a connected Flash client has not sent data to the Wowza Pro server in the time defined by this property (in milliseconds), Wowza Pro will send an RTMP ping message to the client to make sure the client connection is still valid and listening.

Application/MaximumPendingWriteBytes
The maximum number of bytes that can be queued up to be sent to a client before the client is disconnected. Set this value to zero to turn off this check. The pending bytes queue is checked during the client validation process.

Application/MaximumSetBufferTime
The maximum number of milliseconds honored server side for client side call to NetStream.setBufferTime(secs). Set this value to zero to turn off this check. The default value is 60000 (or 60 seconds). This is to combat Replay Media Catcher which will set a very large client side buffer to trick the server into sending all the media data at once. This can cause the server to consume a large amount of Java heap memory.

Properties/Property – Name, Value
Properties in the form of name value pairs can be attached to a virtual host definition. These properties are available in the server side API through the IVHost.getProperties() interface.

Streams.xml
The Streams.xml configuration file is used to define the server side stream types (server side NetStream implementations). Below is a description of each of the settings in the Streams.xml file.

Stream – Name, Description, ClassBase, ClassPlay
A stream definition consists of a "Name", "Description", "ClassBase", "ClassPlay". The "Name" element must be unique and is the identifier that is used to reference the stream type in the Application.xml file (described below) as well as from the Flash player client. The "Description" element is only used for debugging purposes. The "ClassBase" and "ClassPlay" define the Java classes that are going to be instantiated to service this stream type. The concept of stream types is described below in the "Client Side Scripting" chapter of this document.

Properties/Property – Name, Value

Properties in the form of name value pairs can be attached to each stream type definition. These properties are available in the server side API through the IStream.getProperties() interface.

Note

Wowza Pro includes three different methods for performing a seek operation on a media stream; "videoKeyFrame", "audio" and "enhanced". The seek method is defined by the "seekTarget" property of the "default", 'record" and "file" stream types. The "videoKeyFrame" method (which is the default) will seek to the closest key frame. The "audio" method will seek to the closest audio packet and will use the previous video key frame as the video to initially display and will begin video playback when it reaches the next video key frame. The "enhanced" method (which only works with Flash player 9,0,0 or greater) will seek to closest frame and if needed will generate a key frame. The "enhanced" method consumes the most system resources.

MP3Tags.xml

The MP3Tags.xml configuration file is used to define the property names that are used in the onId3(var info:Object) info object when playing an MP3 file. Each of the ID3V2 tags that are embedded in an MP3 file is identified by a four character identifier (you can find a complete list of the standard identifiers at http://www.id3.org/). This configuration file is used to map these four character identifiers to more meaningful names.

MediaReaders.xml

The MediaReaders.xml configuration file is used to define the Java classes that are used to read the media file formats such as Flash media, H.264/HE-AAC and MP3 files. It can also be used to configure custom file extensions for any media type.

MediaWriters.xml

The MediaWriters.xml configuration file is used to define the Java classes that are used to write recorded flv files. This configuration file provides a means for defining your own classes that will be invoked when media files are written by the server.

RTP.xml

The RTP.xml configuration file is used to define the Java classes that are used to translate raw RTP media packets into Flash media packets.

Authentication.xml

The Authentication.xml configuration file is used to define the Java classes and settings that are used to secure RTSP connections to the server. By default there are three authentication methods: none (no authentication), basic (password and username are sent in clear text) and digest (password is hashed using MD5 and is never sent in clear text over the network). Usernames and passwords are stored in the file [install-dir]/conf/rtp.password. The format of this file is a line per user with the username first followed by a space followed by the password. The authentication method (RTP/Authentication/Method) can be set for an entire virtual host in VHost.xml or on an application by application basis in Application.xml.

MediaCasters.xml

The MediaCasters.xml configuration file is used to define services that connect to other streaming servers to provide content for Wowza Media Server Pro. An example of one such service is SHOUTcast. This is also the configuration file used to configure the live stream repeater.

log4j.properties

The log4j.properties file is used to configure server logging. The server uses the Java based log4j logging system. By default the server is configured to log basic information to the console window and detail information in W3C Extended Common Log Format (ECLF) to log files. Detailed information on how to configure the logging system can be found in the "Logging" section of this chapter.

Application.xml

The Application.xml configuration file found at the root of the "conf" folder is the default application configuration file. The next section describes how application configuration works.

Runtime Configuration

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\WowzaMediaServerPro-Service.conf (Windows)

The bin\WowzaMediaServerPro-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/WowzaMediaServerPro/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/WowzaMediaServerPro/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
```

/etc/WowzaMediaServerPro/WowzaMediaServerPro-Service.conf (Linux)

The /etc/WowzaMediaServerPro/WowzaMediaServerPro-Service.conf is the configuration file used when the server is invoked as a service. The most important settings in this file are:

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

Application Configuration

Application configuration is done through an application configuration XML file. When a Flash client makes a request to the Wowza Pro server, the Wowza Pro server goes through the following procedure ([application] is the name of the application to which the client is connecting):

1. Check for an application folder named "[install-dir]/applications/[application]". If this folder is present it will proceed to step 2. If this folder is not present the connection will be terminated.

2. Check for the application configuration file "[install-dir]/conf/[application]/Application.xml". If this file is present it will load it and will not proceed to step 3. If this file is not present it will proceed to step 3.

3. Load the application configuration file "[install-dir]/conf/Application.xml".

Application.xml

Below is a description of each of the settings in the Application.xml file.

Application/ApplicationTimeout

The time in milliseconds the server will wait before shutting down an application to which no clients are connected. A value of zero will keep applications running until the virtual host is shutdown. If this value in not provided (section commented out) the value set in the VHost.xml will be used.

Connections/AutoAccept

Possible values are "true" or "false". This setting determines if the application will automatically accept incoming connection request. If "true" all incoming connection request will automatically be accepted. If "false" the application is required to make a server side call to "client. acceptConnection()" to accept an incoming connection request (see the "Creating a Custom Module" chapter for details).

Connections/AllowDomains

Connections/AllowDomains is a comma delimited list of domain names or ip address for which client connections will be accepted. The domain names or ip addresses that are specified here represent the domain name or ip address of the Flash swf file that is connecting to the Wowza Pro server or the ip address of the client connecting to Wowza Pro. If this value is left empty then connections from all domains or ip addresses are accepted. For example if you have a .swf file that is located at the url:

```
http://www.mycompany.com/flash/myflashmovie.swf
```

To configure your server such that only content from your domain can access your Wowza Pro server you would set AllowDomains to www.mycompany.com. You can also add an ip address (or ip address wildcard) to accept all connections from a particular ip address. You might flilter based on ip address when you are working with a client side encoder such as On2 Flix Live which does not provide a valid referrer.

You can use the wildcard "*" to match partial domain names or ip addresses. For example if you would like to match all domain names that end with mycompany.com you would specify the domain name *.mycompany.com.

The allow domains processing occurs just before the event method onConnect. So if you would like to provide more fine grained access control to your server, you can override the onConnect event handler in a custom module and provide your own filtering mechanism.

Streams/StreamType

The name (as defined in the Streams.xml file) of the default stream type for this application. An explanation of stream types can be found in the "Stream Types" section of "Client Side Scripting" chapter of this document.

Streams/StorageDir and SharedObjects/StorageDir

Streams/StorageDir is the full path to the directory where this application will read and write its stream files (.flv) to and from. SharedObjects/StorageDir is the full path to the directory where this application will read and write its remote stored object files to and from. If these values are left blank, an application will use the following directories as its Streams/StorageDir and SharedObjects/StorageDir:

```
%WMSCONFIG_HOME%/applications/[application]/streams/[appinstance]
%WMSCONFIG_HOME%/applications/[application]/sharedobjects/[appinstance]

%WMSCONFIG_HOME%            the value of the environment variable WMSCONFIG_HOME
[application]               the name of the application
[appinstance]               the name of the application instance
```

There are several dynamic properties that can be used as part of the StorageDir path using the syntax ${[variable-name]}. The following properties are available:

```
${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
```

For example the default Streams/StorageDir can be specified using the path:

```
${com.wowza.wms.ConfigHome}/applications/${com.wowza.wms.context.Application}/
       streams/${com.wowza.wms.context.ApplicationInstance}
```

Streams/Properties

Streams/Properties are property values that override values defined in [install-dir]/conf/Streams.xml on a per-application basis. For example, to turn on enhanced seek for any of the video on demand stream types, add the property "seekTarget" to this property collection and set the value to "enhanced".

Client/IdleFrequency

Client/IdleFrequency is the time in milliseconds between idle events. If this value is set to -1 then the value specified in VHost.xml will be used. If a value other than -1 is specified it will override the value specified in VHost.xml for all clients connecting to the application defined by this Application.xml file. See the VHost.xml description of this property for more information.

Client/Access – StreamReadAccess, StreamWriteAccess, StreamAudioSampleAccess, StreamVideoSampleAccess, SharedObjectReadAccess and SharedObjectWriteAccess

The Client/Access configuration parameters controls the default access a client connection has to assets connected to a particular Wowza Pro application. An individual client's access can be modified through the server side API. This is most commonly done in the onConnect or onConnectAccept event handler. Each of these settings is a comma delimited list of names that are matched against the asset name (stream name or shared object name) to control access. If any part of the asset name matches one of the elements in the list match then the given access is granted. The values are case sensitive. If the parameter is empty (blank) then access is denied to all clients. If the parameter is set to the "*" character, then access is granted to all clients. For example if StreamReadAccess is set to "testa/testb;testc", then the following stream name would be granted the following access:

```
testc                 Granted Access
testc/test            Granted Access
testC/test            Denied Access (incorrect case)
testa/testb           Granted Access
testa/testb123        Granted Access
testa/testb/file123   Granted Access
testa/test            Denied Access (incomplete match)
```

StreamReadAccess: controls access to view or listen to a NetStream object.

StreamWriteAccess: controls access to write or publish to a NetStream object.

StreamVideoSampleAccess: controls access to call BitmapData.draw() to take a snapshot of a NetStream object.

StreamAudioSampleAccess: controls access to call SoundMixer.computeSpectrum() to grab the waveform data of a NetStream object.

SharedObjectReadAccess: controls access to read values from a RemoteSharedObject.

SharedObjectWriteAccess: controls access to write values to a RemoteSharedObject.

RTP/Authentication/Method

The authentication method used to secure RTSP connections to Wowza Pro. Authentication methods are defined and configured in Authentication.xml. By default there are three authentication methods: none (no authentication), basic (password and username are sent in clear text) and digest (password is hashed using MD5 and is never sent in clear text over the network). Usernames and passwords are stored in the file rtp.password. The format of this file is a line per user with the username first followed by a space followed by the password. The authentication method can also be set at the virtual host level in VHost.xml.

RTP – AVSyncMethod, MaxRTCPWaitTime

These two settings control how Wowza Pro synchronizes the audio and video channels when receiving a RTP stream. AVSyncMethod configures the methodology used to synchronize the audio and video channels. There are three possible values; senderreport (use the Sender Report (SR) packets that are sent over the Real-time Control Protocol (RTCP) channel), rtptimecode (assume the RTP timecodes are absolute timecode values), systemclock (synchronize based on the system clock). The default value is senderreport. MaxRTCPWaitTime is the maximum time in milliseconds Wowza Pro will wait to receive a Sender Report (SR) packet over the Real-time Control Protocol (RTCP) channel. If not SR packets are received within this time period the server will default to using the rtptimecode method.

RTP/Properties

RTP/Properties are property values that override values defined in [install-dir]/conf/RTP.xml on a per-application basis.

MediaCaster/Properties

MediaCaster/Properties are property values that override values defined in [install-dir]/conf/MediaCasters.xml on a per-application basis. For example, to set the stream time out value for any of the media caster types, add the property "streamTimeout" to this property collection and set it to a non-zero value.

MediaReader/Properties

MediaReader/Properties are property values that override values defined in [install-dir]/conf/MediaReaders.xml on a per-application basis.

Modules/Module – Name, Description and Class

The modules section is a list of modules that are available to this application. The "Name" and "Description" elements are only for logging and debugging. The "Class" element is the full package name and class name of the module. Please see the "Server Side Modules" chapter of this document for information on configuring modules.

Properties/Property – Name, Value

Properties in the form of name value pairs can be attached to an application definition. All application properties are copied to child application instances upon instance creation. These properties are available in the server side API through the IApplicationInstance.getProperties() interface.

Logging

Wowza Media Server Pro 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 Media Server Pro 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 Pro 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))
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 Pro 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 on a stream
pause	Play has paused on a stream
unpause	Play has unpaused on a stream
seek	Seek has occurred on a stream
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 Pro generates the following logging status values:

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

Wowza Media Server Pro 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 Media Server Pro.

```
# create log appenders stdout and R
log4j.rootCategory=INFO, stdout, R

# 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.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.DatePattern='.'yyyy-MM-dd
log4j.appender.R.File=${com.wowza.wms.ConfigHome}/logs/wowzamediaserverpro_access.log
log4j.appender.R.layout=com.wowza.wms.logging.ECLFPatternLayout
log4j.appender.R.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.R.layout.OutputHeader=true
log4j.appender.R.layout.QuoteFields=false
log4j.appender.R.layout.Delimeter=tab
```

Note

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

In this example the logging properties file has been simplified to highlight a few key features. The first statement in this file sets the logging level to "INFO" and defines two appenders; "stdout" and "R". 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 are: 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 Pro 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]/wowzamediaserverpro_access.log
[install-dir]/logs/[vhost]/wowzamediaserverpro_error.log
[install-dir]/logs/[vhost]/wowzamediaserverpro_stats.log
```

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

```
[install-dir]/logs/[vhost]/[application]/wowzamediaserverpro_access.log
[install-dir]/logs/[vhost]/[application]/wowzamediaserverpro_error.log
[install-dir]/logs/[vhost]/[application]/wowzamediaserverpro_stats.log
```

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

Server Security

The default installation of Wowza Media Server Pro 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 the Wowza Pro server on a lowered numbered ports such as 80 (the http port) or 443 (the https port), 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 Media Server Pro installation files.

```
chown wowza:wowza WowzaMediaServerPro
chown –R wowza:wowza WowzaMediaServerPro-1.7.2
chmod –R 775 WowzaMediaServerPro-1.7.2
rm –f /var/run/WowzaMediaServerPro.pid
rm –f /var/run/WowzaMediaServerPro.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/WowzaMediaServerPro/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".

If you have started Wowza Pro as a service running as root, then you will need to execute the following command to clear the run files:

```
rm –rf /var/run/WowzaMediaServerPro*
```

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 Media Server Pro installation files. Open a "Terminal" window and enter the following commands:

```
cd /Library
sudo chown wowza:admin WowzaMediaServerPro
sudo chown –R wowza:admin WowzaMediaServerPro-1.7.2
```

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/WowzaMediaServerPro/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 3

Wowza Pro in Action

How do I start streaming using Wowza Media Server Pro?

Wowza Media Server Pro can be used to deliver streaming video to many user's in a multiple server deployment. Below we cover several topics as they relate to delivering video on demand and live content in such an environment.

H.264/HE-AAC Streaming with Non-Flash Encoders (RTSP/RTP/MPEG-TS)

Wowza Media Server Pro supports the Real-Time Streaming Protocol (RTSP) and Real-time Transport Protocol (RTP) for incoming streaming of H.264/ HE-AAC content. This enables the use of live encoders such as Telestream Wirecast and Apple QuickTime Broadcaster. This section covers the basic Wowza Pro features as they relate to RTSP/RTP streaming. For up to date, step by step instructions on how to setup and use Wowza Pro with live encoders, visit the Wowza Media Server Forums at http://www.wowzamedia.com/forums and choose the "Live Encoders" forum.

Wowza Pro currently 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
MPEG-TS	ISO/IEC 13818-1
MPEG-TS over RTP	rfc2038

There are two methods for delivering RTP based H.264/HE-AAC live content to Wowza Media Server Pro. The most common method is to use an encoder that supports the QuickTime announce command. Using this method the encoder creates a RTSP session with Wowza Pro and sends the Session Description Protocol (SDP) information using the announce command. The RTSP session is used to manage the RTP session startup and shutdown. The second method is a native RTP based solution. The SDP information is communicated to Wowza Pro either through the file system or an HTTP request. The following two sections will cover these two methods.

Real-Time Streaming Protocol (RTSP) Streaming

Wowza Media Server Pro natively supports the Real-Time Streaming Protocol (RTSP) for incoming streaming on H.264/HE-AAC content. This capability is enabled on any port that is defined in VHost.xml. Access to RTSP streaming is controlled through authentication. Wowza Pro supports three methods of RTSP authentication; none (no authentication), basic (password and username are sent in clear text) and digest (password is hashed using MD5 and is never sent in clear text over the network). Authentication configuration is done in VHost.xml, Application.xml and Authentication.xml. The default authentication method is "digest" which is the strongest and most secure method. Usernames and passwords are defined in the file

"[install-dir]/conf/rtp.password". Before an RTSP session can be initiated a valid username and password must be added to the rtp.password file.

An RTSP session is generally established based on four pieces of information; host address (and port), streaming path (sometimes called location), username and password. The username and password information is discussed above. The host name is the network address of the Wowza Pro server along with the network port. By default RTSP communication takes place over port 554 which is not a port on which Wowza Pro is listening. Wowza Pro by default is listening on port 1935. For this reason the host address should be set to [server-ip-address]:1935 where [server-ip-address] is the ip address of the server running Wowza Pro.

The streaming path is a unique name given to the live stream. In Wowza Pro this path is used to determine the application name, application instance name and stream name that are required for Flash streaming. The format is as follows:

```
[application]/[appinstance]/[streamname]
```

Where [application] is the application name [appinstance] is the application instance name and [streamname] is the stream name. The stream name can contain additional path elements. For example a streaming path of:

```
streamtest/myStream.sdp
```

Would be interpreted as:

```
[application]     streamtest
[appinstance]     _definst_
[streamname]      myStream.sdp
```

A streaming path of:

```
streamtest/_definst_/livevideos/myStream.sdp
```

Would be interpreted as:

```
[application]     streamtest
[appinstance]     _definst_
[streamname]      livevideos/myStream.sdp
```

The actual video and audio data is transmitted to the Wowza Pro server in one of two ways; 4 separate UDP ports or interleaved over the RTSP TCP connection. Most encoders default to UDP transmission. When using UDP transmission, the encoder and Wowza Pro will negotiate a set of ports to use for RTP transmission. The UDP port range is 6970-9999. It is important that these ports be open for UDP traffic on your firewall.

Many of the RTSP/RTP encoders support a large list of video and audio codecs. When using this method of live streaming, Wowza Pro only supports H.264/AVC1 (not MPEG4) for video content and HE-AAC for audio content. It is important that you configure the encoder to encode the content using these codecs.

You can record a stream coming from a RTSP based encoder by using the "live-record" stream type. The file will be stored in the content folder that you have configured for your application. If you do not specify a stream prefix or you specify a prefix of "flv:" then the file will be recorded to an flv . If you specify a stream prefix of "mp4:" the file will be written to an mp4 container format (also called the Quicktime file format).

Native Real-time Transport Protocol (RTP) Streaming

Wowza Pro can also be configured to receive an H.264/HE-AAC stream from a native RTP stream. This method does not involve the use of a RTSP session. Instead the stream is pulled on demand through the use of one of several special stream types. The stream types that can be used to pull a native RTP stream are; rtp-live, rtp-live-record, rtp-live-lowlatency and rtp-live-record-lowlatency. When using this method, Wowza Pro supports both unicast UDP streams as well as multicast streams.

The procedure for using this method is as follows (this assumes the application name "rtplive"):

1. Create the folder "[install-dir]/applications/rtplive".

2. Create the folder "[install-dir]/conf/rtplive" and copy "[install-dir]/Application.xml" into this new folder.

3. Edit the newly copied "Application.xml" file and change the "Streams/StreamType" to "rtp-live".

4. From the encoder generate a Session Description Protocol (SDP) file that describes the native stream (consult your encoders documentation for instructions on how to do this). For this example we assume the filename "myStream.sdp".

5. Copy the SDP file into the "[install-dir]/content" folder.

6. Double click "[install-dir]/examples/NativeRTPVideoStreaming/client/live.html", set "Server" to "rtmp://[server-ip-address]/rtplive" and "Stream" to "myStream.sdp" and click the "Play" button.

It will take time for the video to be displayed for the first connection. This is due to the fact that Wowza Pro must wait until the proper signal is transmitted that synchronizes the audio and video streams (the RTCP SR packet). Wowza Pro must also wait until the first key frame is transmitted.

The video for subsequent connections to the server will be displayed much more quickly. Wowza Pro will continue to receive this stream until the last client connection has disconnected. At that time Wowza Pro will wait for a timeout period (defined by KeepAliveTime in MediaCasters.xml). If no new clients connect to this stream, the stream will be dropped and will not be restarted until another client requests the stream.

Note

The NativeRTPVideoStreaming example utilizes this method of streaming.

Note

The Session Description Protocol (SDP) information can also be made available to Wowza Pro through a URL. Using this method the stream name is "[SDP URL]". For example if the SDP information is hosted at the web address:

http://192.168.1.7/rtp/myStream.sdp

Use the stream name:

http://192.168.1.7/rtp/myStream.sdp

Native RTP streaming uses an internal streaming mechanism called MediaCasters. There are several MediaCaster properties that can be used to control how Wowza Pro monitors changes to the native RTP stream and the underlying SDP data (file or HTTP URL). These properties are: streamTimeout, sdpFileCheckFreqency and sdpHTTPCheckFreqency. By default each of these monitoring features are turned off (their values are set to zero). MediaCaster property values can be set on an application by application basis in the "MediaCasters/Properties" section of the Application.xml file. For example to set values for each of these properties, add the following XML snippet to the conf/rtplive/Application.xml file:

```
<MediaCaster>
      <Properties>
            <Property>
                  <Name>streamTimeout</Name>
                  <Value>15000</Value>
                  <Type>Integer</Type>
            </Property>
            <Property>
                  <Name>sdpFileCheckFreqency</Name>
                  <Value>2000</Value>
                  <Type>Integer</Type>
            </Property>
            <Property>
                  <Name>sdpHTTPCheckFreqency</Name>
                  <Value>10000</Value>
                  <Type>Integer</Type>
            </Property>
      </Properties>
</MediaCaster>
```

Each of these settings are described below:

streamTimeout

The streamTimeout property is measured in milliseconds. When set to a value greater than zero, Wowza Pro will monitor the incoming native RTP streams. If it does not see any audio or video packets for the duration set by this value it will force a reset of the native RTP stream.

sdpFileCheckFreqency

The sdpFileCheckFreqency property is measured in milliseconds. This value controls how often Wowza Pro will check for file modification date and file size changes to the SDP file that was used to start the native RTP stream. When a file modification date or file size change is detected, the stream will be reset and the SDP file will be re-read.

sdpHTTPCheckFreqency

The sdpHTTPCheckFreqency property is measured in milliseconds. This value controls how often Wowza Pro will check for changes to SDP data retrieved using an HTTP URL. When a SDP data change is detected, the stream will be reset and the new SDP data will be used to start the native RTP stream.

You can record a stream coming from a native RTP based encoder by using the "rtp-live-record" or "rtp-live-record-lowlatency" stream type. The file will be stored in the content folder that you have configured for your application. If you do not specify a stream prefix or you specify a prefix of "flv:" then the file will be recorded to an flv container. If you specify a stream prefix of "mp4:" the file will be written to an mp4 container format (also called the Quicktime file format).

If you would like to have more control over when the stream starts and stops and how it gets recorded, you can use the MediaCasterStreamManager AddOn package. This package includes a simple Flash client that connects to Wowza Pro and is used to start and stop native RTP streams. See this forum post for more information:

http://www.wowzamedia.com/forums/showthread.php?t=4533

MPEG Transport Stream Streaming

The Wowza Pro Unlimited with MPEG-TS edition can also be configured to receive an H.264/HE-AAC stream from an MPEG Transport Stream (MPEG-TS) encoder. This method does not involve the use of a RTSP session. Instead the stream is pulled on demand through the use of one of several special stream types. The stream types that can be used to pull a native RTP stream are; rtp-live and rtp-live-lowlatency. When using this method, Wowza Pro supports both unicast UDP streams as well as multicast streams.

The procedure for using this method is as follows (this assumes the application name "rtplive"):

7. Create the folder "[install-dir]/applications/rtplive".

8. Create the folder "[install-dir]/conf/rtplive" and copy "[install-dir]/Application.xml" into this new folder.

9. Edit the newly copied "Application.xml" file and change the "Streams/StreamType" to "rtp-live".

10. Configure the encoder to send the MPEG-TS stream to the server running Wowza Pro (unicast) or to a multicast address that is properly routed to the server running Wowza Pro.

11. Double click "[install-dir]/examples/NativeRTPVideoStreaming/client/live.html", set "Server" to "rtmp://[server-ip-address]/rtplive" and "Stream" to "udp://[ip-address]:[port]" (where [ip-address] is the ip address of the destination of the MPEG-TS stream and [port] is the UDP port) and click the "Play" button.

It will take time for the video to be displayed for the first connection. This is due to the fact that Wowza Pro must wait for a video key frame before it can send the incoming stream to the Flash player. The video for subsequent connections to the server will be displayed much more quickly. Wowza Pro will continue to receive this stream until the last client connection has disconnected. At that time Wowza Pro will wait for a timeout period (defined by KeepAliveTime in MediaCasters.xml). If no new clients connect to this stream, the stream will be dropped and will not be restarted until another client requests the stream.

You can record a stream coming from a MPEG-TS based encoder by using the "rtp-live-record" or "rtp-live-record-lowlatency" stream type. The file will be stored in the content folder that you have configured for your application. If you do not specify a stream prefix or you specify a prefix of "flv:" then the file will be recorded to an flv container. If you specify a stream prefix of "mp4:" the file will be written to an mp4 container format (also called the Quicktime file format).

Note

MPEG Transport Stream Streaming requires the Wowza Pro Unlimited with MPEG-TS edition license.

If you would like to have more control over when the stream starts and stops and how it gets recorded, you can use the MediaCasterStreamManager AddOn package. This package includes a simple Flash client that connects to Wowza Pro and is used to start and stop native RTP streams. See this forum post for more information:

http://www.wowzamedia.com/forums/showthread.php?t=4533

Load Balancing

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

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

Multiple Server Live Streaming (Live Stream Repeater)

The following example illustrates a suggested configuration and implementation for delivering a live media event across multiple Wowza Media Server Pro servers. We will walk through the configuration and deployment of the live stream repeater. The live stream repeater uses multiple Wowza Pro 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 Pro server. The Flash client code will request the content from an edge server using a special stream type and content name that will instruct the edge server to source the live stream from the origin server. Orgin and edge configuration is an application level configuration. A single Wowza Pro 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 change the Streams/StreamType to "liverepeater-origin"

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 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). Uncomment the

Repeater/OriginURL section and set OriginURL to rtmp url of the orgin 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". From your media encoder you are going to publish content to the stream name "mycoolevent" using the following connection string:

```
rtmp://origin.mycompany.com/liverepeater
```

From your Flash client code you are going to play the content using the following connection string:

```
rtmp://edge1.mycompany.com/liverepeater
```

To play the content you will use the following play command:

```
netStream.play("mycoolevent");
```

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

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.

Also on the edge servers in "[install-dir]/conf/liverepeater/Application.xml" you will need to configure the MediaCaster property "streamTimeout". This property will instruct Wowza Pro to monitor the stream on the origin. If there is a break in the stream longer than the stream timeout, then the edge will connect to the next origin in the origin list. The configuration looks like this:

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

Note

You can override the OriginURL value defined in each edge server's Application.xml file by specifying the origin as part of the stream name. The stream name in this case would take the form: liverepeater:rtmp://origin.mycompany.com/liveorigin/mycoolevent.

Note

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

Chapter

4

Server Management Console and Monitoring

How do I manage and monitor Wowza Media Server Pro?

Wowza Media Server Pro 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 Media Server Pro 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 the Wowza Pro 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 Media Server Pro 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 Pro management objects are located under the "MBean" tab in the "WowzaMediaServerPro" 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 the Wowza Pro server from a remote computer. Both the JMV and the Wowza Pro 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 Pro 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 Pro built-in JMX interface configuration

The remote JMX interface built into the Wowza Pro server can be configured through the "JMXRemoteConfiguration" and "AdminInterface" sections of the "conf/Server.xml" file. These sections 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 Pro and "RMIServerHostName" should be set to the external ip address or domain name of the machine. For example, if the server running Wowza Pro 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 the Wowza Pro 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, the Wowza Pro 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 Pro start scripts. The following scripts in the "bin" folder can be edited to enable remote JMX monitoring

```
startup.bat                        – Windows standalone startup script
WowzaMediaServerPro-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 "WowzaMediaServerProJMX".

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:\WowzaMediaServerProJMX, 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:\WowzaMediaServerProJMX\jmxremote.password
-Dcom.sun.management.jmxremote.access.file=C:\WowzaMediaServerProJMX\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 Pro 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 Pro" 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 Pro 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 Pro 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 Pro 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 Pro server 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 "WowzaMediaServerPro" 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 the Wowza Pro server. The Wowza Pro server 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 "WowzaMediaServerPro" 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.

Custom HTTP Interfaces (HTTPProvider)

Wowza Media Server Pro includes the ability to add custom HTTP interfaces to the server. These interfaces are called HTTPProviders. By default Wowza Media Server Pro is configured to use the "com.wowza.wms.http.HTTPServerVersion" HTTPProvider which returns the current Wowza Pro version and build number. You can see this in action by opening a web browser and entering the address:

```
http://[server-ip-address]:1935
```

Where [server-ip-address] is the ip address of the server running Wowza Pro. The server will respond with: "Wowza Media Server Pro [edition] [version] build[build-number]". This HTTP interface is customizable through the VHost.xml file. See the above description of the VHost.xml for more information.

Chapter

5

Client Side Scripting

How do I interact with Wowza Media Server Pro from the Flash player client?

Wowza Media Server Pro fully supports the Flash player API. Most of this interface as it relates to Wowza Pro is encapsulated by three Flash player objects: NetConnection, NetStream and SharedObject. This section will highlight where Wowza Media Server Pro implementation of this object interface differs or has been extended.

Stream Types

One of the major differences between Wowza Media Server Pro and the Adobe Flash Media Interactive Server is the way Wowza Media Server Pro handles the NetStream object on the server. Wowza Media Server Pro provides a mechanism for defining custom server side NetStream implementations or stream types. These stream types are Java classes that are dynamically bound to the server at run time. A stream type is made available to the server by defining an entry in the Streams.xml file described in the "Server Administration" chapter of this document. A stream type is uniquely identified by the value of its "Name" element.

Wowza Media Server Pro ships with several different stream types each coded and tuned to only support a narrow set of functionality. For instance the "file" stream type is only able to stream a dynamic playlist of static ".flv" content from the server to client. If your application attempted to use the "file" stream type to support video chat it would not function properly.

Wowza Media Server Pro provides several examples that highlight the usage of the different stream types; "Simple Video Streaming", "Video Recording" and "Video Chat". These examples clearly illustrate how to develop your application to use the provided stream types. The Wowza Pro stream types are:

Stream Type	Use
file, default	Video on demand streaming of static Flash media, H.264/ HE-AAC and MP3 content
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 Pro servers in an origin/edge configuration (use to configure origin application)
liverepeater-edge	Publish and play live video content across multiple Wowza Media Server Pro servers in an origin/edge configuration (use to configure edge application)
liverepeater-edge-lowlatency	Publish and play live video content across multiple Wowza Media Server Pro servers in an origin/edge configuration (use to configure edge application when latency is important)
rtp-live	Play native RTP streams (see: H.264 Streaming with Non-Flash Encoders (RTSP/RTP/MPEG-TS)).
rtp-live-lowlatency	Play native RTP streams (see: H.264 Streaming with Non-Flash Encoders (RTSP/RTP/MPEG-TS) 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

Below is a short description of how the NetStream.play and NetStream.publish client side API calls are used in conjunction with each the stream types.

file, default

The "file" stream type is used to stream a single file or dynamic playlist of static flash video content (.flv), H.264/ HE-AAC content (.f4v, .mp4, .m4a, .mov, .mp4v, .3gp, and .3g2) or MP3 audio content (.mp3) to the Flash player client. The single file or playlist is specified by making calls from the client to NetStream.play.

```
play(name, [, start [, len [, reset]]]);
```

name The name of content to play and/or add to the dynamic playlist. The server will locate the file on the file system based on the definition of the "Streams/StorageDir" setting in your application's Application.xml file (described in the "Server Administration" chapter).

Wowza Media Server Pro can play H.264/ HE-AAC and MP3 files as well as flash video files. To play a H.264/ HE-AAC file, prepend the "mp4:" qualifier to the media name (include the file extension). For example to play the file "mycoolvideo.m4v", specify the stream name "mp4: mycoolvideo.m4v". To play an MP3 file, prepend the "mp3:" qualifier to the media name. For example to play the MP3 file "mycoolsong.mp3", specify the name "mp3:mycoolsong". If you just want the server to return the MP3 ID3 tags and not play the media file, prepend the qualifier "id3:".

start The time in seconds from the beginning of the media file to start playback. A value less than or equal to zero will start play at the beginning of the media file. The default value is 0.

len The duration of playback for this media file in the playlist in seconds. A value less than zero will play the media file from value defined by start through the end of the media file. A value of 0 will just send the closest key frame to "start" (great for displaying media thumbnails). A value greater than zero will play the media for the duration of "len" seconds. The default value is -1.

reset A boolean value (true or false) which indicates if the previous playlist entries should be deleted before this entry is added. If true the current playlist associated with this NetStream object will be cleared and this entry will be added to the beginning of the playlist and start playing. If false this entry will be added to the end of the playlist. The default value is true.

For example, to stream a single .flv file with the name "mycoolvideo.flv" the command would be:

```
var ns:NetStream;
ns.play(“mycoolvideo”);
```

To play the H.264/ HE-AAC movie "mycoolvideo.m4v" the command would be:

```
var ns:NetStream;
ns.play(“mp4:mycoolvideo.m4v”);
```

To build a dynamic playlist that first plays the video "mycoolvideo1" starting 20 seconds into the video for 60 seconds then switching to "mycoolvideo2" and playing that file from the beginning to the end would be:

```
var ns:NetStream;
ns.play(“mycoolvideo1”, 20, 60, true);
ns.play(“mycoolvideo2”, 0, -1, false);
```

Playback can be controlled and monitored through client side calls to the NetStream object API.

To see this stream type in action, check out the "FastPlayVideoStreaming" and "SimpleVideoStreaming" examples.

record

The "record" stream type is used to capture video content from the Flash player client's "Camera" and "Microphone" objects to an ".flv" file on the server. Recording is controlled through client side calls to NetStream.publish.

```
publish(name [, howToPublish]);
```

name The name of flash video file to record to on the server. This name should not include the ".flv" extension. The server will locate the file on the file system based on the definition of the "Streams/StorageDir" setting in your application's Application.xml file (described in the "Server Administration" chapter). A value of "null" will stop recording, flush the remaining video content to the ".flv" file, close the file and send the "NetStream.Unpublish.Success" event to the onStatus handler of the NetStream object.

howToPublish The method for publishing to the ".flv" file. A value of "record" will cause the server to overwrite the previously existing ".flv" file with the same name. A value of "append" will append the video content.

Recording can be controlled and monitored through client side calls to the NetStream object API.

To see this stream type in action, check out the "VideoRecording" example.

live

The "live" stream type is used to publish and play live video content captured by the Flash player client's "Camera" and "Microphone" objects or from a digital video encoder. This stream type is tuned for delivering live media events that do not required a low latency connection. The "live-lowlatency" or "chat" stream types are better suited for video and audio chat applications.

Video and audio capture is controlled through client side calls to NetStream.publish and playback is controlled through calls to NetStream.play.

```
publish(name);
```

name The unique name for the published live stream. A value of "null" will stop publishing the live stream.

```
play(name, [, start [, len [, reset]]]);
```

name The name of the live stream to play.

To see this stream type in action, check out the "LiveVideoStreaming" example.

live-lowlatency

These stream types are variants of the "live" stream type that are tuned for video and audio chat applications. They are invoked in the same manner as the "live" stream type.

To see this stream type in action, check out the "VideoChat" example.

live-record, live-record-lowlatency

The "live-record" and "live-record-lowlatency" stream types are variants of the "live" and "live-lowlatency" stream types that provide media recording capabilities. Video and audio capture is controlled through client side calls to NetStream.publish and playback is controlled through calls to NetStream.play.

```
publish(name [, howToPublish]);
```

name The unique name for the published live stream. A value of "null" will stop publishing the live stream.

howToPublish The method for publishing to the ".flv" file. A value of "record" will cause the server to overwrite the previously existing ".flv" file with the same name. A value of "append" will append the video content.

Recording can be controlled and monitored through client side calls to the NetStream object API.

```
play(name, [, start [, len [, reset]]]);
```

name The name of the live stream to play.

Media can be recorded to two different container formats; flv and mp4. The container format is controlled by the stream name prefix. If the stream name prefix is "flv:" or not specified then the media will be recorded to an flv container. If the stream prefix is "mp4:" content will be recorded to a mp4 container (also called the Quicktime file format). The mp4 container can only contain H.264, AAC and MP3 media formats. All other formats such as NellyMoser, Sorenson Spark and VP6 will be omitted when recording to an mp4 container.

shoutcast, shoutcast-record

The "shoutcast" and "shoutcast-record" stream types are used to re-stream a SHOUTcast (or Icecast) stream through the Wowza Pro server as an MP3 or AAC+ stream ("shoutcast-record" in addition to re-streaming the SHOUTcast stream will record it to a file). This stream type uses the Wowza Media Server Pro's MediaCaster technology to maintain a connection to the SHOUTcast server. This system will maintain a single connection per unique SHOUTcast url. What this means is if 10 client connections are listening to the same SHOUTcast stream, the server will only maintain a single connection to the source SHOUTcast server.

Playback is controlled through client side calls to the NetStream object API. For example if you wanted to play the SHOUTcast url http://192.168.1.5/reggae the NetStream play call would be:

play("http://192.168.1.5/reggae");

Note

The urls that are needed to connect to a SHOUTcast or Icecast server are the urls that are contained within the .pls (SHOUTcast) or .m3u (Icecast) playlist files. Most websites that publish SHOUTcast or Icecast streams present links on their sites that refer to these .pls and .m3u files. So to use the SHOUTcast stream type you must first download these playlist files to your local machine, open the files with a text editor and use the links found inside to connect to their published streams.

This stream type will trigger two NetStream event handlers based on the metadata included as part of the SHOUTcast stream: onHeaderData and onMetaData. Each of these event handlers will receive a single parameter which is an object that contains the metadata for that event. The onHeaderData event handler will be triggered once at start of audio streaming. The onMetaData handler will be triggered when the metadata information for the stream (such as song title, artist or album) changes during the streaming session. See this stream type in action, try out the "SHOUTcast" example.

liverepeater-origin, liverepeater-edge, liverepeater-edge-lowlatency

These stream types are used by the live stream repeater. Consult the "Scalability for Live Streaming" section of the "Server Administration" chapter for more details.

Creating a NetStream object in Wowza Media Server Pro

There are two methods for specifying the stream type to use when creating a NetStream object in the Flash player client. First, the "Stream/StreamType" entry in your application's Application.xml defines the default stream to use each time a NetStream object is created from the player client. This value can be overridden in the Flash player client by making a remote call to "setStreamType" before creating a NetStream object in your application's client code. Below is an example:

```
var nc:NetConnection;
var ns:NetStream;

nc = new NetConnection();
nc.onStatus = function(infoObj)
{
      if (infoObj.code == "NetConnection.Connect.Success")
      {
            nc.call("setStreamType", null, "live");
            ns = new NetStream(nc);
      }
}
nc.connect("rtmp://wms.mycompany.com/myapplication/myinstance");
```

Client to Server Calls

Wowza Media Server Pro supports the same method as the Adobe Flash Media Server for making client to server side calls. From the Flash player client the NetStream.call method can be used to directly call server side methods. The signature for NetStream.call is as follows:

```
ActionScript 2:

call(handlerName [, resultObj [, param1 … param(n)]]);

ActionScript 3:

call(handlerName , responder [, param1 … param(n)]);
```

handlerName The name of the method on the server side that will be executed by this client side call.

resultObj(AS2) A reference to a result object that contains an onResult function that will be called when the server side call has completed. The default value is "null".

responder(AS3) An ActionScript3 Responder object that will receive the result of the function call. Set this value to null if the function does not return data.

param1 … param(n) Optional parameters that are passed to the server side method.

Below is an ActionScript 3 code snippet that illustrates how to call the server side method "doSomething" and to print out the resulting return value in the onResult handler.

```
var nc:NetConnection;

function doSomethingResult(returnObj)
{
	var param:String;
	for(param in returnObj)
		trace("return: "+param+"="+ returnObj[param]);
}


ncOnStatus = function(infoObj)
{
	if (infoObj.code == "NetConnection.Connect.Success")
	{
		nc.call("doSomething", new Responder(doSomethingResult), "test param1");
	}
}

nc = new NetConnection();
nc.addEventListener(NetStatusEvent.NET_STATUS, ncOnStatus);
nc.connect("rtmp://wms.mycompany.com/myapplication/myinstance");
```

The "Creating a Custom Module" chapter of this document will document how to create server side methods in Java.

Content Protection (SecureToken, SecureURLParams...)

One of the main advantages that streaming media content has over progressive download is content protection and security. Simply streaming your content using the Flash client and Wowza Media Server Pro does not always provide sufficient security against products that are built to intercept streaming media. The Wowza Pro server includes several features and add on packages to help protect against content interception: RTMPE and RTMPTE, the AllowDomains setting in Application.xml, the "Media Security" package (which includes SecureToken and SecureURLParams), the "Stream Name Alias" package and the onConnect method handler. The next section describes each of these methods of content protection.

RTMPE and RTMPTE

RTMPE and RTMPTE are encrypted versions of the RTMP and RTMPT protocols. These protocols provide 128-bit encrypted channel between the Flash player and the Wowza Pro server. Encryption keys are per-session. RTMPE and RTMPTE provide strong protection against media ripping software.

AllowDomains

The "Connections/AllowDomains" setting in the Application.xml file is a comma delimited list of domain names or ip address for which client connections will be accepted. The domain names or ip addresses that are specified here represent the domain name or ip address of the Flash swf file that is connecting to the Wowza Pro server or the ip address of the client connecting to Wowza Pro. See the "Application Configuration" section of the "Server Administration" chapter for more details.

This setting will refuse server connections from Flash players that are not hosted in your domain or at a specific ip address. So if another website tries to link to content served by your Wowza Pro server, the connection will be refused and closed. This method is very straight forward to implement but only provides a medium level of security. This method does not block applications such as "Replay Media Catcher" which will act (or spoof) as if the request is being initiated from your website. The "SecureToken" example, discussed below, provides a higher level of security against the spoofing threat.

Note

To temporarily turn off AllowDomains for development of your Wowza Pro solution, add the following property definition to the JAVA_OPTS variable defined in [install-dir]/bin/setenv.sh (or setenv.bat on Windows): -Dcom.wowza.wms.AllowDomains.enable=false.

Media Security Package (SecureToken and SecureURLParams)

The "Media Security" package is an add on package that includes both the SecureToken and SecureURLParams security systems. This package can be downloaded from the following forum post:

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

Here is a brief description of each of these technologies. For more indepth information consult the documentation that accompanies the "Media Security" package.

SecureToken

SecureToken is a challenge and response based security system that provides a high level of content protection against spoofing threats like those posed by the "Replay Media Catcher". Each connection is protected by a random single use key and a password (shared secret). The basic security methodology is described below.

The way SecureToken works is that upon client connection the provided custom module generates a unique key for the pending connection. The generated key is encrypted using the TEA (Tiny Encryption Algorithm) algorithm using a password (shared secret) that is shared between the Wowza Pro server and your Flash client movie. The encrypted unique key is returned as the "secureToken" parameter of the object that is the first parameter to the event callback NetConnection.onStatus. The Flash client movie then decrypts the unique key using the shared password and sends the result back to the custom module by calling NetConnection.call("secureTokenResponse", null, decodedKey). The server then compares this key to the originally generated key. If they match then processing for that connection continues. If the values do not match then the connection is aborted. If the Flash client movie tries to create a NetStream object without first calling "secureTokenResponse" with the correctly decoded key, then the connection is aborted.

SecureURLParams

SecureURLParams is a simple method for providing password like protection to the connection process between the Flash player client and Wowza Pro. This security measure is generally used to protect the publishing process. In most cases it is used along side SecureToken. SecureURLParams is used to protect publishing and SecureToken is used to protect playback.

With SecureURLParams you define name value pair combinations that are assigned to protect one of three server functions: connect, play or publish. The name value pairs are passed to Wowza Pro as part of the rtmp connection url (works with all variants of rtmp such as rtmpe).

Stream Name Alias Package

The "Stream Name Alias" package is a simple and powerful system for masking and protecting complex URL based stream names. It can also be used to control which stream names are accepted by the server side implementation of NetStream.play and NetStream.publish. This package can be downloaded from the following forum post:

http://www.wowzamedia.com/forums/showthread.php?t=1505

onConnect Method Handler

The third method for securing content is to devise your own custom security system using the Wowza Media Server Pro server side API. Using the Java API you can easily extend the server to provide your own custom authentication system to protect your content. The following two chapters describe in detail how the custom module system works. The onConnect method is an event handler that is called each time a client attempts to make a connection to the Wowza Pro server. This is a great place to integrate a secure authentication system. The Java environment also provides the built in services and classes to do database authentication and cryptography.

Note

For more up to date security information or examples of how to secure other open source Flash Media players such as the JW Media Player, see the "Useful Code" section of the Wowza Media Systems Forums at http://www.wowzamedia.com/forums/.

AddOn Packages

Wowza Media Server Pro provides a very open and dynamic server side API that is covered in detail in the next chapter. Many Wowza Pro features, such as several of the security features discussed above, are provided as "AddOn Packages". To see a list of "AddOn Packages", visit the following page on the Wowza Pro web site:

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

This page will be updated as "AddOn Packages" are made available or are updated.

Chapter

6

Server Side Modules

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

Server side functionality in Wowza Media Server Pro is encapsulated in a set of modules. At runtime these modules are dynamically bound to a given application based on the configuration information specified in an application's Application.xml file. Wowza Media Server Pro ships with five server side modules: ModuleCore, ModuleProperties, ModuleClientLogging, ModuleFastPlay and ModuleFLVPlayback. This chapter provides a brief introduction to modules and module configuration. It also describes in detail each of the five modules that ship with the server. The next chapter "Creating a Custom Module" will discuss how to extend the server's functionality by creating your own custom server side modules.

Server Side Module Defined

A server side module in Wowza Media Server Pro is a Java Archive (jar) file that is dynamically linked into the server at runtime. The jar file can contain multiple classes and resources needed to implement a set of functionality. The four included modules are linked into the wms-server.jar file that is loaded at application startup. A module is added to an application by adding an entry to the Modules section of the application's Application.xml file. Below is an example of an application that loads the ModuleCore module.

```
<Root>
      <Application>
            <Connections>
                  <AutoAccept>true</AutoAccept>
            </Connections>
            <Streams>
                  <StreamType>file</StreamType>
                  <StorageDir></StorageDir>
            </Streams>
            <SharedObjects>
                  <StorageDir></StorageDir>
            </SharedObjects>
            <Modules>
                  <Module>
                        <Name>core</Name>
                        <Description>Core Module</Description>
                        <Class>com.wowza.wms.module.ModuleCore</Class>
                  </Module>
            </Modules>
      </Application>
</Root>
```

Included Modules

Wowza Media Server Pro ships with four modules. Each module is described below:

ModuleCore – (com.wowza.module.ModuleCore)

The ModuleCore module represents the server side implementation of the 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:

```
setStreamType(streamType:String);
getStreamType();
```

Returns and sets the default stream type for this client connection.

```
getClientID();
```

Returns the client ID for this client connection.

```
getVersion();
```

Returns the server name and version.

```
getLastStreamId();
```

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

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.

```
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 Pro 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.

Chapter

7

Creating a Custom Module

What is a Wowza Media Server Pro Custom Module?

This chapter describes in detail how custom server side modules work. A server side module is a Java Archive (jar) file that encapsulates a set of functionality that is dynamically linked into the server at runtime. All modules must be compiled and built using a Java Development Kit (JDK) version 5 (aka 1.5) or greater. Wowza Media Systems provides an Eclipse based integrated development environment called the Wowza IDE that automates much of the process of creating a server side module.

Getting Started

The first tool you need for Java development is a Java Development Kit (JDK) version 5 (aka 1.5) or greater. This kit can be obtained from the Sun website at http://java.sun.com/javase/downloads. Next, you will need the Wowza IDE. The Wowza IDE can be downloaded from the Wowza Labs page at http://www.wowzamedia.com/labs.html. The "Creating a Wowza Media Server Pro Module" chapter of the Wowza IDE: User's Guide is a great starting point for learning how to create your first server side module.

Note

Before you read the rest of this chapter it might be a good idea to browse Wowza Media Server Pro javadocs to get a feel for the server side API. These Java docs can be opened from the "Start" menu by selecting "Wowza Media Server Pro>Documentation>Server Side API".

Note

Debug logging is useful during module development to debug module and method loading issues. To turn on debug logging, edit your conf/log4j.properties file and change the log level (in the first configuration property) from "INFO" to "DEBUG".

Module Basics

A server side module is Java Archive or jar file that contains a set of classes. These classes are dynamically loaded at runtime based on settings in Wowza Media Server Pro's configuration files. A module (or jar file) is made available to the server by placing it in the "[install-dir]/lib" directory of the Wowza Pro server.

A single module or jar file can expose multiple module classes. A module class is a public class that extends the com.wowza.wms.module.ModuleBase abstract class. A new module class instance is created for each application instance that is started by the server. So if you define a module for the application "myapplication" and two clients connect to the server with the following connection strings:

```
rtmp://localhost/myapplication/instance1
rtmp://localhost/myapplication/instance2
```

Two instances of your module class will be instantiated. One for each application instance. All class level properties will be unique to each application instance.

The public methods defined for a module class define the classes interface to the player client. There are three types of public methods that are callable by the server and/or the Flash player client: event methods (onApp, onConnect and onStream), custom methods and the onCall method. Each is described below:

Event Methods (onApp, onConnect and onStream)

Event methods are invoked by the server based on events that occur during server processing. There are three sets of event methods; onApp, onConnection and onStream. Each of these sets are represented by an interface class in the server side API; IModuleOnApp, IModuleOnConnect and IModuleOnStream. 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 bottom of the <Modules> list defined in Application.xml. So the last <Modules> entry in the list will be called first and it will work its way up to the first item in the list.

IModuleOnApp

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

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

IModuleOnConnect

```
public abstract void onConnect(IClient client,
                        RequestFunction function, AMFDataList params);
public abstract void onDisconnect(IClient client);
public abstract void onConnectAccept(IClient client);
public abstract 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 abstract void onStreamCreate(IMediaStream stream);
public abstract 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.

To implement one of these interfaces, you can either use the Java "implements" keyword followed by a comma delimited list of interfaces that you wish to implement or you can simply use the interface definitions as a reference for your method signatures and define them in your class as public methods. Below is a sample class that implements the IModuleOnApp methods.

```
package com.mycompany.module;

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

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

      public void onAppStop(IApplicationInstance appInstance)
      {
            getLogger().info("onAppStop");
      }
}
```

Note

Use the IModuleOnConnect interface to authenticate client connections. To setup authentication edit your application's Application.xml file and set Connections/AutoAccept to false. Next, in your applications server side module implement the IModuleOnConnect interface. Finally, in the onConnect event method you can validate the client connection by inspecting any extra parameters sent to the server from the client as part of the call to NetConnection.connect(). To accept/reject the client connection call client. acceptConnection() or client.rejectConnection().

Custom Methods

Custom methods are server side Java methods that you want to expose to the Flash player client through calls to the NetConnection.call() or are call 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 client.

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 can call "this.invokePrevious(client, function, params)". Wowza Pro will search up the module list and find the next module that implements the "play" method and it will invoke that method in that module. This provides a means to create an adhoc stack of methods. 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 any 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);
	}
}
```

Your modules are going to use a combination of the method types described above. When you are creating custom modules, you might want to group them according to functionality so they can be shared by your applications.

Custom Method Parameters

Parameters passed from the Flash player client to Wowza Media Server Pro 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);
      }
}
```

Returning Results from a Custom Method

A custom method 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 client. 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);
	}
}
```

Module 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 can be 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);
```

Server To Client Calls

A custom server side method can call a function in Flash player client 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 client side code:

```
var nc:NetConnection;

nc = new NetConnection();
nc.serverToClientMethod = function(param1, param2)
{
      return "Hello World";
}
nc.connect("rtmp://wms.mycompany.com/mymodules");
```

To call this method from the server, the custom method would look 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);
      }
}
```

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:

```
WowzaMediaServerPro/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 Pro 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.

Other Server Extension Options

There are several other ways of using custom modules to extend the functionality of Wowza Media Server Pro. This section will cover two of these extension points; ServerListeners and HTTPProviders.

ServerListeners

A ServerListener is a class that gets invoked during the initialization process of the Wowza Pro server. It is notified of events during the life cycle of the server. This extension point can be used to startup custom functionality that is going to co-exist along side the Wowza Pro server. This mechanism might be used to auto-start a servlet container that will handle http requests or a SOAP server for providing a custom SOAP interface to integrate with a Windows .NET application or non-Java application.

A ServerListener must implement the com.wowza.wms.server.IServerNotify2 interface. Configuration for this extension point is done in the Server.xml file by adding an <ServerListener> entry to the <ServerListeners> list and setting the <BaseClass> value to the full path to your ServerListener class. Below is a simple ServerListener class:

```
package com.mycompany.module;

import com.wowza.wms.application.*;
import com.wowza.wms.logging.*;
import com.wowza.wms.server.*;

public class TestServerListener implements IServerNotify2
{
	public void onServerConfigLoaded(IServer server)
	{
		WMSLoggerFactory.getLogger(Application.class).debug("onServerConfigLoaded ");
	}

	public void onServerCreate(IServer server)
	{
		WMSLoggerFactory.getLogger(Application.class).debug("serverCreate");
	}

	public void onServerInit(IServer server)
	{
		WMSLoggerFactory.getLogger(Application.class).debug("serverInit");
	}

	public void onServerShutdownComplete(IServer server)
	{
		WMSLoggerFactory.getLogger(Application.class).debug("serverShutdownComplete");
	}

	public void onServerShutdownStart(IServer server)
	{
		WMSLoggerFactory.getLogger(Application.class).debug("serverShutdownStart");
	}

}
```

To add this ServerListener to Server.xml the XML snippet looks like this:

```
<ServerListeners>
	<ServerListener>
		<BaseClass>com.mycompany.module.TestServerListener</BaseClass>
	</ServerListener>
</ServerListeners>
```

HTTPProviders

An HTTPProvider is a class bound to a HostPort definition in the VHost.xml configuration file. This class is invoked when http requests are made to the serer over the port defined by the HostPort. The API for this extension point is very similar to the HTTPServlet API. This extension point can be used to provide an HTML interface into the Wowza Pro server. This mechanism might be used to create a custom administration interface to the server or provide query parameter output consumable by the Flash player LoadVars mechanism to provide a custom load balancing solution.

An HTTPProvider must implement the com.wowza.wms.http.IHTTPProvider interface. Configuraiton for this extension point is done in the VHost.xml file. Below is a simple HTTPProvider class:

```
import java.io.OutputStream;

import com.wowza.wms.application.WMSProperties;
import com.wowza.wms.bootstrap.*;
import com.wowza.wms.stream.*;
import com.wowza.wms.vhost.*;
import com.wowza.wms.logging.*;

public class HTTPServerVersion implements IHTTPProvider
{
        private WMSProperties properties = null;

        public void onBind(IVHost vhost, HostPort hostPort)
        {
        }

        public void onUnbind(IVHost vhost, HostPort hostPort)
        {
        }

        public void setProperties(WMSProperties properties)
        {
                this.properties = properties;
        }

        public void onHTTPRequest(IVHost vhost, IHTTPRequest req, IHTTPResponse resp)
        {
                String testStr = "HelloWorld;
                String retStr = "<html><head><title>"+testStr+
                        "</title></head><body>"+ testStr+"</body></html>";
                try
                {
                        OutputStream out = resp.getOutputStream();
                        byte[] outBytes = retStr.getBytes();
                        out.write(outBytes);
                }
                catch (Exception e)
                {
                        WMSLoggerFactory.getLogger(HTTPServerVersion.class).error(
                                "HTMLServerVersion: "+e.toString());
                }
        }
}
```

Chapter

8

Virtual Hosting

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

Wowza Media Server Pro 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
                        MediaCasters.xml
                        MediaReaders.xml
                        MediaWriters.xml
                        MP3Tags.xml
                        RTP.xml
                        Streams.xml
                        VHost.xml
                        rtp.password
                  [content]
                  [logs]
            [vhost2]
                  [applications]
                  [conf]
                        Application.xml
                        Authentication.xml
                        MediaCasters.xml
                        MediaReaders.xml
                        MediaWriters.xml
                        MP3Tags.xml
                        RTP.xml
                        Streams.xml
                        VHost.xml
                        rtp.password
                  [content]
                  [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 Media Server Pro 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 "WowzaMediaServerPro" 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

9

Examples & AddOn Packages

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

Wowza Media Server Pro ships with many examples that highlight the functionality of the server. This chapter describes each of these examples and provides. All examples are implemented using ActionScript 3.0. For most examples, there is also an ActionScript 2.0 implementation provided in the "clientAS2" folder that is inside each of the example folders.

Wowza Media Systems also provide several AddOn Packages that extend and enhance the functionality of Wowza Pro. 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 illustrates how to implement a custom video player that streams static Flash video (.flv) content from the server to the client. It utilizes the "file" stream type.

FastPlayVideoStreaming

This example illustrates how to use the ModuleFastPlay module. Fast play is a TiVo® style fast forward/fast rewind streaming playback mechanism.

LiveVideoStreaming

This example illustrates how to setup and playback live video. It utilizes the "live" stream type.

NativeRTPVideoStreaming

This example illustrates how to setup and playback live video from a native RTP source. It utilizes the "rtp-live" stream type.

VideoChat

This example 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 example 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 example illustrates how to implement a simple text chat application.

SHOUTcast

This example illustrates how re-stream SHOUTcast MP3 or AAC+ audio data through the Wowza Pro server. It utilizes the "shoutcast" stream type.

RemoteSharedObjects

This example 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 SecureURLParams functionality as well as a document that covers other methods of securing Wowza Pro. To obtain the latest version of this package visit the following Wowza Pro forum thread:

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

BWChecker

This example provide a means for testing the bandwidth between individual Flash client connections and he Wowza Pro server. 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 Pro. To obtain the latest version of this package visit the following Wowza Pro forum thread:

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

RTMPSConnectionModule

This example illustrates how to create a module that can accept RTMPS connections.