Wowza Media Server® 3

Flash Media Server to Wowza Media Server 3 API Mapping

Wowza Media Server 3: Flash Media Server to Wowza Media Server 3 API Mapping

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

Third Party Information

This document contains links to third party websites that are not under the control of Wowza Media Systems, LLC ("Wowza") and Wowza is not responsible for the content on any linked site. If you access a third party website mentioned in this document, then you do so at your own risk. Wowza provides these links only as a convenience, and the inclusion of any link does not imply that Wowza endorses or accepts any responsibility for the content on third party sites.
This document also refers to other third party software that is not licensed, sold, distributed or otherwise endorsed by Wowza. Please ensure that any and all use of Wowza® software and third party software is properly licensed.

Trademarks

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

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

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

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

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

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

Third Party Copyright Notices

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

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

Silver Egg Technology: Copyright © 2001, Silver Egg Technology

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

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

Apache Commons Lang libraries and Modeler libraries: Copyright © 2001-2008, The Apache Software Foundation

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

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

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

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

Table of Contents

Introduction

The goal of this document is to provide a mapping between the Flash Media Server server-side API and the Wowza Media Server 3 server-side API. This document is by no means a complete reference for the Wowza Media Server 3 server-side API. Please consult the Wowza Server Javadocs that are part of the server install for a complete reference.

All Java objects in the Wowza Media Server 3 are organized into packages. Some of these packages are provided by the Java 6 runtime environment. The table below is a listing of all the objects referenced in this document along with their corresponding java package. These package names are imported into your custom modules using the "import" statement. An import statement can either reference a single class using the syntax:

```
import com.wowza.wms.application.IApplication;
```

Or it can import an entire package using the syntax:

```
import com.wowza.wms.applicaton.*;
```

Java Objects and Packages

Object	Package
File	java.io
HashMap	java.util
IApplication	com.wowza.wms.application
IApplicationInstance	com.wowza.wms.application
IClient	com.wowza.wms.client
IMediaStream	com.wowza.wms.stream
IServer	com.wowza.wms.server
ISharedObject	com.wowza.wms.sharedobject
ISharedObjects	com.wowza.wms.sharedobject
ISharedObjectSlotNotify	com.wowza.wms.sharedobject
Iterator	java.util
IVHost	com.wowza.wms.vhost
List	java.util
Map	java.util
Stream	com.wowza.wms.stream.publish
WMSLogger	com.wowza.wms.logging
WMSLoggerFactory	com.wowza.wms.logging

Flash Media Server to Wowza Media Server 3 API Mapping

The remainder of this document is a set of mapping tables mapping each of the API object, methods, properties and event handlers in Flash Media Server to the equivalent call in the Wowza Media Server 3 API. In many cases the mapping in not one to one or the functionality is provided by Java packages that are part of the Java 6 runtime environment. In these cases a snippet of Java code is provided to show how to use the Wowza Media Server 3 API and Java packages to perform an equivalent or similar function.

<table>
<tr><th colspan="2">Global Functions</th></tr>
<tr><td>Functions</td><td></td></tr>
<tr><td>setInterval
clearInterval</td><td>Note: This code snippet is similar to the functionality provided by setInterval and clearInterval. This class that implements the Runnable interface. This class is being submitted to the virtual host's thread pool for execution. When a thread is available the run method of this object will be invoked. Put the code you wish to execute below the "// do action here" comment. The "Thread.currentThread().sleep(1000)" statement will put the thread to sleep for 1000 miliseconds before is executes the action again. As written this object will remain running until the done boolean is set to true. While running (the run method is executed), this object will use a single thread in the thread pool.

<pre>
import com.wowza.wms.client.*;
import com.wowza.wms.vhost.*;

{
   class MyTimer implements Runnable
   {
      private boolean done = false;

      public void run()
      {
         while (true)
         {
            // do action here and set done to true to exit

            if (done)
               break;

            try
            {
               // interval in milliseconds
               Thread.currentThread().sleep(1000);
            }
            catch (Exception e)
            {
            }
         }
      }
   }

   client.getVHost().getThreadPool().execute(new MyTimer());
}
</pre>
</td></tr>
<tr><td>getGlobal</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
<tr><td>load</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
<tr><td>protectObject</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
<tr><td>setAttributes</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
</table>

Global Functions	

trace	`getLogger().debug(String logStr);`
	`getLogger().info(String logStr);`
	`getLogger().warn(String logStr);`
	`getLogger().error(String logStr);`
	`getLogger().fatal(String logStr);`
	`-- or --`
	`WMSLoggingFactory.getLogger(null).debug(String logStr);`
	`WMSLoggingFactory.getLogger(null).info(String logStr);`
	`WMSLoggingFactory.getLogger(null).warn(String logStr);`
	`WMSLoggingFactory.getLogger(null).error(String logStr);`
	`WMSLoggingFactory.getLogger(null).fatal(String logStr);`

Application Object	
Methods	
acceptConnection	`IClient.acceptConnection();`
broadcastMsg	`IApplicationInstance.broadcastMsg(String handlerName, Object .. params);`

clearSharedObjects	Note: This code snippet shows show to obtain the path to the remoted shared object files. You can then use simple java File calls to loop through the directory and delete files as needed.
	
	`import java.io.*;`
	`import com.wowza.wms.client.*;`
	`import com.wowza.wms.sharedobject.*;`
	
	`{`
	`  String rsoStoragePath =`
	`  client.getAppInstance().getRsoStoragePath();`
	`  File rsoDir = new File(rsoStoragePath);`
	`  String[] fileList = rsoDir.list();`
	`  for(int i=0;i<fileList.length;i++)`
	`  {`
	`  File file = new File(rsoStoragePath +`
	`  File.separatorChar + fileList[i]);`
	`  file.delete();`
	`  }`
	`}`

clearStreams	Note: This code snippet shows show to obtain the path to the stream files. You can then use simple java File calls to loop through the directory and delete files as needed.
	
	`import java.io.*;`
	`import com.wowza.wms.client.*;`
	`import com.wowza.wms.sharedobject.*;`
	
	`{`
	`  String streamStoragePath =`
	`  client.getAppInstance().getStreamStoragePath();`
	`  File rsoDir = new File(streamStoragePath);`
	`  String[] fileList = rsoDir.list();`
	`  for(int i=0;i<fileList.length;i++)`
	`  {`
	`  File file = new File(streamStoragePath +`
	`  File.separatorChar + fileList[i]);`
	`  file.delete();`
	`  }`
	`}`

disconnect	`IClient.setShutdownClient(true);`

gc	`{`
	`  System.gc();`
	`}`

getStats	`IApplicationInstance.getIOPerformanceCounter();` `IApplicationInstance.getConnectionCounter();`
registerClass	(no equivalent in Wowza Media Server 3)
registerProxy	(no equivalent in Wowza Media Server 3)
rejectConnection	`IClient.rejectConnection(String reason);`
shutdown	`IApplication.removeAppInstance(IApplicationInstance appInstance);`

<table>
<tr><th colspan="2">Application Object</th></tr>
<tr><td>Properties</td><td></td></tr>
<tr><td>allowDebug</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
<tr><td>clients</td><td><pre>IApplicationInstance.getClientCount();
IApplicationInstance.getClient(int index);
IVHost.getClient(int clientId);</pre></td></tr>
<tr><td>config</td><td>Note: Most of the configuration options for the system are accessible through both the Java API (see the Wowza Media Server 3 API javadocs) and the JMX interface.</td></tr>
<tr><td>hostname</td><td><pre>IVHost.getHostPortsList();</pre></td></tr>
<tr><td>name</td><td><pre>IApplicationInstance.getName();</pre></td></tr>
<tr><td>server</td><td><pre>IServer.getInstance().getVersion();</pre></td></tr>
<tr><td>Event Handlers</td><td></td></tr>
<tr><td>onAppStart
onAppStop
onConnect
onConnectAccept
onConnectReject
onDisconnect</td><td>Note: This class is a basic custom module that implements the on[Event] handlers to receive notification of application level events.

<pre>package com.mycompany.module;

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

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

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

   public void onConnect(IClient client,
                  RequestFunction function, AMFDataList params)
   {
      // here is where you can accept/reject a connection request
      // client.acceptConnection();
      // client.rejectConnection("message as to why rejected");

      getLogger().info("onConnect");
   }

   public void onConnectAccept(IClient client)
   {
      getLogger().info("onConnectAccept");
   }

   public void onConnectReject(IClient client)
   {
      getLogger().info("onConnectReject");
   }

   public void onDisconnect(IClient client)
   {
      getLogger().info("onDisconnect");
   }
}</pre></td></tr>
<tr><td>onStatus</td><td>(no equivalent in Wowza Media Server 3)</td></tr>
</table>

Client Object	
Methods	
call	`IClient.call(String handlerName, IModuleCallResult resultObj, Object ... params);`
getBandwidthLimit	(no equivalent in Wowza Media Server 3)

getStatus	`IClient.getIOPerformanceCounter();`
	`IClient.getConnectionCounter();`
	`IClient.getPingRoundTripTime();`

ping	`IClient.ping(IModulePingResult pingResult);`
__resolve	(no equivalent in Wowza Media Server 3)
setBandwidthLimit	(no equivalent in Wowza Media Server 3)
Properties	
agent	`IClient.getFlashVer();`
ip	`IClient.getIp();`
protocol	`IClient.getProtocol(); [1=RTMP, 3=RTMPT]`
readAccess	Note: In Wowza Server read access is treated separately for streams and shared objects. By default sharedObjectReadAccess and streamReadAccess are set to READ_ACCESS_ALL. To deny read access to a client, set read access to READ_ACCESS_NONE. All read access values are case sensitive on all platforms. Set read access to a semi-colon delimited list of stream or shared object partial or full names. See documentation for IClient.setSharedObjectReadAccess and IClient.setStreamReadAccess for details. IClient.READ_ACCESS_ALL IClient.READ_ACCESS_NONE IClient.getSharedObjectReadAccess(); IClient.setSharedObjectReadAccess(String sharedObjectReadAccess); IClient.getStreamReadAccess(); IClient.setStreamReadAccess(String streamReadAccess);
referrer	`IClient.getReferrer();`
secure	`IClient.isSecure();`
uri	`IClient.getUri();`
virtualKey	(no equivalent in Wowza Media Server 3)
writeAccess	Note: In Wowza Server write access is treated separately for streams and shared objects. By default sharedObjectWriteAccess and streamWriteAccess are set to WRITE_ACCESS_ALL. To deny write access to a client, set write access to WRITE_ACCESS_NONE. All write access values are case sensitive on all platforms. Set write access to a semi-colon delimited list of stream or shared object partial or full names. See documentation for IClient.setSharedObjectWriteAccess and IClient.setStreamWriteAccess for details. IClient.WRITE_ACCESS_ALL IClient.WRITE_ACCESS_NONE IClient.getSharedObjectWriteAccess(); IClient.setSharedObjectWriteAccess(String sharedObjectWriteAccess); IClient.getStreamWriteAccess(); IClient.setStreamWriteAccess(String streamWriteAccess);

Event Handlers	

[command name]	Note: This class is a basic custom module that implements a single custom command/handler named myFunction. This handler can be invoke from the client by calling NetConnection.call("myFunction", null);.
	
	`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)`
	`  {`
	`  getLogger().info("myFunction");`
	`  }`
	`}`

File Object	
Methods (text files)	
open close read readln readAll write writeln writeall eof flush	Note: This code snippet illustrates a method for reading and writing text files using the java.io package that is included in the Java 5 runtime environment. For more information consult the Java 5 javadocs (http://java.sun.com/j2se/1.5.0/docs/api/).

```
package com.mycompany.module;

import java.io.*;

import com.wowza.wms.module.*;

public class MyModule extends ModuleBase
{
   private void readTextFile(String fileName)
   {
      try
      {
         BufferedReader inf = null;
         String line = null;

         // read file line by line
         inf = new BufferedReader(new FileReader(fileName));
         while((line = inf.readLine()) != null)
         {
            // do something with each line of the file
         }
         inf.close();
      }
      catch (Exception e)
      {
         getLogger().error("Error reading text file: "+
                     fileName+" :"+e.toString());
      }
   }

   private void writeTextFile(String fileName, boolean isAppend)
   {
      try
      {
         PrintWriter outf = new PrintWriter(
               new FileWriter(fileName, isAppend));

         // write String
         outf.print("Hello World");

         // write String with carriage return
         outf.println("Hello World");

         // flush data to file
         outf.flush();

         outf.close();
      }
      catch (Exception e)
      {
         getLogger().error("Error writing text file: "+
                     fileName+" :"+e.toString());
      }
   }
}
```

Methods (binary files)	
open close readBytes writeBytes seek position (property) eof flush	Note: This code snippet illustrates a method for reading and writing binary files using the java.io package that is included in the Java 5 runtime environment. For more information consult the Java 5 javadocs (http://java.sun.com/j2se/1.5.0/docs/api/).

```
package com.mycompany.module;

import java.io.*;

import com.wowza.wms.module.*;

public class MyModule extends ModuleBase
{
   private void readBinaryFile(String fileName)
   {
      try
      {
         RandomAccessFile inf = new RandomAccessFile(fileName, "r");

         int bufflen = 512;
         byte[] buffer = new byte[bufflen];

         // read [bufflen] bytes from the file
         int bytesRead = inf.read(buffer, 0, bufflen);
         if (bytesRead != bufflen)
            getLogger().info("hit end of file");

         // seek to the 1024 byte in the file
         inf.seek(1024);

         // get the current position in the file
         long currPos = inf.getFilePointer();

         inf.close();
      }
      catch (Exception e)
      {
         getLogger().error("Error reading binary file: "+
                     fileName+" :"+e.toString());
      }
   }

   private void writeBinaryFile(String fileName, boolean isAppend)
   {
      try
      {
         int bufflen = 512;
         byte[] buffer = new byte[bufflen];

         // open a binary file to append to a file or over write
         PrintStream outf = new PrintStream(new
                  FileOutputStream(fileName, isAppend));

         // write bytes to the file
         outf.write(buffer, 0, bufflen);

         outf.close();
      }
      catch (Exception e)
      {
         getLogger().error("Error writing binary file: "+
                     fileName+" :"+e.toString());
      }
   }

   private void writeReadBinaryFile(String fileName, boolean isAppend)
   {
      try
```

	{
	int bufflen = 512;
	byte[] buffer = new byte[bufflen];
	
	// open a binary file to read/write a file
	RandomAccessFile outfRand = new RandomAccessFile(
	fileName, "rw");
	
	// seek to the 1024 byte in the file
	outfRand.seek(1024);
	
	// get the current position in the file
	long currPos = outfRand.getFilePointer();
	
	// write bytes to the file
	// if you write past the end of the file
	// it will extend the file
	outfRand.write(buffer, 0, bufflen);
	
	outfRand.close();
	}
	catch (Exception e)
	{
	getLogger().error("Error writing binary file: "+
	fileName+" :"+e.toString());
	}
	}
	}

Methods	
copyTo	FileUtils.copyFile(new File(fromFilename), new File(toFilename));

list	import java.io.*;
	
	{
	File folder = new File(String foldername);
	folder.list();
	folder.list(FilenameFilter filter);
	}

mkdir	import java.io.*;
	
	{
	File folder = new File(String foldername);
	folder.mkdir();
	folder.mkdirs();
	}

remove	import java.io.*;
	
	{
	File file = new File(String filename);
	file.delete();
	file.deleteOnExit();
	}

renameTo	import java.io.*;
	
	{
	File file = new File(String filename);
	file.renameTo(new File(toFilename));
	}

toString	import java.io.*;
	
	{
	File file = new File(String filename);
	file.toString();
	}

Properties	

canAppend	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.canWrite();`
	`}`

canRead	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.canRead();`
	`}`

canReplace	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.canWrite();`
	`}`

canWrite	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.canWrite();`
	`}`

creationTime	(no equivalent in Wowza Media Server 3)

exists	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.exists();`
	`}`

isDirectory	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.isDirectory();`
	`}`

isFile	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.isFile();`
	`}`

lastModified	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.lastModified();`
	`}`

length	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.length();`
	`}`

mode	(no equivalent in Wowza Media Server 3)

name	`import java.io.*;`
	
	`{`
	`  File file = new File(String filename);`
	`  file.getName();`
	`}`

type	(no equivalent in Wowza Media Server 3)

LoadVars Object	
Description	
	In Flash Media Server the LoadVars object is a means of loading external data from a url into a container class. Similar functionality can be achieved with the code snippet below.
Example	

```
package com.mycompany.util;

import java.io.*;
import java.util.*;
import java.net.*;

public class LoadVarsUtil
{
   static public Map loadVars(String inUrl)
   {
   URL url;
   HttpURLConnection urlConn;
   DataInputStream input = null;

   Map ret = new HashMap();

   try
   {
      url = new URL (inUrl);
      urlConn = (HttpURLConnection)url.openConnection();
      urlConn.setRequestProperty("Content-Type", "text/html");
      urlConn.setRequestProperty("User-Agent",
         "Mozilla/4.0 (Windows XP 5.1) Java/1.5");
      urlConn.setRequestMethod("GET");

      if (urlConn.getResponseCode() == HttpURLConnection.HTTP_OK )
      {
         input = new DataInputStream (urlConn.getInputStream());

            int rChunk = urlConn.getContentLength();
            byte[] myData = new byte[rChunk];

            int bytesRead = input.read(myData, 0, rChunk);
            if (bytesRead == rChunk)
            {
               String resStr = new String(myData, "UTF8");
               if (resStr != null)
               {
                  String[] valuePairs = resStr.split("[&]");
                  for(int i=0;i<valuePairs.length;i++)
                  {
                     String valuePair = valuePairs[i];
                     int pos = valuePair.indexOf("=");
                     if (pos >= 0)
                     {
                        String name = valuePair.substring(0, pos);
                        String value = valuePair.substring(pos+1);

                        try
                        {
                           ret.put(name, value);
                        }
                        catch (Exception e)
                        {

                        }
                     }
                  }
               }
            }
```

LoadVars Object	

	input.close();
	input = null;
	}
	
	}
	catch (Exception e)
	{
	System.out.println("doHTTPPost: "+e.toString());
	e.printStackTrace();
	}
	
	return ret;
	}
	}

Log Object	
Methods	

onLog	getLogger().debug(String logStr);
	getLogger().info(String logStr);
	getLogger().warn(String logStr);
	getLogger().error(String logStr);
	getLogger().fatal(String logStr);
	-- or --
	WMSLoggingFactory.getLogger(null).debug(String logStr);
	WMSLoggingFactory.getLogger(null).info(String logStr);
	WMSLoggingFactory.getLogger(null).warn(String logStr);
	WMSLoggingFactory.getLogger(null).error(String logStr);
	WMSLoggingFactory.getLogger(null).fatal(String logStr);

NetConnection Object
(server to server NetConnections are not supported in Wowza Media Server 3 – we are considering this feature for a future release.)

SharedObject	
Constructor	

	Note: Below is a snippet of code to illustrate how to create a new shared object server side.
	
	import com.wowza.sharedobject.*;
	
	{
	String soName = "mySharedObject";
	boolean isPersistent = false;
	
	ISharedObjects sharedObjects =
	client.getAppInstance().getSharedObjects(isPersistent);
	ISharedObject sharedObject = sharedObjects.getOrCreate(soName);
	}

Methods	
clear	ISharedObject.clear();

close	Note: To immediately remove a reference to a shared object set it's variable to null.
	
	import com.wowza.sharedobject.*;
	
	{
	String soName = "mySharedObject";
	boolean isPersistent = false;
	
	ISharedObjects sharedObjects =
	client.getAppInstance().getSharedObjects(isPersistent);
	ISharedObject sharedObject = sharedObjects.get(soName);
	
	// to immediately remove the reference set the local variable to null
	sharedObject = null;

SharedObject	
	}

commit	import com.wowza.sharedobject.*;
	import java.util.*;
	
	{
	  ISharedObject sharedObjects =
	  client.getAppInstance().getSharedObjects(true);
	  List soNames = sharedObjects.getObjectNames();
	  Iterator iter = soNames.iterator();
	  while (iter.hasNext())
	  {
	  String soName = (String)iter.next();
	  ISharedObject so = sharedObjects.get(soName);
	  so.flush();
	  }
	}

flush	ISharedObject.flush();

get	import com.wowza.sharedobject.*;
	
	{
	  String soName = "mySharedObject";
	  boolean isPersistent = false;
	
	  ISharedObjects sharedObjects =
	  client.getAppInstance().getSharedObjects(isPersistent);
	  ISharedObject sharedObject = sharedObjects.get(soName);
	}

getProperty	ISharedObject.getProperty(String name);
getPropertyNames	ISharedObject.getSlotNames();
lock unlock	ISharedObject.lock(); ISharedObject.unlock();
mark	(no equivalent in Wowza Media Server 3)
purge	ISharedObject.purge(int version);
send	ISharedObject.send(String handlerName, Object ... params);
setProperty	ISharedObject.setProperty(String name, AMFData data);
size	ISharedObject.size();
Properties	
autoCommit	(no equivalent in Wowza Media Server 3)
isDirty	(no equivalent in Wowza Media Server 3)
name	ISharedObject.getName();
resyncDepth	(no equivalent in Wowza Media Server 3)
version	ISharedObject.getVersion();
Event Handlers	
[handler name]	(no equivalent in Wowza Media Server 3)
onStatus	(no equivalent in Wowza Media Server 3)

onSync	import com.wowza.sharedobject.*;
	
	{
	  class SOListener implements ISharedObjectSlotNotify
	  {
	  public void onSlotSetValue(ISharedObject sharedObject,
	  ISharedObjectSlot slot)
	  {
	  getLogger().info("onSlotSetValue");
	  }
	
	  public void onSlotDelete(ISharedObject sharedObject,
	  ISharedObjectSlot slot)
	  {
	  getLogger().info("onSlotDelete");
	  }
	  }

SharedObject	

	SOListener soListener = new SOListener();
	sharedObject.addSlotListener(soListener);
	}

SoapCall Object
(see WebService object)

SoapFault Object
(see WebService object)

Stream Object	
Methods	
clear	IMediaStream.clear();
flush	(no equivalent in Wowza Media Server 3)

get	Note: The Wowza Media Server 3 manages two distinct sets of streams for each client connection; play streams and publish stream. The snippet of code below illustrates how to iterate through each of these stream sets.
	
	import com.wowza.wms.client.*;
	import com.wowza.wms.stream.*;
	
	{
	// get all the publish stream attached to a client
	List publishStream = client.getPublishStreams();
	Iterator iterPublish = publishStream.iterator();
	while (iterPublish.hasNext())
	{
	IMediaStream stream = (IMediaStream)iterPublish.next();
	}
	
	// get all the play streams attached to a client
	List playStreams = client.getPlayStreams();
	Iterator iterPlay = playStreams.iterator();
	while (iterPlay.hasNext())
	{
	IMediaStream stream = (IMediaStream)iterPlay.next();
	}
	}

length	IMediaStream.length();
play	(see the com.wowza.wms.stream.publish.* package in Wowza Media Server 3 API documentation)
record	(no equivalent in Wowza Media Server 3)
send	IMediaStream.send(String handlerName, Object ... params);
setBufferTime	IMediaStream.setBufferTime(int bufferTime);
setVirtualPath	(no equivalent in Wowza Media Server 3)
size	IMediaStream.size();
Properties	
bufferTime	IMediaStream.getBufferTime();
name	IMediaStream.getName();
syncWrite	(no equivalent in Wowza Media Server 3)
Event Handlers	
onStatus	IMediaStream.registerOnStatus(IMediaStreamCallback callback); IMediaStream.unregisterOnStatus();
onPlayStatus	IMediaStream.registerOnPlayStatus(IMediaStreamCallback callback); IMediaStream.unregisterOnPlayStatus();
[command name]	IMediaStream.registerCallback(String handlerName, IMediaStreamCallback callback); IMediaStream.unregisterCallback(String handlerName);

WebService Object	
Description	
	In Flash Media Server the WebService object interface is a set of APIs for making Simple Object Access Protocol (SOAP) calls to a SOAP server. There are several Java open source API's to accomplish the same or similar functionality. One of those projects is Apache Foundation Axis. Below is a link to this project.
References	
	http://ws.apache.org/axis/

XML Object	
Description	
	In Flash Media Server the XML object interface is a set of APIs for manipulating XML documents. The Java 5 runtime environment contains a set of classes and interfaces that provide similar functionality. Below are a few references to tutorials on how to use the DOM, SAX, XPath and XSLT APIs in Java.
References	
	http://www.brics.dk/~amoeller/XML/programming/index.html
	http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/TOC.html
	Note: Reference the "javax.xml" and "org.w3c.dom" packages. http://java.sun.com/j2se/1.5.0/docs/api/

XMLSocket/XMLStreams Object	
Description	
	In Flash Media Server the XMLSocket and XMLStreams object interfaces are a set of APIs for communicating over XML to other Flash Media Servers or 3rd party severs. The Java 5 runtime environment contains a set of classes in "java.nio" package for streaming data over sockets and pipes that provides similar functionality. Below is a reference to this package in the Java 5 javadocs.
References	
	Note: Reference the "java.nio" package. http://java.sun.com/j2se/1.5.0/docs/api/