Red5 - Reference Documentation

Version 1.0

Steven Gong, Paul Gregoire, Daniel Rossi

Red5 is an Open Source Flash Server written in Java that supports:

- Streaming Audio/Video (FLV and MP3)
- Recording Client Streams (FLV only)
- Shared Objects
- Live Stream Publishing
- Remoting (AMF)

For the current release information goto the Releases [http://www.red5.org/red5-server/] page at red5.org.

Lastly the documentation is under development at the moment. If you have unanswered questions after using it, please check out our mailing list [http://groups.google.com/group/red5interest]

1.1. 0.8 Public Beta Release

More stable version of 0.7 for production environment testing. This public beta release will have many more features and so the focus should be on testing and bug reports.

- Hot Deployment
- Auto-unpacking of wars
- New Install application for example apps

** examples are downloaded and installed on demand

- No installed examples by default
- New Download Repository
- Official Release Home (red5.org)
- New Welcome page with new styles
- Bug fixes

Part I. Getting Started

Red5 intro here

- Chapter 2, *Frequently Asked Questions*
- Chapter 3, *Configuration Files*
- Chapter 4, *Migration Guide*
- Chapter 5, *Red5 Libraries*
- Chapter 6, *Building Red5*
- Chapter 8, *Releasing Red5*
- Chapter 9, *Recommended System Requirements*

The best way you can help make this FAQ more useful is by asking questions: either in any of the places above, or by leaving your questions in the comments below.

- Bugs and requests for new features can be submitted to Trac
- Ideas for new features can be talked about on the mailinglist

2.1. Questions

2.1.1. General

- **What is Red5?**
- **What does Red5 stand for?**
- **Is there a migration guide from FMS to Red5?**
- **applications? How do I create new applications?**
- **What are configuration files?**
- **Is there a mailing list?**
- What is the mailing list etiquette? (TODO)
- **What Ports does Red5 use?**
- **How can I help? I'm interested in helping the project. How can I help?**
- **Who is on the Red5 Team?**
- Are there any benchmarks? (TODO)

2.1.2. Documentation

- **Where is the official documentation?**
- **Can I get the documentation in PDF format?**
- **Where can I find the latest javadocs?**

2.1.3. Configuration

- **How to disable Socket policy checking for 443 (RTMPS and HTTPS)?**

2.1.4. Streaming

- **How do I stream to/from custom directories?**
- **How to detect the end of recording?**
- **How can I record RTMP streams from Red5?**
- **Does Red5 support multicast streaming?**

- **Can Red5 stream using UDP?**

2.1.5. Codecs

- **What Codecs does Red5 Support?**
- **What is RTMFP and when will it be available in Red5?**

2.1.6. Database

- **What databases are supported?**
- **Can I use Hibernate with Red5?**

2.1.7. Scripting

- **What scripting languages are available?**
- **Does Red5 support Actionscript 1?**
- **Does Red5 support Actionscript 3?**

2.1.8. Shared Objects

- **How do you setup a Remote SharedObject?**
- **How can I make a Remote SharedObject persistant on the server?**
- **What are remote SharedObject slots?**

2.1.9. Legal

- **Licence Information**
- **Is Red5 Legal?**
- **Codec Licenses** (TODO)
- **Third Party Licenses** (TODO)

2.1.10. Red5 WAR version

- **Is there any documentation on the Red5 war version?**

2.1.11. Misc

- **Is there an IRC channel?**
- **Are there forums?**
- **Are there any frameworks that I can start with?**
- **What is Paperworld3D?**
- **What is Jedai?**

- **Are there free tools** (TODO)
- **Are there development tools?**
- **there video tutorials Are there video tutorials** (TODO)
- **Are there any examples on the web?**
- **Is there professional support?**
- **Are there hosting solutions?**
- **What Red5 groups can I join?**

2.1.12. Troubleshooting

- **Why am I receiving "closing due to long handshake?**

2.2. Answers

2.2.1. General

2.2.1.1. What is Red5?

Red5 is an open source Flash RTMP server written in Java that supports:

- Streaming Audio/Video (FLV and MP3)
- Recording Client Streams (FLV only)
- Shared Objects
- Live Stream Publishing
- Remoting

2.2.1.2. What does Red5 stand for?

Originally referenced to Star Wars. Red5 was the "one who did the impossible".

2.2.1.3. Is there a migration guide from FMS to Red5?

Yes: Migration Guide

2.2.1.4. How do I create new applications?

Creating New Applications

2.2.1.5. What are configuration files?

see: Configuration Files In Red5

2.2.1.6. Is there a mailing list?

Check the mailing list page [http://trac.red5.org/wiki/MailingLists].

2.2.1.7. What Ports does Red5 use?

```
http.port=5080  // tomcat or jetty servlet container

   rtmp.port=1935  // traditional rtmp

   rtmpt.port=8088  // rtmp tunneled over http

   mrtmp.port=9035  // used with an edge/origin setup

   proxy.source_port=1936  // used to debug
```

These default ports can be changed in "RED5_HOME]\conf\red5.properties"

Additionally, most users only forward port 1935 and 5080

2.2.1.8. I'm interested in helping the project. How can I help?

You can create a new Trac ticket for any contributions you want to make, attach the files there or link it. Make sure you signup on the mailinglist [http://trac.red5.org/wiki/MailingLists] as well..

2.2.1.9. Who is on the Red5 Team?

The Red5 Project (red5 AT osflash.org)

2.2.1.9.1. Project Managers

- Chris Allen (mrchrisallen AT gmail.com)
- John Grden (johng AT acmewebworks.com)

2.2.1.9.2. Active Members

- Dominick Accattato (daccattato AT gmail.com)
- Steven Gong (steven.gong AT gmail.com)
- Paul Gregoire (mondain AT gmail.com)
- Thijs Triemstra (info AT collab.nl)
- Dan Rossi (electroteque AT gmail.com)
- Anton Lebedevich (mabrek AT gmail.com)

2.2.1.9.3. Inactive Members

- Luke Hubbard (luke AT codegent.com)
- Joachim Bauch (jojo AT struktur.de)
- Mick Herres (mickherres AT hotmail.com)

- Grant Davies (grant AT bluetube.com)
- Steven Elliott (steven.s.elliott AT gmail.com)
- Jokul Tian (tianxuefeng AT gmail.com)
- Michael Klishin (michael.s.klishin AT gmail.com)
- Martijn van Beek (martijn.vanbeek AT gmail.com)

2.2.2. Documentation

2.2.2.1. Where is the official documentation?

2.2.2.2. Can I get the documentation in PDF format?

\

2.2.2.3. Where can I find the latest javadocs?

http://api.red5.nl

2.2.3. Configuration

2.2.3.1. How to disable Socket policy checking for 443 (rtmps and https)?

You can change the port to something over 1024 like 8443 or comment out the RTMPS section.

2.2.4. Streaming

2.2.4.1. How do I stream to/from custom directories?

Customize Stream Paths

2.2.4.2. How to detect the end of recording ?

See the API docs [http://api.red5.nl/org/red5/server/api/stream/IStreamAwareScopeHandler.html].

2.2.4.3. How can I record RTMP streams from Red5?

See: http://ptrthomas.wordpress.com/2008/04/19/how-to-record-rtmp-flash-video-streams-using-red5

2.2.4.4. Does Red5 support multicast streaming?

It should be noted that multicasting support is not available in the Flash Player. For that reason, no media server can deliver a multi-casting solution to the Flash Player. In addition, many networks have multicasting turned off so it may not be realiable for other platforms either such as Windows Media Player. These solutions usually fall back to unicasting when clients cannot receive muliticasted media. In regards to Unicasting, Red5 already has this functionality. In addition, we have an edge-origin solution sometimes referred to as stream-reapeating.

2.2.4.5. Can Red5 stream using UDP?

No. Even though Java can stream using UDP, the Flash Player can not receive data sent using UDP.

2.2.5. Codecs

2.2.5.1. What Codecs does Red5 Support?

Video codecs:

- ScreenVideo
- On2 VP6
- Sorenson H.263
- H264

Audio codecs:

- ADPCM
- NellyMoser
- MP3
- Speex
- AAC

2.2.5.2. What is RTMFP and when will it be available in Red5?

RTMFP stands for "RTMFP (Real Time Media Flow Protocol". You can read more about it in the release notes. Just search the following page [http://labs.adobe.com/technologies/flashplayer10/releasenotes.html].

To understand what this protocol is and does, read the following FAQ [http://download.macromedia.com/pub/labs/flashplayer10/flashplayer10_rtmfp_faq_071708.pdf].

Red5 does not support RTMFP. At the moment, there isn't enough exposure to RTMFP and discussion can resume once it is released and more is known about the protocol.

2.2.6. Database

2.2.6.1. What databases are supported?

Red5 is built with Java. So any database that has a JDBC driver will work.

2.2.6.2. Can I use Hibernate with Red5?

2.2.7. Scripting

2.2.7.1. What scripting languages are available?

Scripting support (JavaScript, Groovy, Beanshell, JRuby, Jython)

2.2.7.2. Does Red5 support Actionscript 1?

Not yet, but there is development in this area and proof of concepts have been presented at conferences.

2.2.7.3. Does Red5 support Actionscript 3?

Not yet, but there is development in this area and proof of concepts have viewed by Red5 team members.

2.2.8. Shared Objects

2.2.8.1. How do you setup a Remote SharedObject?

see: http://livedocs.adobe.com/fms/2/docs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000607.html

2.2.9. Legal

2.2.9.1. Licence Information

http://www.opensource.org/licenses/lgpl-license.php

For an easier explanation, please see:

2.2.9.2. Is Red5 Legal?

Please read our response: http://osflash.org/red5/fud

2.2.10. Red5 WAR version

2.2.10.1. Is there any documentation on the Red5 war version?

read: Deploying To Tomcat

2.2.11. Misc

2.2.11.1. Is there an IRC channel?

Yes: #red5 on irc.freenode.net

Flash non-IRC based chat: http://red5.newviewnetworks.com/iChatBar2/#

2.2.11.2. Are there any examples on the web?

Below is a list of applications that use Red5.

- http://www.snappmx.com/ a Rapid Application Development System that supports the creation of Red5 applications.
- http://code.google.com/p/openmeetings by Sebastian Wagner.
- http://www.dokeos.com videconf module by Sebastian Wagner.
- http://spreed.com

- http://www.videokent.com/videochat.php
- http://www.weekee.tv an online video editing site by Weekee team.
- http://blipback.com BlipBack is a video comment widget that you can embed on any number of social network sites or blogs you]have. Blipback lets you or your friends record short video comments directly to your page.
- http://artemis.effectiveui.com Bridge AIR applications to the Java runtime.
- http://jooce.com Jooce is your very own, private online desktop - with public file sharing capabilities. A highly-secure, on]ine space to keep, view, listen to - and instantly share with friends - all your files, photos,music and video.
- http://facebook.com/video Video uploading/recording/messaging system that allows you to record a video on the upload page or]send a private message to another user and attach a video.
- http://www.f-ab.net F-ab is a simple browser for Flash movies. F-ab has "FLVPhone", which is a video conferencing telephone]sing the Flash movie. Red5 is embedded in F-ab to communicate with the remote FLVPhone.
- Streaming video chat software script is a RED5 based system that allows you to build [http://www.streamingvideosoftware.info] comprehensive pay per minute / pay per view video chat site.
- http://pixelquote.com Huge Pixelwall where visitors can simply add Pixels with their Messages - by Simon Kusterer.
- http://nonoba.com/chris/fridge-magnets Classical fridge magnet toy.
- http://www.quarterlife.com Video blogging
- http://www.avchat.net Red5 Flash Audio/Video Chat Software
- http://www.avchat.net/fms-bandwidth-checker.php Red5 bandwidth checker with upload/ download and latency tests
- http://www.justepourrire-nantes.fr Red5 Flash Video streaming
- http://www.nielsenaa.com/TV/tv.php Red5 Flash Php/MySql/Ajax driven scheduled & streamed multi channel TV - VOD
- http://www.videoflashchat.com VideoFlashChat - Red5 version for Web Based Video Chat
- http://www.videogirls.biz VideoGirls BiZ - Red5 version for Pay Per View Video Chat Software
- http://www.ligachannel.com Ligachannel.com - Italian singer site. Red5 used for VOD Protected Streaming and audio/video reco]ding widgets
- http://www.sticko.com/ Video portal with widgets for popular social networking sites
- http://www.zingaya.jp/ VOIP server built on Red5 for Flashphone

- http://www.gchats.com/red5chat/visichat/ Visichat, flash video and audio chat with red5
- http://www.agileagenda.com/ The AgileAgenda web service was written with Red5
- http://www.videoondemandsoftware.com RED5 based Video on demand HD-TV quality pay per view/minutes software
- http://www.videochatsoftware.org Flash red5 video chat software
- http://www.hubbabubba.com/ HubbaBubba world
- http://www.deltastrike.org/ DeltaStrike - free online realtime multiplayer strategy game

2.2.11.3. Is there any professional support?

Companies Listed:

- Infrared5 (www.infrared5.com [http://www.infrared5.com])
- Red5Server (http://www.red5server.com)

2.2.11.4. Are there hosting solutions?

- Red5Server (http://www.red5server.com)

2.2.11.5. Are there forums?

see: http://red5server.com/forum/

2.2.11.6. What is Jedai?

see: http://jedai.googlecode.com

2.2.11.7. Are there any frameworks that I can start with?

- see: http://jedai.googlecode.com
- see: http://paperworld3d.googlecode.com

2.2.11.8. Are there development tools?

See http://trac.red5.org/wiki/Red5Plugin

2.2.11.9. What is Paperworld3D ?

See http://www.paperworld3d.org

2.2.11.10. What Red5 groups can I join?

Linked in Red5 group [http://www.linkedin.com/e/gis/64004/24689F7691AB]

2.2.12. Troubleshooting

2.2.12.1. Why am I receiving "closing due to long handshake?

```
issue: Closing RTMPMinaConnection from [IP_ADDRESS] : 2610 to [IP_ADDRESS] (in: 3415
out 3212 ), with id 512231886 due to long handshake
```

solution: Have you installed the example your trying to connect to? The examples are installed on demand starting with Red5 0.8. Just check the welcome page http://localhost:5080/ and look for a link that allows you to install them. After an example is installed, you should be able to run the examples.

Note: We are improving this so that if an example is chosen, it will be installed.

3.1. Directory "conf"

3.1.1. jetty.xml

The settings of the HTTP server and servlet container are specified using this file. It runs on all available interfaces on port 5080 by default.

See the Jetty homepage http://jetty.mortbay.org/jetty6/ for further information about the syntax of this file.

3.1.2. keystore

Contains a sample private key and certificate to be used for secure connections.

3.1.3. log4j.properties

Controls the log levels and output handlers for the logging subsystem.

Further information about log4j can be found on the official homepage http://logging.apache.org/log4j/docs/.

3.1.4. realm.properties (Jetty)

This file defines users passwords and roles that can be used for protected areas.

The format is:

```
<username>: <password>[,<rolename> ...]
```

Passwords may be clear text, obfuscated or checksummed. The class "org.mortbay.util.Password" should be used to generate obfuscated passwords or password checksums

3.1.5. tomcat-users.xml (Tomcat)

This file defines users passwords and roles that can be used for protected areas.

The format is:

```
<user name="<username>" password="<password>" roles="[,<rolename> ...]" />
```

Passwords may be clear text, obfuscated or checksummed. For information on different digest support or available realm implementations use the how-to: http://tomcat.apache.org/tomcat-5.5-doc/realm-howto.html

Further information about tomcat realms can be found on the official homepage http://tomcat.apache.org/tomcat-5.5-doc/catalina/docs/api/org/apache/catalina/realm/package-summary.html

3.1.6. red5.globals

Specifies the path to the configuration file for the default global context to be used for Red5.

By default this file is located in "/webapps/red5-default.xml".

3.1.7. red5.properties

File containing key / value pairs to configure the host and port of basic services like RTMP or remoting.

3.1.8. red5.xml

The main configuration file that wires together the context tree. It takes care of loading "red5-common.xml" and "red5-core.xml" and sets up the rest of the server. This is the first file to be loaded by Red5. The J2EE container is selected in this configuration file by configuring one of the following bean elements.

- Jetty

```
<bean id="jetty6.server" class="org.red5.server.JettyLoader" init-method="init" autowire="byType" />
```

- Tomcat

```
<bean id="tomcat.server" class="org.red5.server.TomcatLoader" init-method="init" destroy-method="shutd
   ... cut for brevity ...
</bean>
```

3.1.9. red5-common.xml

Classes that are shared between all child contexts are declared in this file. It contains information about the object serializers / deserializers, the codecs to be used for the network protocols as well as the available video codecs. Configuration files used by Red5

The object (FLV) cache is configured / spring-wired in this file. Four implementations are currently available; The first one is our own creation (simple byte-buffers) and the others use WhirlyCache, or Ehcache. If no caching is desired then the NoCache implementation should be specified like so:

```
<bean id="object.cache" class="org.red5.server.cache.NoCacheImpl"/>
```

The other bean configurations are as follows (Only one may be used at a time):

- Red5 homegrown simple example

```
<bean id="object.cache" class="org.red5.server.cache.CacheImpl" init-method="init" autowire="byType">
   <property name="maxEntries"><value>5</value></property>
</bean>
```

- EhCache http://ehcache.sourceforge.net/

```
        <bean id="object.cache" class="org.red5.server.cache.EhCacheImpl" init-method="init">
           <property name="diskStore" value="java.io.tmpdir" />
           <property name="memoryStoreEvictionPolicy" value="LFU" />
           <property name="cacheManagerEventListener"><null/></property>
           <property name="cacheConfigs">
              <list>
                 <bean class="net.sf.ehcache.config.CacheConfiguration">
                    <property name="name" value="flv.cache" />
                    <property name="maxElementsInMemory" value="5" />
                    <property name="eternal" value="false" />
                    <property name="timeToIdleSeconds" value="0" />
                    <property name="timeToLiveSeconds" value="0" />
                    <property name="overflowToDisk" value="false" />
                    <property name="diskPersistent" value="false" />
                 </bean>
              </list>
           </property>
        </bean>
```

- Whirlycache https://whirlycache.dev.java.net/

```
<bean id="object.cache" class="org.red5.server.cache.WhirlyCacheImpl" init-method="init" autowire="b
<property name="maxEntries" value="5" />
<property name="cacheConfig">
<bean class="com.whirlycott.cache.CacheConfiguration">
<property name="name" value="flv.cache" />
<property name="maxSize" value="5" />
<!-- This policy removes cached items, biased towards least frequently used (LFU) Items -->
<property name="policy"><value>com.whirlycott.cache.policy.LFUMaintenancePolicy</value></property>
<!-- This policy removes cached items, biased towards least recently used (LRU) Items -->
<!-- property name="policy"><value>com.whirlycott.cache.policy.LRUMaintenancePolicy</value></propert
<!-- This policy removes cache items in the order in which they were added -->
Configuration files used by Red5
<!-- property name="policy"><value>com.whirlycott.cache.policy.FIFOMaintenancePolicy</value></proper
<!-- A predicate for filtering Collections of Items based on their expiration time -->
<!-- property name="policy"><value>com.whirlycott.cache.policy.ExpirationTimePredicate</value></prop
<!-- property name="backend"><value>com.whirlycott.cache.impl.ConcurrentHashMapImpl</value></propert
<property name="backend"><value>com.whirlycott.cache.impl.FastHashMapImpl</value></property>
</bean>
```

3.1.10. red5-core.xml

All available network services are specified here. By default these are RTMP and RTMPT. The actual settings for the RTMPT server can be found in "red5-rtmpt.xml" when using Jetty as the J2EE container. The RTMPT handler is selected by configuring one of the following bean elements.

- Jetty

```
<bean id="rtmpt.server" class="org.red5.server.net.rtmpt.RTMPTLoader" init-method="init" autowire="bTy
```

- Tomcat

```
<bean id="rtmpt.server" class="org.red5.server.net.rtmpt.TomcatRTMPTLoader" init-method="init" autowir
   ... cut for brevity ...
</bean>
```

3.1.11. red5-rtmpt.xml

Sets up the mapping between the RTMPT URLs and the servlets to use as well as specify the host and port to run on. By default the RTMPT server runs on all available interfaces on port 8088.

See the Jetty homepage http://jetty.mortbay.org/jetty6/ for further information about the syntax of this file.

3.1.12. web.xml (Tomcat)

Default web.xml file used by Tomcat. The settings from this file are applied to a web application before it's own WEB_INF/web.xml file. Further info about the configuration of this file may be found here: http://tomcat.apache.org/tomcat-5.5-doc/jasper-howto.html#Configuration

3.1.13. web-default.xml (Jetty)

Default web.xml file used by Jetty. The settings from this file are applied to a web application before it's own WEB_INF/web.xml file.

3.2. Webapp config directory

3.2.1. red5-web.xml

Red5 applications are configured within this file. The scripting implementations or Java applications are configured via Spring bean elements.

- Java Application

```
<bean id="web.handler" class="org.red5.server.webapp.oflaDemo.Application" singleton="true" />
```

- Javascript / Rhino application

```
<bean id="web.handler" class="org.red5.server.script.rhino.RhinoScriptFactory">
   <constructor-arg index="0" value="classpath:applications/main.js"/>
   <!-- Implemented interfaces -->
   <constructor-arg index="1">
      <list>
         <value>org.red5.server.api.IScopeHandler</value>
         <value>org.red5.server.adapter.IApplication</value>
      </list>
   </constructor-arg>
   <!-- Extended class -->
   <constructor-arg index="2">
      <value>org.red5.server.adapter.ApplicationAdapter</value>
   </constructor-arg>
</bean>
```

- Ruby application

```
<bean id="web.handler" class="org.red5.server.script.jruby.JRubyScriptFactory">
   <constructor-arg index="0" value="classpath:applications/main.rb"/>
   <constructor-arg index="1">
      <list>
         <value>org.red5.server.api.IScopeHandler</value>
         <value>org.red5.server.adapter.IApplication</value>
      </list>
   </constructor-arg>
</bean>
```

This document describes API differences between the Macromedia Flash Communication Server / Adobe Flash Media Server and Red5. It aims at helping migrate existing applications to Red5.

If you don't have an application in Red5 yet, please read the tutorial about howto create new applications first.

4.1. Application callbacks

When implementing serverside applications, one of the most important functionalities is to get notified about clients that connect or disconnect and to be informed about the creation of new instances of the application.

4.1.1. Interface IScopeHandler

Red5 specifies these actions in the interface IScopeHandler http://dl.fancycode.com/red5/api/org/red5/server/api/IScopeHandler.html. See the API documentation for further details.

4.1.2. Class ApplicationAdapter

As some methods may be called multiple times for one request (e.g. connect will be called once for every scope in the tree the client connects to), the class ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html defines additional methods.

This class usually is used as base class for new applications.

Here is a short overview of methods of the FCS / FMS application class and their corresponding methods of ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html in Red5:

Table 4.1. FMS to Red5 Callback Methods

onAppStart	appStart / roomStart
onAppStop	appStop / roomStop
onConnect	appConnect / roomConnect / appJoin / roomJoin
onDisconnect	appDisconnect / roomDisconnect / appLeave / roomLeave

The app'" methods are called for the main application, the room'" methods are called for rooms (i.e. instances) of the application.

You can also also use the ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html to check for streams, shared objects, or subscribe them. See the API documentation for further details.

4.1.2.1. Execution order of connection methods

Assuming you connect to rtmp://server/app/room1/room2

At first, the connection is established, so the user "connects" to all scopes that are traversed up to room2:

1. app (-> appConnect)
2. room1 (-> roomConnect)
3. room2 (-> roomConnect)

After the connection is established, the client object is retrieved and if it's the first connection by this client to the scope, he "joins" the scopes:

1. app (-> appJoin)
2. room1 (-> roomJoin)
3. room2 (-> roomJoin)

If the same client establishes a second connection to the same scope, only the connect methods will be called. If you conect to partially the same scopes, only a few join methods might be called, e.g. rtmp://server/app/room1/room3 will trigger

1. appConnect("app")
2. joinConnect("room1")
3. joinConnect("room3")
4. roomJoin("room3")

The appStart method currently is only called once during startup of Red5 as it currently can't unload/load applications like FCS/FMS does. The roomStart methods are called when the first client connects to a room.

4.1.3. Accepting / rejecting clients

FCS / FMS provide the methods acceptConnection and rejectConnection to accept and reject new clients. To allow clients to connect, no special action is required by Red5 applications, the *Connect methods just need to return true in this case.

If a client should not be allowed to connect, the method rejectClient can be called which is implemented by the ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html class. Any parameter passed to rejectClient is available as the application property of the status object that is returned to the caller.

4.2. Current connection and client

Red5 supports two different ways to access the current connection from an invoked method. The connection can be used to get the active client and the scope he is connected to. The first possibility uses the "magic" Red5 http://dl.fancycode.com/red5/api/org/red5/server/api/Red5.html object:

```
import org.red5.server.api.IClient;
import org.red5.server.api.IConnection;
```

```
import org.red5.server.api.IScope;
import org.red5.server.api.Red5;
public void whoami() {
    IConnection conn = Red5.getConnectionLocal();
    IClient client = conn.getClient();
    IScope scope = conn.getScope();
    // ...
}
```

The second possiblity requires the method to be defined with an argument of type IConnection http://dl.fancycode.com/red5/api/org/red5/server/api/IConnection.html as implicit first parameter which is automatically added by Red5 when a client calls the method:

```
import org.red5.server.api.IClient;
import org.red5.server.api.IConnection;
import org.red5.server.api.IScope;
public void whoami(IConnection conn) {
    IClient client = conn.getClient();
    IScope scope = conn.getScope();
    // ...
}
```

4.3. Additional handlers

For many applications, existing classes containing application logic that is not related to Red5 are required to be reused. In order to make them available for clients connecting through RTMP, these classes need to be registered as handlers in Red5.

There are currently two ways to register these handlers:

1. By adding them to the configuration files.
2. By registering them manually from the application code.

The handlers can be executed by clients with code similar to this:

```
nc = new NetConnection();
nc.connect("rtmp://localhost/myapp");
nc.call("handler.method", nc, "Hello world!");
```

If a handler is requested, Red5 always looks it up in the custom scope handlers before checking the handlers that have been set up in the context through the configuration file.

4.3.1. Handlers in configuration files

This method is best suited for handlers that are common to all scopes the application runs in and that don't need to change during the lifetime of an application.

To register the class com.fancycode.red5.HandlerSample as handler sample, the following bean needs to be added to WEB-INF/red5-web.xml:

```
<bean id="sample.service"
   class="com.fancycode.red5.HandlerSample"
   singleton="true" />
```

Note that the id of the bean is constructed as the name of the handler (here sample) and the keyword service.

4.3.2. Handlers from application code

All applications that use handlers which are different for the various scopes or want to change handlers, need a way to register them from the serverside code. These handlers always override the handlers configured in red5-web.xml. The methods required for registration are described in the interface IServiceHandlerProvider http://dl.fancycode.com/red5/api/org/red5/server/api/service/IServiceHandlerProvider.html which is implemented by ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html.

The same class as above can be registered using this code:

```
public boolean appStart(IScope app) {
    if (!super.appStart(scope))
        return false;
    Object handler = new com.fancycode.red5.HandlerSample();
    app.registerServiceHandler("sample", handler);
    return true;
}
```

Note that in this example, only the application scope has the sample handler but not the subscopes! If the handler should be available in the rooms as well, it must be registered in roomStart for the room scopes.

4.4. Calls to client methods

To call methods from your Red5 application on the client, you will first need a reference to the current connection object:

```
import org.red5.server.api.IConnection;
import org.red5.server.api.Red5;
import org.red5.server.api.service.IServiceCapableConnection;
...
IConnection conn = Red5.getConnectionLocal();
```

If the connection implements the IServiceCapableConnection http://dl.fancycode.com/red5/api/org/red5/server/api/service/IServiceCapableConnection.html interface, it supports calling methods on the other end:

```
if (conn instanceof IServiceCapableConnection) {
    IServiceCapableConnection sc = (IServiceCapableConnection) conn;
    sc.invoke("the_method", new Object[]{"One", 1});
}
```

If you need the result of the method call, you must provide a class that implements the IPendingServiceCallback http://dl.fancycode.com/red5/api/org/red5/server/api/service/IPendingServiceCallback.html interface:

```
import org.red5.server.api.service.IPendingService;
import org.red5.server.api.service.IPendingServiceCallback;
class MyCallback implements IPendingServiceCallback {
    public void resultReceived(IPendingServiceCall call) {
        // Do something with "call.getResult()"
    }
}
```

The method call looks now like this:

```
if (conn instanceof IServiceCapableConnection) {
    IServiceCapableConnection sc = (IServiceCapableConnection) conn;
    sc.invoke("the_method", new Object[]{"One", 1}, new MyCallback());
}
```

Of course you can implement this interface in your application and pass a reference to the application instance.

4.5. SharedObjects

The methods to access shared objects from an application are specified in the interface ISharedObjectService http://dl.fancycode.com/red5/api/org/red5/server/api/so/ISharedObjectService.html.

When dealing with shared objects in serverside scripts, special care must be taken about the scope they are created in.

To create a new shared object when a room is created, you can override the method roomStart in your application:

```
import org.red5.server.adapter.ApplicationAdapter;
import org.red5.server.api.IScope;
import org.red5.server.api.so.ISharedObject;
public class SampleApplication extends ApplicationAdapter {
  public boolean roomStart(IScope room) {
      if (!super.roomStart(room))
```

```
            return false;
        createSharedObject(room, "sampleSO", true);
        ISharedObject so = getSharedObject(room, "sampleSO");
        // Now you could do something with the shared object...
        return true;
    }
}
```

Now everytime a first user connects to a room of a application, e.g. through rtmp://server/application/room1, a shared object sampleSO is created by the server.

If a shared object should be created for connections to the main application, e.g. rtmp://server/application, the same must be done in the method appStart.

For further informations about the possible methods a shared object provides please refer to the api documentation of the interface ISharedObject http://dl.fancycode.com/red5/api/org/red5/server/api/so/ISharedObject.html.

4.5.1. Serverside change listeners

To get notified about changes of the shared object similar to onSync in FCS / FMS, a listener must implement the interface ISharedObjectListener http://dl.fancycode.com/red5/api/org/red5/server/api/so/ISharedObjectListener.html:

```
import org.red5.server.api.so.ISharedObject;
import org.red5.server.api.so.ISharedObjectListener;
public class SampleSharedObjectListener
Migration Guide
        implements ISharedObjectListener {
    public void onSharedObjectUpdate(ISharedObject so,
                                     String key, Object value) {
        // The attribute &lt;key&gt; of the shared object &lt;so&gt;
        // was changed to &lt;value&gt;.
    }
    public void onSharedObjectDelete(ISharedObject so, String key) {
        // The attribute &lt;key&gt; of the shared object &lt;so&gt; was deleted.
    }
    public void onSharedObjectSend(ISharedObject so,
                                   String method, List params) {
        // The handler &lt;method&gt; of the shared object &lt;so&gt; was called
        // with the parameters &lt;params&gt;.
    }
    // Other methods as described in the interface...
}
```

Additionally, the listener must get registered at the shared object:

```
ISharedObject so = getSharedObject(scope, "sampleSO");
so.addSharedObjectListener(new SampleSharedObjectListener())
```

4.5.2. Changing from application code

A shared object can be changed by the server as well:

```
ISharedObject so = getSharedObject(scope, "sampleSO");
so.setAttribute("fullname", "Sample user");
```

Here all subscribed clients as well as the registered handlers are notified about the new / changed attribute.

If multiple actions on a shared object should be combined in one update event to the subscribed clients, the methods beginUpdate and endUpdate must be used:

```
ISharedObject so = getSharedObject(scope, "sampleSO");
so.beginUpdate();
so.setAttribute("One", "1");
so.setAttribute("Two", "2");
so.removeAttribute("Three");
so.endUpdate();
```

The serverside listeners will receive their update notifications through separate method calls as without the beginUpdate and endUpdate.

Calls to shared object handlers through remote_so.send(<handler>, <args>) from a Flash client or the corresponding serverside call can be mapped to methods in Red5. Therefore a handler must get registered through a method of the ISharedObjectHandlerProvider http://dl.fancycode.com/red5/api/org/red5/server/api/so/ISharedObjectHandlerProvider.html interface similar to the application handlers:

```
package com.fancycode.red5;
class MySharedObjectHandler {
    public void myMethod(String arg1) {
        // Now do something
    }
}
...
ISharedObject so = getSharedObject(scope, "sampleSO");
so.registerServiceHandler(new MySharedObjectHandler());
```

Handlers with a given name can be registered as well:

```
ISharedObject so = getSharedObject(scope, "sampleSO");
so.registerServiceHandler("one.two", new MySharedObjectHandler());
```

Here, the method could be called through one.two.myMethod. Another way to define event handlers for SharedObjects is to add them to the red5- web.xml similar to the file-based application handlers. The beans must have a name of

<SharedObjectName>.<DottedServiceName>.soservice, so the above example could also be defined with:

```
<bean id="sampleSO.one.two.soservice"
   class="com.fancycode.red5.MySharedObjectHandler"
   singleton="true" />
```

4.6. Persistence

Persistence is used so properties of objects can be used even after the server has been restarted. In FCS / FMS usually local shared objects on the serverside are used for this.

Red5 allows arbitrary objects to be persistent, all they need to do is implement the interface IPersistable http://dl.fancycode.com/red5/api/org/red5/server/api/persistence/IPersistable.html. Basically these objects have a type, a path, a name (all strings) and know how to serialize and deserialize themselves.

Here is a sample of serialization and deserialization:

```
import java.io.IOException;
import org.red5.io.object.Input;
import org.red5.io.object.Output;
import org.red5.server.api.persistence.IPersistable;
class MyPersistentObject implements IPersistable {
  // Attribute that will be made persistent
  private String data = "My persistent value";
  void serialize(Output output) throws IOException {
      // Save the objects's data.
      output.writeString(data);
  }
  void deserialize(Input input) throws IOException {
      // Load the object's data.
      data = input.readString();
  }
  // Other methods as described in the interface...
}
```

To save or load this object, the following code can be used:

```
import org.red5.server.adapter.ApplicationAdapter;
import org.red5.server.api.IScope;
import org.red5.server.api.Red5;
import org.red5.server.api.persistence.IPersistenceStore;
class MyApplication extends ApplicationAdapter {
  private void saveObject(MyPersistentObject object) {
      // Get current scope.
      IScope scope = Red5.getConnectionLocal().getScope();
      // Save object in current scope.
      scope.getStore().save(object);
  }
  private void loadObject(MyPersistentObject object) {
```

```
      // Get current scope.
      IScope scope = Red5.getConnectionLocal().getScope();
      // Load object from current scope.
      scope.getStore().load(object);
  }
}
```

If no custom objects are required for an application, but data must be stored for future reuse, it can be added to the IScope http://dl.fancycode.com/red5/api/org/red5/server/api/IScope.html through the interface IAttributeStore http://dl.fancycode.com/red5/api/org/red5/server/api/IAttributeStore.html. In scopes, all attributes that don't start with IPersistable.TRANSIENT_PREFIX are persistent.

The backend that is used to store objects is configurable. By default persistence in memory and in the filesystem is available.

When using filesystem persistence for every object a file is created in "webapps/<app>/ persistence/<type>/<path>/<name>.red5", e.g. for a shared object "theSO" in the connection to "rtmp://server/myApp/room1" a file at "webapps/myApp/persistence/SharedObject/room1/theSO.red5" would be created.

4.7. Periodic events

Applications that need to perform tasks regularly can use the setInterval in FCS / FMS to schedule methods for periodic execution.

Red5 provides a scheduling service (ISchedulingService http://dl.fancycode.com/red5/api/org/red5/server/api/scheduling/ISchedulingService.html) that is implemented by ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html like most other services. The service can register an object (which needs to implement the IScheduledJob http://dl.fancycode.com/red5/api/org/red5/server/api/scheduling/IScheduledJob.html interface) whose execute method is called in a given interval.

To register an object, code like this can be used:

```
import org.red5.server.api.IScope;
import org.red5.server.api.IScheduledJob;
import org.red5.server.api.ISchedulingService;
import org.red5.server.adapter.ApplicationAdapter;
class MyJob implements IScheduledJob {
  public void execute(ISchedulingService service) {
      // Do something
  }
}
public class SampleApplication extends ApplicationAdapter {
  public boolean roomStart(IScope room) {
      if (!super.roomStart(room))
          return false;
      // Schedule invokation of job every 10 seconds.
      String id = addScheduledJob(10000, new MyJob());
      room.setAttribute("MyJobId", id);
      return true;
  }
```

```
}
```

The id that is returned by addScheduledJob can be used later to stop execution of the registered job:

```
public void roomStop(IScope room) {
    String id = (String) room.getAttribute("MyJobId");
    removeScheduledJob(id);
    super.roomStop(room);
}
```

4.8. Remoting

Remoting can be used by non-rtmp clients to invoke methods in Red5. Another possibility is to call methods from Red5 to other servers that provide a remoting service.

4.8.1. Remoting server

Services that should be available for clients need to be registered the same way as additional application handlers are registered. See above for details.

To enable remoting support for an application, the following section must be added to the WEB-INF/web.xml file:

web.xml -

```
<servlet>
   <servlet-name>gateway</servlet-name>
   <servlet-class>
      org.red5.server.net.servlet.AMFGatewayServlet
   </servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>gateway</servlet-name>
   <url-pattern>/gateway/*</url-pattern>
</servlet-mapping>
```

The path specified in the <url-pattern> tag (here gateway) can be used by the remoting client as connection url. If this example would have been specified for an application myApp, the URL would be:

```
http://localhost:5080/myApp/gateway
```

Methods invoked through this connection will be executed in the context of the application scope. If the methods should be executed in subscopes, the path to the subscopes must be added to the URL like:

```
http://localhost:5080/myApp/gateway/room1/room2
```

4.8.2. Remoting client

The class RemotingClient http://dl.fancycode.com/red5/api/org/red5/server/net/remoting/RemotingClient.html defines all methods that are required to call methods through the remoting protocol.

The following code serves as example about how to use the remoting client:

```
import org.red5.server.net.remoting.RemotingClient;
String url = "http://server/path/to/service";
RemotingClient client = new RemotingClient(url);
Object[] args = new Object[]{"Hello world!"};
Object result = client.invokeMethod("service.remotingMethod", args);
// Now do something with the result
```

By default, a timeout of 30 seconds will be used per call, this can be changed by passing a second parameter to the constructor defining the maximum timeout in milliseconds.

The remoting headers AppendToGatewayUrl, ReplaceGatewayUrl and RequestPersistentHeader are handled automatically by the Red5 remoting client.

Some methods may take a rather long time on the called server to complete, so it's better to perform the call asynchronously to avoid blocking a thread in Red5. Therefore an object that implements the interface IRemotingCallback http://dl.fancycode.com/red5/api/org/red5/server/net/remoting/IRemotingCallback.html must be passed as additional parameter:

```
import org.red5.server.net.remoting.RemotingClient;
import org.red5.server.net.remoting.IRemotingCallback;
public class CallbackHandler implements IRemotingCallback {
  void errorReceived(RemotingClient client, String method,
                     Object[] params, Throwable error) {
      // An error occurred while performing the remoting call.
  }
  void resultReceived(RemotingClient client, String method,
                      Object[] params, Object result) {
      // The result was received from the server.
  }
}
String url = "http://server/path/to/service";
RemotingClient client = new RemotingClient(url);
Object[] args = new Object[]{"Hello world!"};
IRemotingCallback callback = new CallbackHandler();
client.invokeMethod("service.remotingMethod", args, callback);
```

4.9. Streams

TODO: How can streams be accessed from an application?

5.1. Spring scripting support

- http://www.mozilla.org/rhino/
- http://static.springframework.org/spring/docs/2.0.2/reference/dynamic-language.html#dynamic
- spring-support.jar

5.2. Groovy

- http://groovy.codehaus.org/
- asm-2.2.2.jar
- antlr-2.7.6.jar
- groovy-1.0.jar

5.3. Beanshell

- http://www.beanshell.org/
- bsh-2.0b5.jar
- cglib-nodep-2.1_3.jar

5.4. Ruby

- http://jruby.codehaus.org/
- jruby.jar
- cglib-nodep-2.1_3.jar

5.5. Jython / Python

- http://www.jython.org/Project/index.html
- jython.jar

5.6. Java 5 Libraries

The following are need for Java 5 only:

5.7. Script related JSR's

- jsr-223-1.0-pr.jar
- jsr173_1.0_api.jar

5.8. Javascript / Rhino

- http://www.mozilla.org/rhino/
- js.jar
- xbean.jar - needed for E4X

6.1. Build Environment Setup

6.1.1. Ant

Apache Ant 1.7 and above is required for building the Red5 project source code. download here http://archive.apache.org/dist/ant/binaries/

The path to the ant binary must be on your system PATH environment variable (test by typing ant -version at a system prompt) defined, typically

```
PATH=$PATH:/usr/local/ant
```

You can check this on windows by typing set PATH or on unix by typing echo $PATH

6.1.2. Java

Java 1.5 or 1.6 and above is required for running ant, compiling the source and running the Red5 server.

Download Java 5 http://java.sun.com/j2se/1.5.0/download.html

Download Java 6 http://java.sun.com/j2se/1.6.0/download.html

You must have the environment variables for JAVA_HOME and JAVA_VERSION defined, typically

```
JAVA_HOME=C:\development\j2sdk\1.5.0_07 JAVA_VERSION=1.5
```

You can check this on windows by typing

```
set JAVA_HOME
```

or on unix by typing

```
$ echo $JAVA_HOME
```

6.1.3. Red5

You must have the environment variables for RED5_HOME defined, typically

```
RED5_HOME=/www/red5_server
```

Warning

FAILURE TO SETUP YOUR ENVIRONMENT VARIABLES WILL PREVENT YOUR FROM BEING ABLE TO BUILD PROPERLY

Note

You don't need netbeans or eclipse unless you need an IDE for java. Download Netbeans [http://www.netbeans.org] Download Eclipse [http://www.eclipse.org]

6.2. Building

6.2.1. Getting Red5 Source

The Red5 source code is available at the google code project page [http://code.google.com/p/red5/] and svn repository.

1. With your favourite SVN client check out the source code from svn at this address http://red5.googlecode.com/svn/java/server/trunk/ or https://red5.googlecode.com/svn/java/server/trunk/ if you have a google code login.
2. Team members will be added to the google code project group and in your google code login you will find the svn password for committing changes at this address http://code.google.com/hosting/settings.

6.2.2. Getting Red5 Demo Applications Source

1. With your favourite SVN client check out the source code from svn at this address http://red5.googlecode.com/svn/java/example/trunk/ or https://red5.googlecode.com/svn/java/example/trunk/ if you have a google code login.

6.2.3. Getting Red5 Flash Demo Source

1. With your favourite SVN client check out the source code from svn at this address http://red5.googlecode.com/svn/flash/trunk/ or https://red5.googlecode.com/svn/flash/trunk/ if you have a google code login.

6.2.4. Running the ant build

To build the red5 source simply run the following command from the command line inside the red5 source directory.

```
$ ant dist
```

6.2.5. Current Ant Targets

- all - Runs clean, prepare, compile, jar, javadoc targets
- bootstrap - Compile and start the server using the bootstrap class
- checkout - checks out the Red5 server source (requires svnant.jar)
- checkout-all - checks out the entire Red5 project sources from the root level to a specified directory
- clean - cleans up all the files and directories

- compile - Compiles Red5
- compile_core - Build Red5 server sources and downloads java 6 dependancies
- compile_core_compatibility - Build Red5 server sources and downloads java 5 dependancies
- compile_demos - Copies over the root and installer webapp
- compile_script - Compiles scripting sources
- compile_tests - Compiles junit test classes
- compile_war - Compiles Red5 into a war distribution
- console - launches a non-SSL jconsole for managing Red5 in JMX.
- console-ssl - launches a SSL jconsole for managing Red5 in JMX with SSL enabled.
- doc-all - Generate docbook documentation for html-single, multi html and pdf.
- doc-clean - Cleans the docbook files.
- doc-html - Compile reference documentation to chunked html.
- doc-htmlsingle - Compile reference documentation to single html.
- doc-pdf - Compile reference documentation to pdf.
- doc-prepare - Extra preparation for the documentation.
- dist - Make Binary distribution.
- dist-archive - Create archive file for distribution.
- dist-cluster - Create Edge/Origin distribution.
- dist-debian - Create Debian package.
- dist-edge - Builds a Red5 edge distribution.
- dist-origin - Builds a Red5 origin distribution.
- dist-installer - Make Installer distribution.
- dist-macosx - Create Mac OSX installer.
- dist-windows - Create Windows installer.
- dist-redhat - Create Redhat installer.
- ivyclear - Clears out the Ivy cache.
- jar-determine-classpath - Determine classpath for jar file.
- jar - Make Archive.
- javadoc - Generate JavaDoc.

- java6.check - Checks for Java 6.
- prepare - Prepares for building Red5.
- server - Compile and start the server.
- shutdown - Shuts down the running Red5 instance.
- udp_server - Compile and start experimental UDP server.
- run-tests - Run JUnit tests and generate HTML reports.
- run-tests-systemtest - Runs some end-to-end system tests against a test server using a flash client.
- run-tests-server - Run the selftest server.
- svn-add - Add files to svn.
- remotejar - Creates a jar that may be deployed with remote applications.
- retrieve - Retrieves the libraries if needed.
- rtmps_keystore - Creates the keystore file in the conf directory required by RTMPS.
- truststore - Creates a duplicate keystore file and generates a truststore file for jconsole SSL connections.
- upload-snapshot - Uploads a snapshot of Red5 to the repository.
- war_demos - Build wars for demo apps.
- webwar - Make Web Archive.

6.2.6. Ant and Ivy

When cleaning the dependancy libraries using and ant ivy with the following command

```
$ ant ivyclear
```

It is required to run the rebuild of Red5 in a particular way to make sure ivy retrieved the libraries correctly.

```
$ ant -Divy.conf.name="java6, eclipse" dist
```

6.3. How to build with eclipse

This guide assumes eclipse 3.1.0 and you have downloaded the entire red5 build from the subversion repository at https://red5.googlecode.com/svn/java/server/trunk

6.3.1. Recommended Eclipse Plugins

The following plugins are recommended or required for building red5 in eclipse.

- IvyIDE - http://ant.apache.org/ivy/ivyde/download.cgi. See here for installation / update instructions
- Spring IDE - http://springide.org/project/wiki
- Subclipse SVN Plugin - http://subclipse.tigris.org/

6.3.2. Importing the Red5 Project

There are two ways to import the Red5 project. Either import an already downloaded working copy of the Red5 project or import the project directly from SVN.

Procedure 6.1. Import the checked out working copy.

1. Start Eclipse.
2. Begin to import project File → Import
3. In the Import dialog box select the item Existing Projects into Workspace and hit next.
4. Hit the browse button next to the Select root directory text box.
5. Select the root folder where you downloaded the red5 repository,(e.g. c:\projects \osflash\red5 or /www/red5_server) and hit ok.
6. Make sure red5 is selected in the projects area and hit Finish.
7. Eclipse should automatically build the project, you can force a build from the menu Project → Build Project

Procedure 6.2. Import the project working copy from SVN. (Subclipse must be installed).

1. Begin to import project File → Import
2. In the Import dialog box select SVN and then select the item Checkout Projects from SVN and hit next.
3. A list of available SVN urls will be available, if it is not available select Create a new repository location click Next and enter. http://red5.googlecode.com/svn/java/server/trunk or.\ https://red5.googlecode.com/svn/java/server/trunk if you have a google code login.
4. Click Finish.
5. Eclipse should automatically build the project, you can force a build from the menu Project → Build Project

6.3.3. Updating the Red5 source from Eclipse.

Procedure 6.3. Updating the Red5 source from trunk.

1. In eclipse right click the Red5 source project.

2. Locate to Team → Update
3. The source will be updated from SVN.
4. Right click the Red5 project and select Refresh.
5. The project should also be cleaned after each update, by the following Project → Clean

6.3.4.

Procedure 6.4. Debugging Red5 in Eclipse.

1. Click the arrow next to the icon menu and then click Debug Configurations.
2. Click Java Application in the menu then right click and New.
3. Type a name for the debug configuration (ie Red) and type org.red5.server.Bootstrap as the main class.
4. Select the Arguments tab.
5. In the Program Arguments enter

```
-Dlogback.ContextSelector=org.red5.logging.LoggingContextSelector -Dcatalina.useNaming=tr
```

6. In the VM Arguments enter

```
-cp ./conf
```

7. In OSX with JDK 5 and JDK6 to specify JDK6 the PATH variable has to be set. Goto the Environment Tab, add a new variable called PATH, and place this in there.

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6/Home/bin
```

8. Click Apply and Close.
9. Goto the build configure path dialog Build Path → Configure Build Path
10. In the Source tab choose Add Folder and select the src/conf directory.
11. Make sure "Allow output folders for source folders" is selected.
12. Under red5_server/src/conf, select Output Folder and choose Edit.
13. Select Specific Output Folder, select the root directory and choose "create new folder" and choose "conf".
14. Select conf, the output folder for the Red5 configs will now be placed into red5_server/conf.

15. With the imported red5 project selected click the debug icon and it will launch the server.

16. Console logging will appear in the console window.

If you get an error in the console like:

```
java.net.BindException: Address already in use: bind at sun.nio.ch.Net.bind(Native
Method) at sun.nio.ch.ServerSocketChannelImpl.bind(Unknown
Source) at sun.nio.ch.ServerSocketAdaptor.bind(Unknown Source) at
org.apache.mina.io.socket.SocketAcceptor.registerNew(SocketAcceptor.java:362)
at org.apache.mina.io.socket.SocketAcceptor.access$800(SocketAcceptor.java:46)
at org.apache.mina.io.socket.SocketAcceptor$Worker.run(SocketAcceptor.java:238)
Exception in thread "main"
```

Then the socket red5 wants to run is in use, you can change the socket port in the property rtmp.port in the property file red5.properties.

6.3.5. Ant, Ivy and Eclipse

When cleaning the dependancy libraries using ant and ivy with the following command

```
$ ant ivyclear
```

It is required to run the rebuild of Red5 in a particular way to make sure ivy retrieved the libraries correctly.

```
$ ant -Divy.conf.name="java6, eclipse" dist
```

Then back in eclipse right click the ivy.xml in the project and click Refresh it will also resolve the libraries in Eclipse.

Red5's formatting convention begins with Sun's Code Conventions for the Java Programming Language which you will find here: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

The follwing rules used by Red5 may differ Sun's standard:

- Maximum line length is 120 characters
- Use spaces instead of tabs
- Indendation size is 4 spaces
- Use fully qualified import statements, i.e. do not use asterisks
- Do not insert a new line before opening brace. Insert a new line before closing brace

```
//This is the way we want it!
public void correctBlock() {
}

//Not like this
public void badBlock()
{
}
```

- The conditional part of if and while statements should not use the equality operator for testing true or false values.

```
//Do this
if (foo)

//Do NOT do this
if (foo==true)
```

- Always use the short form of the classname in code. Certain exception do apply where the fully qualified classname is required such as when java.sql.Date and java.util.Date are used in the same class. Otherwise, you should always import the class and use the short name.

```
//Do this
try {
} catch (IOException e) {
}

//Do NOT do this
try {
} catch (java.io.IOException e) {
}
```

7.1. Javadoc

Provide documentation to your methods in addition to providing notes within the method to help readers follow the flow. Internal method comments should be blocked by "//" and not "/*" even if they cover more than one line. If your code is in response to a bug report, add the tracking number to the comment.

Add your name and email (optional) to the class or method *@author* tag if you make a sugnificant contribution. This provides recognition as well as providing a point of contact for the modification.

Use *@todo* as a reminder of what needs to be done at a later time.

7.2. Unit Tests

All new classes and methods should have a corresponding unit test.

- Use the naming scheme *Test.java for unit tests
- Do not define public static Test suite() or constructor methods, the build system will automatically do the right thing without them

7.3. Logging

- Use SLF4J org.slf4j.Logger rather than Log4j, JCL, or Java logging (JULI)
- Log as much as necessary for someone to figure out what broke
- Do not log throwables that you throw - leave it to the caller
- Use SLF4J varargs {} expansion
- If your logging call includes a loop, put it inside an isDebugEnabled() or isTraceEnabled() block

7.3.1. Levels

- Use trace level for detailed/diagnostic logging
- Use debug level for things an application developer would need to know
- Use info level for things an administrator would need to know
- Use warn level for things indicating an application or transient problem
- Use error level for things indicating a problem with the server itself

7.4. Exceptions

- A situation is only exceptional, if the program can not handle it with reasonably effort. Wrong input data should be an expected situation of the regular code, that could be handled gracefully.

- The intention of exception-handling is to separate real error-handling from the regular part of the code, so don't force the caller to mix it with unnecessary exceptions.
- Only if your code really has a problem to continue e.g., when a parameter is invalid, feel free to throw an exception!
- Do NOT throw an exception, if you only suppose the caller of your code could have a problem with a special result. Try to return a special result value instead e.g., null, and let the caller decide with a regular if-else-statement. If the caller really has a problem, HE WILL throw an exception on his own.
- But if your code throws an exception, even though it has no real problem and it could continue without an exception and return a special result value, you forestall the decision of the caller, whether the special result is really an error or not.
- If you throw an exception, where the caller would decide that it is no error in the context of the caller, you force the caller to write an exception handler in his regular part or to abort i.e., you force the caller to mix regular code with exception handling. That is the opposite of the intentention of exception handling.

```
//Bad example
java.lang.Class.forName(String) throws ClassNotFoundException
```

- In most programs/situations it is an error if this method does not find the class, therefore it throws an exception and forestalls the decision of the caller.
- But maybe there is a program that should check a list of class names, whether the classes are present or not. Such a program is forced to mix its regular code with error handling of an exception, that is no error at all in that context.
- The method should return a special result value instead: null. Many callers of that method have expected that situation and therefore are not in an unexpected situation/ exceptional state. They could decide the situation on their own.
- Only throw checked exceptions (not derived from RuntimeException), if the caller has a chance to handle it.
- Exceptions that signal programming errors or system failures usually cannot be handled/ repaired at runtime -> unchecked exception.
- If your code really has a problem to continue e.g., when a parameter is invalid, throw an unchecked exception (derived from RuntimeException) and do NOT throw a checked exception, because if not even your code can handle the problem, in the very most cases the caller has no chance to handle the problem, too. Instead there maybe somebody somewhere in the highest layers who catches all RuntimeException's, logs them and continues the regular service.
- Only if it is not possible to return special result values cleanly, use checked exceptions to force the caller to decide the situation. The caller should deescalate the situation by catching and handling one or more checked exceptions, e.g. with special result values(?)

or by escalating with an unchecked exception, because the situation is an error, that can not be handled.

- Checked exceptions are an official part of the interface, therefore do not propagate checked exceptions from one abstraction layer to another, because usually this would break the lower abstraction. E.g. do not propagate SQLException to another layer, because SQLExceptions are an implementation detail, that may change in the future and such changes should not affect the interfaces and their callers.
- Never throw NullPointerException or RuntimeException. Use either IllegalArgumentException, or NullArgumentException (which is a subclass of IllegalArgumentException anyway). If there isn't a suitable subclass available for representing an exception, create your own.

= Eclipse Users =

7.5. Formatter Profile

The profile can be downloaded from here - red5_codeformat.xml [http://red5.googlecode.com/svn/java/server/trunk/.settings/red5_codeformat.xml]

```
Window -> Preferences
Java -> Code Style -> Formatter
Click on import and select the red5_codeformat.xml file downloaded above
Press OK after importing
```

7.6. Code Template

The latest version with the required header format can be downloaded here - red5_codetemplate.xml [http://red5.googlecode.com/svn/java/server/trunk/.settings/red5_codetemplate.xml]

```
Window -> Preferences
Java -> Code Style -> Code Templates
Click on import and select the red5_codetemplate.xml file downloaded above
Press OK after importing
```

= Submitting a Patch =

If you make changes to Red5, and would like to contribute the to the project, you should create a patch via svn and post it to the Red5 users list or via Trac. To create a patch, simply execute the following command:

```
$> svn diff > your-changes.patch
```

Note: You may also use Eclipse to create a patch (Team -> Create Patch...), but this may require committers to modify the patch to match their project layout (workspace per

branch or all branches in one workspace) and some committers may not be using Eclipse/Subclipse.

This document describes the steps necessary to create a new release of Red5:

1. Make sure everything has been committed to the trunk or correct branch.

2. Update the file doc/changelog.txt with informations about the new release.

3. Create tags of the modules that are linked into the main code tree:

    documentation at http://red5.googlecode.com/svn/doc/tags Tags for versions should always be the version string with dots replaced by underscores, e.g. version "1.2.3" becomes tag "1_2_3".

If you would tag the documentation folder for version "1.2.3", you would use the url http://red5.googlecode.com/svn/doc/tags/1_2_3

The following are suggested system requirements:

Table 9.1. Red5 System Requirements

Supported operating systems	Windows 2000 Server Windows 2003 Server, Standard Edition Linux Variants Mac OSX 10.4 and above
Minimum Hardware Requirements (Development / Budget / Low Traffic)	X86-compatible CPU (Pentium 4, 3.2 GHz c 1 GB Available Memory 100MB or 1GB Ethernet card 200 MB of available disk space (SATA II)
Recommended Hardware Requirements (High Traffic Production)	Dual-core / Quad Core (Intel XEON 2Ghz ar 2 - 4 GB Available memory or above 1GB Ethernet Card with big pipe network 200 MB of available disk space (10K RPM a Network Storage Cluster Solution for mass c
Software Requirements	Java JRE 1.5 or 1.6 Service Scripts (Java Service Wrapper, Fire

9.1. Java Memory Tweaking

- http://www.peuss.de/node/67 [http://www.peuss.de/node/67]
- http://andrigoss.blogspot.com/2008/02/jvm-performance-tuning.html [http://andrigoss.blogspot.com/2008/02/jvm-performance-tuning.html]
- http://java.sun.com/javase/technologies/hotspot/largememory.jsp [http://java.sun.com/javase/technologies/hotspot/largememory.jsp]

9.2. Real World Production Run 5-31-07 and 6-1-07

We had a very successful production event using Red 5.

Here's the server config:

Xubuntu Linux AMD 64 3500+ processor 4 GB RAM (memory hole not plugged - so only about 3 GB free RAM) Red 5 trunk ver 2082 Gbit Internet connection

Client side:

Random public users.

Publishing side:

One publishing point using the Flash player as encoder.

Event Stats:

Two day event. 6 hours each day. Video 320x240, 80-85% quality, 18 fps, 22khz audio. Secondary space for still graphics - changed them periodically.

Average number of users on line: 100 Peak number of concurrent users: 120 Bandwidth per user: 400kbps - 1 Mbps Feedback from users: Excellent quality, no problems. Total users visiting the webcast: approx 1,000

Summary:

We experienced no problems or notable events on the Red 5 server. Excellent performance.

Part II. Red5 Core Components

Core components available in Red5.

- Chapter 10, *Create new applications in Red5*
- Chapter 11, *Logging Setup*
- Chapter 12, *Deploying Red5 To Tomcat*
- Chapter 13, *Customize Stream Paths*
- Chapter 14, *Security*
- Chapter 15, *Scripting Implementations*
- Chapter 16, *Clustering*
- Chapter 17, *Management*
- Chapter 18, *Red5 Demo Applications*
- Chapter 19, *Testing Red5*
- Chapter 20, *Plugins*

This document describes how new applications can be created in Red5. It applies to the new API introduced by Red5 0.4.

10.1. The application directory

Red5 stores all application definitions as folders inside the "webapps" directory beneath the root of Red5. So the first thing you will have to do in order to create a new application, is to create a new subfolder in "webapps". By convention this folder should get the same name the application will be reached later.

Inside your new application, you will need a folder "WEB-INF" containing configuration files about the classes to use. You can use the templates provided by Red5 in the folder "doc/templates/myapp".

During the start of Red5, all folders inside "webapps" are searched for a directory "WEB-INF" containing the configuration files.

10.2. Configuration

The main configuration file that is loaded is "web.xml". It contains the following parameters:

10.2.1. webAppRootKey

Unique name for this application, should be the public name:

```
<context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>/myapp</param-value>
</context-param>
```

10.3. Handler configuration

Every handler configuration file must contain at least three beans:

10.3.1. Context

The context bean has the reserved name web.context and is used to map paths to scopes, lookup services and handlers. The default class for this is org.red5.server.Context.

By default this bean is specified as:

```
<bean id="web.context" class="org.red5.server.Context"
    autowire="byType" />
```

Every application can only have one context. However this context can be shared across multiple scopes.

10.3.2. Scopes

Every application needs at least one scope that links the handler to the context and the server. The scopes can be used to build a tree where clients can connect to every node and share objects inside this scope (like shared objects or live streams). You can see the scopes as rooms or instances.

The default scope usually has the name web.scope, but the name can be chosen arbitrarily.

The bean has the following properties:

- server This references the global server red5.server. - parent References the parent for this scope and usually is global.scope. - context The server context for this scope, use the web.context from above. - handler The handler for this scope (see below). - contextPath The path to use when connecting to this scope. - virtualHosts A comma separated list of hostnames or ip addresses this scope runs at.

A sample definition looks like this:

```
<bean id="web.scope" class="org.red5.server.WebScope"
    init-method="register">
    <property name="server" ref="red5.server" />
    <property name="parent" ref="global.scope" />
    <property name="context" ref="web.context" />
    <property name="handler" ref="web.handler" />
    <property name="contextPath" value="/myapp" />
    <property name="virtualHosts" value="localhost, 127.0.0.1" />
</bean>
```

You can move the values for contextPath and virtualHosts to a separate properties file and use parameters. In that case you need another bean:

```
<bean id="placeholderConfig"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
Create new applications in Red5
    <property name="location" value="/WEB-INF/red5-web.properties" />
</bean>
```

Assuming a red5-web.properties containing the following data:

```
webapp.contextPath=/myapp
webapp.virtualHosts=localhost, 127.0.0.1
```

the properties of the scope can now be changed to:

```
<property name="contextPath" value="${webapp.contextPath}" />
<property name="virtualHosts" value="${webapp.virtualHosts}" />
```

The contextPath specified in the configuration can be seen as "root" path of the scope.

You can add additional elements after the configured path when connecting to dynamically create extra scopes.

These extra scopes all use the same handler but have their own properties, shared objects and live streams.

10.4. Handlers

Every context needs a handler that implements the methods called when a client connects to the scope, leaves it and that contains additional methods that can be called by the client. The interface these handlers need to implement is specified by org.red5.server.api.IScopeHandler, however you can implement other interfaces if you want to control access to shared objects or streams.

A sample implementation that can be used as base class can be found at org.red5.server.adapter.ApplicationAdapter. Please refer to the javadoc documentation for further details.

The bean for a scope handler is configured by:

```
<bean id="web.handler"
    class="the.path.to.my.Application"
    singleton="true" />
```

10.5. Logging

Logging Setup Tutorial [Documentation/Tutorials/LoggingSetup]

The logging system uses Simple Logging Facade for Java (SLF4J [http://www.slf4j.org/]). This framework supports many of the logging systems available for Java and also provides simple implementations. The logging used by our dependencies are mainly Log4j and Apache commons logging and SLF4J allows us to combine them into one system. SLF4J gives you the ability to select a logging implementation and provides proxies for you dependencies if their maintainers did not select the same framework.

We prefer the logback [http://logback.qos.ch/] log implementation, but you may use whatever you like. There are some hoops you will have to jump through to get Log4j [http://logging.apache.org/] or Commons logging [http://commons.apache.org/logging/] to work. Blog post about using other loggers here [http://gregoire.org/2009/05/05/support-for-other-slf4j-loggers/].

After you chose an implementation framework, some of the SLF4J jars must **NOT** be in your applications classpath or they will cause conflicts. The default case it to use Logback, so the following jars must be included:

- slf4j-api - The core API
- logback-core - Current Logback core library
- logback-classic - Logback support library
- log4j-over-slf4j - Log4j proxy/bridge
- jcl-over-slf4j - Apache commons logging proxy/bridge
- jul-to-slf4j - java.util.logging proxy/bridge

The items denoted as "proxy/bridge" listen for the logging calls to those implementations and pass them through to SLF4J.

The following two strategies are to be consider **untested**.

If you prefer to use Log4j instead, the following jars are required:

- slf4j-api - The core API
- log4j - Current Log4j library (1.2+)
- slf4j-log4j12 - Log4j adapter
- jcl-over-slf4j - Apache commons logging proxy/bridge
- jul-to-slf4j - java.util.logging proxy/bridge

If you prefer to use Commons logging the following jars are required:

- slf4j-api - The core API
- commons-logging - Apache commons logging library
- slf4j-jcl - Commons logging adapter

- log4j-over-slf4j - Log4j proxy/bridge
- jul-to-slf4j - java.util.logging proxy/bridge

If you want to use another implementation not shown here, simply check out the faq SLF4J FAQ [http://www.slf4j.org/faq.html]

Logback is the successor of Log4j and was created by the creator of Log4j and SLF4J. A conversion tool has been created for your log4j properties files configuration converter [http://logback.qos.ch/translator/Welcome.do] There is also an eclipse console plugin eclipse console plugin [http://logback.qos.ch/consolePlugin.html].

11.1. Web applications

In your web applications remove the following entry from your web.xml

```
<context-param>
  <param-name>log4jConfigLocation</param-name>
  <param-value>/WEB-INF/log4j.properties</param-value>
</context-param>
```

Add the following to the web.xml

```
<listener>
    <listener-class>org.red5.logging.ContextLoggingListener</listener-class>
</listener>

<filter>
    <filter-name>LoggerContextFilter</filter-name>
    <filter-class>org.red5.logging.LoggerContextFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>LoggerContextFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

You should also:

- Remove any "log4j" listeners from the web.xml
- Remove any log4j.properties or log4j.xml files
- Create a logback-myApp.xml where myApp is the name for your webapp and place it on your webapp classpath (WEB-INF/classes or in your application jar within WEB-INF/lib)
- Set your display-name in the web application to match the context name you will be using (Use the example oflaDemo as a guide).
- Ensure that the contextName and jmxConfigurator have the correct context name, this is the name of your web application

Sample webapp logback config file (logback-myApp.xml), not to be confused with the red5 log config file located in /conf

```
<xslthl:directive><?xml version="1.0" encoding="UTF-8"?></xslthl:directive>

<configuration>
    <contextName>myApp</contextName>
    <jmxConfigurator contextName="myApp" />

    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <File>example.log</File>
        <Append>false</Append>
        <BufferedIO>false</BufferedIO>
        <ImmediateFlush>true</ImmediateFlush>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</Pattern>
        </layout>
    </appender>
    <root>
        <level value="DEBUG" />
        <appender-ref ref="FILE" />
    </root>
    <logger name="com.example">
        <level value="DEBUG" />
    </logger>
</configuration>
```

Reminder replace everything that says "myApp" with your application name.

11.2. Imports

When using logback and slf4j, your imports should consist only of the following for a non webapp class:

```
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
```

It is suggested that you use Red5LoggerFactory in-place of LoggerFactory to ensure that your application gets the correct logger.

For loggers inside your webapp imports should be:

```
import org.slf4j.Logger;
import org.red5.logging.Red5LoggerFactory;
```

11.3. Logger Instantiation

For non webapp classes:

To log to a "root" logger, change all your logger instantiation statements to:

```
private static Logger log = Red5LoggerFactory.getLogger(MyClassName.class);
```

Reminder replace "MyClassName" with the name of the class itself.

To log to a "context" logger, change all your logger instantiation statements to:

```
private static Logger log = Red5LoggerFactory.getLogger(MyClassName.class, "myApp");
```

Reminder replace "myApp" with the name of the context; "myApp" would become "oflaDemo" for the oflaDemo application.

Your old instantiations probably resemble this:

```
private static Logger log = Logger.getLogger(MyClassName.class.getName());
```

Your applications logging configuration file must contain the name of your application context in its file name; For instance the "oflaDemo" uses the configuration logback-oflaDemo.xml.

Lastly, as an optimation change your log statements to:

```
log.debug("Here is a log message for an object {}", myobject);
```

You no longer need to concatenate strings when logging, if you need more than one variable do the following:

```
log.debug("Here is a log message with a couple vars {} or {} or {}", new Object[] {object1, myobject,
```

= Further Information =

Ceki Gülcü presentation for Jazoon09 [http://jazoon.com/en/conference/presentationdetails.html?type=sid&detail=6080]

SLF4J Presentation (video) [http://beta.parleys.com/share/parleysshare2.swf?pageId=357]

12.1. Preface

This document describes how to deploy Red5 to Tomcat as web application archive (WAR). The standard Red5 deployment consists of a standalone Java application with an embedded J2EE container (Jetty or Tomcat) running as a system service, whereas the WAR version runs inside of a J2EE container.

12.2. Deployment

The Tomcat war deployer scans the webapps directory for wars periodically. When a war is found that has not yet been deployed, the deployer will expand the war file into a directory based on the filename of the war. A war named myapp.war would be expanded into a directory named myapp; depending upon your installation the full path would look similar to this C:\Tomcat- 6.0.14\webapps\myapp.

Red5 server is packaged into a file named ROOT.war, this filename has a special connotation on most J2EE application servers and is normally the default or root web context. The root web context is responsible for servicing requests which do not contain a path component. A url with a path component looks like http://www.example.com/myapp wheres root web application url would resemble this http://www.example.com/. An additional configuration file the context descriptor, is located in the META-INF directory for each web context. Applications that are not accessed via HTTP, do not require a web / servlet context. The root war file contains nearly everything that is in a standalone server build except for embedded server classes and select configuration files.

12.3. Context descriptors

A Context XML descriptor is a fragment of XML data which contains a valid Context element which would normally be found in the main Tomcat server configuration file (conf/server.xml). For a given host, the Context descriptors are located in $CATALINA_HOME/conf/[enginename]/[hostname]/. Note that while the name of the file is not tied to the webapp name, when the deployer creates descriptors from the context.xml files contained in the war; their names will match the web application name.

Context descriptors allow defining all aspects and configuration parameters of a context, such as naming resources and session manager configuration. It should be noted that the docBase specified in the Context element can refer to either the .WAR or the directory which will be created when the .WAR is expanded or the .WAR itself.

12.4. Red5 Configuration

Configuration of the Red5 server consists of a few context parameters in the web.xml, a default context file, a bean ref file, and a Spring web context file for each application that will utilize Red5 features. Web applications that use only AMF to communicate with Red5 do not require a configuration entry in the servers application context. The application context which is managed via Spring is only available to applications that are contained within the root war; due to the way that the web application classloaders work. In addition, Red5 uses a context counterpart called a Scope which serves as a container for the context, handler, server core instance, and a few other objects. A scope is similar

to the application model in FMS. The initial entry point or startup servlet for Red5 is the WarLoaderServlet and it is configured as a servlet listener in the web.xml as shown below. Functionally this servlet takes the place of the Standalone class in a standard Red5 server

```
<listener>
  <listener-class>org.red5.server.war.WarLoaderServlet</listener-class>
</listener>
```

This listener is responsible for starting and stopping Red5 upon receipt of context initialized and context destroyed container events. The war loader is similar in function to the Spring ContextLoaderListener servlet but is specialized for Red5.

12.4.1. Spring contexts

There are two types of contexts used by Red5, "default" and "web"; there may be only one default context but any number of web contexts.

12.4.2. Default context

The default context is synonymous with the global application context and is responsible for providing objects and resources at the top or global level. Spring beans in this level are configured via the defaultContext.xml and beanRefContext.xml which are located in the ROOT classes directory (ex. C:\Tomcat-6.0.14\webapps\ROOT\WEB-INF\classes). The bean ref file defines the default.context bean which as an instance of org.springframework.context.support.ClassPathXmlApplicationContext. Two other configuration files red5-common.xml and red5-core.xml are used to construct the default context; these files are derived from the standalone configuration files of the same names, the primary difference is that the server embedding sections have been removed.

The default context is referenced in the web.xml via the parentContextKey parameter:

```
<context-param>
  <param-name>parentContextKey</param-name>
  <param-value>default.context</param-value>
</context-param>
```

This parameter is used by the ContextLoader to locate the parent context, which in turn allows the global resources to be located. The context loader is used by the WarLoaderServlet to initialize the web contexts.

The scope counterpart to the global context is the global scope and it is referenced in the web.xml via the globalScope parameter:

```
<context-param>
  <param-name>globalScope</param-name>
  <param-value>default</param-value>
```

```
</context-param>
```

12.4.3. Web context

Web context definitions are specified in Spring configuration files suffixed with -web.xml; If your application is named oflaDemo then its configuration file would be named oflaDemo-web.xml. The Spring web context files should not be confused with J2EE context descriptors as they are only used for red5 web contexts and the later are used by Tomcat. Each web context must have a corresponding configuration file, the configuration files are specified using an ant- style parameter in the web.xml as shown below.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>WEB-INF/classes/*-web.xml</param-value>
</context-param>
```

Context configuration files specify the resources that are used to notify the application about joining / leaving clients and provide the methods that a client can call. Additionally, the configuration files specify the scope hierarchy for these classes.

Every context configuration must contain a minimum of three entries - a context, scope, and handler. The only exception to this rule is the root web application since it does not have a handler application, in this case the global handler is used.

- **Context** - Each context must have a unique name assigned since all the

contexts exist within a single Spring application context. The root web context is named web.context, additional contexts suffix this base name with their web application name; for example oflaDemo would be named web.context.oflaDemo. A context is specified in the web context file as shown below.

```
<bean id="web.context" class="org.red5.server.Context">
  <property name="scopeResolver" ref="red5.scopeResolver" />
  <property name="clientRegistry" ref="global.clientRegistry" />
  <property name="serviceInvoker" ref="global.serviceInvoker" />
  <property name="mappingStrategy" ref="global.mappingStrategy" />
</bean>
```

- **Scope** - Every application needs at least one scope that links the handler

to the context and the server. The scopes can be used to build a tree where clients can connect to every node and share objects inside this scope (like shared objects or live streams). You can consider the scopes as rooms or instances. The root scope has the name web.scope, additional scope names should follow the naming convention specified for contexts. A scope for oflaDemo would be named web.scope.oflaDemo so that it will not conflict with other contexts.

- A scope bean has the following properties:

1. server - This references the server red5.server
2. parent - The parent for this scope is normally global.scope
3. context - Context for this scope, use the web.context for root and

- web.context.oflaDemo for oflaDemo

1. handler - Handler for this scope, which is similar to a main.asc in

- FMS.

1. contextPath - The path to use when connecting to this scope.
2. virtualHosts - A comma separated list of host names or IP addresses this scope listens on. In the war version we do not control the host names, this is accomplished by Tomcat.

The root scope definition looks like this:

```
<bean id="web.scope" class="org.red5.server.WebScope" init-method="register">
  <property name="server" ref="red5.server" />
  <property name="parent" ref="global.scope" />
  <property name="context" ref="web.context" />
  <property name="handler" ref="global.handler" />
  <property name="contextPath" value="/" />
  <property name="virtualHosts" value="*,localhost, localhost:8080" />
</bean>
```

The contextPath is similar to the docBase in the J2EE context file for each web application. Where the docBase is used to locate resources by HTTP, the contextPath is use to find resources via RTMP. Your applications may add additional elements after the configured path to dynamically create extra scopes. The dynamically created scopes all use the same handler but have their own properties, shared objects and live streams.

- **Handler** - Every context needs a handler to provide the methods called by

connecting clients. All handlers are required to implement org.red5.server.api.IScopeHandler, however you may implement additional interfaces for controlling access to shared objects or streams. A sample implementation is provided with Red5 that may be used as your base class: org.red5.server.adapter.ApplicationAdapter. Please refer to the javadoc for this class for additional details. As an example the scope handler for the oflaDemo is shown:

```
<bean id="web.handler.oflaDemo"
class="org.red5.server.webapp.oflaDemo.Application"/>
```

The id attribute is referenced by the oflaDemo scope definition:

```
<bean id="web.scope.oflaDemo" class="org.red5.server.WebScope" init-
  method="register">
  <property name="server" ref="red5.server" />
  <property name="parent" ref="global.scope" />
  <property name="context" ref="web.context.oflaDemo" />
  <property name="handler" ref="web.handler.oflaDemo" />
  <property name="contextPath" value="/oflaDemo" />
  <property name="virtualHosts" value="*,localhost, localhost:8080" />
</bean>
```

If you don't need any special server-side logic, you can use the default application handler provided by Red5:

```
<bean id="web.handler" class="org.red5.server.adapter.ApplicationAdapter" />
```

12.4.4. External applications

An external application refers to a web application that accesses Red5 outside of the ROOT web application. Whether these applications exist within the same JVM instance or not, they may only access Red5 via RTMP or the AMF tunnel servlet. The tunnel servlet is configured in the web.xml for each application that requires AMF communication with Red5, an example is shown below:

```
<servlet>
  <servlet-name>gateway</servlet-name>
  <servlet-class>org.red5.server.net.servlet.AMFTunnelServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>gateway</servlet-name>
  <url-pattern></servlet-mapping>
</servlet-mapping>
```

The tunnel servlet class must be on the classpath of the application under which it is executed. In addition to the tunnel servlet the org.red5.server.net.servlet.ServletUtils class is required along with the following library jars:

```
commons-codec-1.3.jar
commons-httpclient-3.0.1.jar
commons-logging-1.1.jar
log4j-1.2.14.jar
mina-core-1.1.2.jar
```

These jars should be placed in the WEB-INF/lib directory of your application. ex.

```
C:\Tomcat-6.0.14\webapps\myapp\WEB-INF\lib
```

12.5. Creating and deploying your application

In the following section, two applications will be covered. The first will be a web application that communicates with Red5 via AMF or RTMP and has its own handler, referred to as "RemoteApp". The second will consist an SWF that communicates with Red5 via RTMP, this application will be called "LocalApp". Any IDE may be used to create these applications as long as it supports Java; the Eclipse IDE is suggested. SWF files outlined in the examples were created using AS3 in Flex.

12.5.1. Remote application

This example will provide you with the minimum amount of configuration needed for a remote Red5 application. The following resources will be created:

- J2EE web application
- Client SWF
- Red5 handler class
- Spring web context

Steps

1. Create a web application named RemoteApp in your IDE.
2. Obtain a red5.jar, which may be downloaded from http://red5.googlecode.com/files/red5.jar or built from source with the command "ant jar". This library is needed if you extend the ApplicationAdapter for your scope handler.
3. Obtain the red5-remoting.jar, this may be accomplished by building yourself from the command line with "ant remotejar" or by downloading it from http://red5.googlecode.com/files/red5-remoting.jar. This library provides the AMF tunnel servlet.
4. Place the library jars in your project library directory and add them to your build classpath.
5. Compile the Java and Flex source.
6. Create a directory named RemoteApp in the Tomcat webapps directory. ex. C:\Tomcat-6.0.14\webapps\RemoteApp
7. Copy the contents of the web directory to the RemoteApp directory.
8. From the bin directory copy the RemoteApp.swf to the webapps\RemoteApp directory.
9. Copy the lib directory and its contents to the WEB-INF, excluding the red5.jar file.
10. Copy the whole example directory and the RemoteApp-web.xml file from the bin directory to the classes directory under ROOT. ex. C:\Tomcat- 6.0.14\webapps\ROOT\WEB-INF\classes

11. Restart tomcat
12. Open your browser and go to: http://localhost:8080/RemoteApp/RemoteApp.html
13. Click on the RTMP or HTTP connect buttons. For a successful test you should see a server response of "Hello World".

12.5.2. Local application

A simple application that resides entirely within the ROOT web application. This example consists of a Spring web context, handler class, and a client SWF.

'Steps '

1. Create a web application named LocalApp in your IDE.
2. Obtain a red5.jar, which may be downloaded from http://red5.googlecode.com/files/red5.jar or built from source with the command "ant jar". This library is needed if you extend the ApplicationAdapter for your scope handler.
3. Place the library jar in your project library directory and add it to your build classpath.
4. Compile the Java and Flex source.
5. Copy the LocalApp.html and LocalApp.swf from the bin directory to the ROOT directory. ex. C:\Tomcat-6.0.14\webapps\ROOT
6. Copy the whole example directory and the LocalApp-web.xml file from the bin directory to the classes directory under ROOT. ex. C:\Tomcat- 6.0.14\webapps\ROOT\WEB-INF\classes
7. Restart tomcat
8. Open your browser and go to: http://localhost:8080/LocalApp.html
9. Click on the connect button. For a successful test you should see a server response of "Hello World".

12.5.3. Example Source

The example application source is available in Subversion at https://red5.googlecode.com/svn/java/example/trunk/

12.6. Additional web configuration

Log4j - The path to the logging configuration file and the Spring logging startup servlet are shown below. These entries should precede the war loader servlet entry so that logging is initialized prior to Red5 startup.

```
<context-param>
  <param-name>log4jConfigLocation</param-name>
  <param-value>/WEB-INF/log4j.properties</param-value>
```

```
</context-param>
<listener>
  <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
</listener>
```

AMF gateway - This servlet provides communication with server applications using AMF.

```
<servlet>
  <servlet-name>gateway</servlet-name>
  <servlet-class>org.red5.server.net.servlet.AMFGatewayServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>gateway</servlet-name>
  <url-pattern>/gateway</url-pattern>
</servlet-mapping>
```

RTMPT - This servlet implements an RTMP tunnel via HTTP, this is normally used to bypass firewall issues.

```
<servlet>
  <servlet-name>rtmpt</servlet-name>
  <servlet-class>org.red5.server.net.rtmpt.RTMPTServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>rtmpt</servlet-name>
  <url-pattern>/open/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>rtmpt</servlet-name>
  <url-pattern>/idle/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>rtmpt</servlet-name>
  <url-pattern>/send/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>rtmpt</servlet-name>
  <url-pattern>/close/*</url-pattern>
</servlet-mapping>
```

Security - The following entries are used to prevent retrieval of sensitive information.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Forbidden</web-resource-name>
    <url-pattern>/WEB-INF/*</url-pattern>
  </web-resource-collection>
  <auth-constraint />
</security-constraint>
<security-constraint>
  <web-resource-collection>
```

```
    <web-resource-name>Forbidden</web-resource-name>
    <url-pattern>/persistence/*</url-pattern>
  </web-resource-collection>
  <auth-constraint />
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Forbidden</web-resource-name>
    <url-pattern>/streams/*</url-pattern>
  </web-resource-collection>
  <auth-constraint />
</security-constraint>
```

12.7. Troubleshooting

If you have problems with deployment or if your application does not start, follow these steps prior to posting a bug. Directory examples use a typical windows based path structure.

1. Stop the Tomcat server
2. Locate your Tomcat installation directory

```
C:\Program Files\Apache\Tomcat
```

1. Delete the "work" directory

```
C:\Program Files\Apache\Tomcat\work
```

1. Delete the "Catalina" directory from the "conf" directory

```
C:\Program Files\Apache\Tomcat\conf\Catalina
```

1. Delete the expanded war directories, if they exist

```
C:\Program Files\Apache\Tomcat\webapps\ROOT
C:\Program Files\Apache\Tomcat\webapps\echo
C:\Program Files\Apache\Tomcat\webapps\SOSample
```

1. Ensure your WAR files are in the webapps directory

```
C:\Program Files\Apache\Tomcat\webapps\ROOT.war
C:\Program Files\Apache\Tomcat\webapps\echo.war
C:\Program Files\Apache\Tomcat\webapps\SOSample.war
```

1. Restart Tomcat

If you still experience problems, gather the following information and post an issue on Trac after you do a quick search to see if others have experienced the same problem.

1. Java version
2. Tomcat version
3. Operating system
4. Red5 version (0.8, Trunk, Revision 2283, etc...)

12.8. Definitions

AMF::

A binary format based loosely on the Simple Object Access Protocol (SOAP). It is used primarily to exchange data between an Adobe Flash application and a database, using a Remote Procedure Call. Each AMF message contains a body which holds the error or response, which will be expressed as an ActionScript Object.

Ant::

Software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, and is best suited to building Java projects.

AS3::

A scripting language based on ECMAScript, used primarily for the development of websites and software using the Adobe Flash Player platform.

Flex::

Software development kit and an IDE for a group of technologies initially released in March of 2004 by Macromedia to support the development and deployment of cross platform, rich Internet applications based on their proprietary Macromedia Flash platform.

RTMP::

Real Time Messaging Protocol (RTMP) is a proprietary protocol developed by Adobe Systems that is primarily used with Adobe Flash Media Server to stream audio, video, and data over the internet to the Adobe Flash Player client. RTMP can be used for Remote Procedure Calls. RTMP maintains a persistent connection with an endpoint and allows real-time communication. Other RPC services are made asynchronously with a single client/server request/response model, so real-time communication is not necessary.

RTMPT::

RTMP using HTTP tunneling.

SWF::

Proprietary vector graphics file format produced by the Flash software from Adobe. Intended to be small enough for publication on the web, SWF files can contain animations or applets of varying degrees of interactivity and function. SWF is also sometimes used for creating animated display graphics and menus for DVD movies, and television commercials.

Tomcat::

A web container, or application server developed at the Apache Software Foundation (ASF). Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own internal HTTP server.

12.9. Bibliography

- Red5 - http://red5.org
- Apache Tomcat - http://tomcat.apache.org
- Wikipedia - http://en.wikipedia.org

This document describes how applications can stream ondemand videos (VOD) from or record to custom directories other than the default streams folder inside the webapp.

13.1. Filename generator service

Red5 uses a concept called scope services for functionality that is provided for a certain scope. One of these scope services is IStreamFilenameGenerator http://dl.fancycode.com/red5/api/org/red5/server/api/stream/IStreamFilenameGenerator.html that generates filenames for VOD streams that should be played or recorded.

13.2. Custom generator

To generate filename in different folders, a new filename generator must be implemented:

```
import org.red5.server.api.IScope;
import org.red5.server.api.stream.IStreamFilenameGenerator;
public class CustomFilenameGenerator implements IStreamFilenameGenerator {
 /** Path that will store recorded videos. */
 public String recordPath = "recordedStreams/";
 /** Path that contains VOD streams. */
 public String playbackPath = "videoStreams/";
 /** Set if the path is absolute or relative */
 public boolean resolvesAbsolutePath = false;
 public String generateFilename(IScope scope, String name, GenerationType type) {
  // Generate filename without an extension.
  return generateFilename(scope, name, null, type);
 }
 public String generateFilename(IScope scope, String name, String extension, GenerationType type) {
  String filename;
  if (type == GenerationType.RECORD)
   filename = recordPath + name;
  else
   filename = playbackPath + name;

  if (extension != null)
   // Add extension
   filename += extension;

  return filename;
 }

 public boolean resolvesToAbsolutePath()
    {
     return resolvesAbsolutePath;
    }
}
```

The above class will generate filenames for recorded streams like recordedStreams/red5RecordDemo1234.flv and use the directory videoStreams as source for all VOD streams.

13.3. Activate custom generator

In the next step, the custom generator must be activate in the configuration files for the desired application.

Add the following definition to yourApp/WEB-INF/red5-web.xml:

```
<bean id="streamFilenameGenerator"
    class="path.to.your.CustomFilenameGenerator" />
```

This will use the class defined above to generate stream filenames.

13.4. Change paths through configuration

While the class described here works as expected, it's a bit unhandy to change the paths inside the code as every change requires recompilation of the class.

Therefore you can pass parameters to the bean defined in the previous step to specify the paths to use inside the configuration file.

Add three methods to your class that will be executed while the configuration file is parsed:

```
public void setRecordPath(String path) {
 recordPath = path;
}
public void setPlaybackPath(String path) {
 playbackPath = path;
}
public void setAbsolutePath(Boolean absolute) {
 resolvesAbsolutePath = absolute;
}
```

Now you can set the paths inside the bean definition:

```
<bean id="streamFilenameGenerator"
   class="path.to.your.CustomFilenameGenerator">
   <property name="recordPath" value="recordedStreams/" />
   <property name="playbackPath" value="videoStreams/" />
   <property name="absolutePath" value="false" />
</bean>
<bean id="streamFilenameGenerator"
   class="path.to.your.CustomFilenameGenerator">
   <property name="recordPath" value="/path/to/recordedStreams/" />
   <property name="playbackPath" value="/path/to/videoStreams/" />
   <property name="absolutePath" value="true" />
</bean>
```

You can also move the paths to the yourApp/WEB-INF/red5-web.properties file and use parameters to access them:

```
<bean id="streamFilenameGenerator"
   class="path.to.your.CustomFilenameGenerator">
   <property name="recordPath" value="${recordPath}" />
   <property name="playbackPath" value="${playbackPath}" />
   <property name="absolutePath" value="${absolutePath}" />
</bean>
```

In that case you will have to add the following lines to your properties file:

red5-web.properties -

```
recordPath=recordedStreams/
playbackPath=videoStreams/
absolutePath=false
recordPath=/path/to/recordedStreams/
playbackPath=/path/to/videoStreams/
absolutePath=true
```

This document describes the Red5 API that was introduced in version 0.6 to protect access to streams and/or shared objects similar to what the properties Client.readAccess and Client.writeAccess provide in the Macromedia Flash Communication Server / Flash Media Server 2.

14.1. Stream Security

Read (playback) and write (publishing/recording) access to streams is protected separately in Red5.

14.1.1. Stream playback security

For applications that want to limit the playback of streams per user or only want to provide access to streams with a given name, the interface IStreamPlaybackSecurity http://dl.fancycode.com/red5/api/org/red5/server/api/stream/IStreamPlaybackSecurity.html is available in Red5.

It can be implemented by any object and registered in the ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html . An arbitrary number of stream security handlers is supported per application. If at least one of the handlers denies access to the stream, the client receives an error NetStream.Failed with a description field giving a corresponding error message.

An example handler that only allows access to streams that have a name starting with liveStream is described below:

```
import org.red5.server.api.IScope;
import org.red5.server.api.stream.IStreamPlaybackSecurity;

public class NamePlaybackSecurity implements IStreamPlaybackSecurity {

    public boolean isPlaybackAllowed(IScope scope, String name, int start,
     int length, boolean flushPlaylist) {
         if (!name.startswith("liveStream")) {
            return false;
         } else {
            return true;
         }
     };
}
```

To register this handler in the application, add the following code in the appStart method:

```
registerStreamPlaybackSecurity(new NamePlaybackSecurity());
```

Red5 includes a sample security handler that denies all access to streams (DenyAllStreamAccess http://dl.fancycode.com/red5/api/org/red5/server/api/stream/support/DenyAllStreamAccess.html).

14.1.2. Stream publishing security

In most applications that allow the user to publish and/or record streams, this access must be limited to prevent the server from being misused. Therefore, Red5 provides the interface IStreamPublishSecurity http://dl.fancycode.com/red5/api/org/red5/server/api/stream/IStreamPublishSecurity.html to deny publishing of certain streams.

Similar to IStreamPlaybackSecurity http://dl.fancycode.com/red5/api/org/red5/server/api/stream/IStreamPlaybackSecurity.html, it can be implemented by any object and registered in the ApplicationAdapter http://dl.fancycode.com/red5/api/org/red5/server/adapter/ApplicationAdapter.html. If one of the registered handlers denies access, the client receives an error NetStream.Failed with a description field giving a corresponding error message.

An example handler that only allows authenticated connections to publish a live stream starting with liveStream and deny all other access is described below:

15.1. I. Select a scripting implementation

Level: Beginner

Red5 includes interpreters for the following scripting languages:

- Javascript - version 1.6 (Mozilla Rhino version 1.6 R7)
- JRuby - version 1.0.1 (Ruby version 1.8.5)
- Jython - version 2.2 (Python version 2.1)
- Groovy - version 1.0
- Beanshell - version 2.0b4

Future versions may include:

- JudoScript
- Scala
- PHP (This one is non-trivial, I may just provide a bridge)
- Actionscript (Maybe SSAS)

The scripting implementation classes are pre-specified in the following locations depending upon your Java version:

```
Java5 - js-engine.jar, jython-engine.jar, groovy-engine.jar
Java6 - resources.jar
```

File location: /META-INF/services/javax.script.ScriptEngineFactory

It is most likely that the classes read from the jdk or jre will be prefered over any specified elsewhere.

15.2. II. Configuring Spring

Level: Intermediate

Step one is to locate your web applications red5-web.xml file. Within the xml config file the web.scope bean definition must supply a web.handler, this handler is your Red5 application (An application must extend the org.red5.server.adapter.ApplicationAdapter class).

The application provides access to the Red5 server and any service instances that are created. The service instances and the application itself may be scripted. Bean definitions in Spring config files may not have the same id, here are some web handler definition examples:

- Java class implementation

```
<bean id="web.handler" class="org.red5.server.webapp.oflaDemo.MultiThreadedApplicationAdapter" />
```

- Javascript implementation

```
<bean id="web.handler" class="org.red5.server.script.rhino.RhinoScriptFactory">
  <constructor-arg index="0" value="classpath:applications/main.js"/>
  <constructor-arg index="1">
    <list>
     <value>org.red5.server.api.IScopeHandler</value>
     <value>org.red5.server.adapter.IApplication</value>
    </list>
</constructor-arg>
<constructor-arg index="2">
 <value>org.red5.server.adapter.ApplicationAdapter</value>
</constructor-arg>
</bean>
```

- Ruby implementation

```
<bean id="web.handler" class="org.springframework.scripting.jruby.JRubyScriptFactory">
<constructor-arg index="0" value="classpath:applications/main.rb"/>
<constructor-arg index="1">
 <list>
  <value>org.red5.server.api.IScopeHandler</value>
  <value>org.red5.server.adapter.IApplication</value>
 </list>
</constructor-arg>
</bean>
```

- Groovy implementation

```
<bean id="web.handler" class="org.red5.server.script.groovy.GroovyScriptFactory">
<constructor-arg index="0" value="classpath:applications/main.groovy"/>
<constructor-arg index="1">
 <list>
  <value>org.red5.server.api.IScopeHandler</value>
  <value>org.red5.server.adapter.IApplication</value>
 </list>
</constructor-arg>
</bean>
```

- Python implementation

```
Red5 Open Source
Flash Server (0.7.1) 51
<bean id="web.handler" class="org.red5.server.script.jython.JythonScriptFactory">
 <constructor-arg index="0" value="classpath:applications/main.py"/>
 <constructor-arg index="1">
  <list>
```

```
      <value>org.red5.server.api.IScopeHandler</value>
      <value>org.red5.server.adapter.IApplication</value>
Scripting Implementations
  </list>
 </constructor-arg>
        <constructor-arg index="2">
  <list>
            <value>One</value>
            <value>2</value>
            <value>III</value>
        </list>
    </constructor-arg>
</bean>
```

In general the configuration using scripted classes is defined using the constructor arguments (see interpreter section) in the following order:

- Argument 1 - Location of the script source file
- Argument 2 - Java interfaces implemented by the script.

The interfaces for the code which extends an Application are basically boilerplate as seen in the examples above; You do not have to use those interfaces in all your script definitions.

- Argument 3 - Java classes extended by the script.

The extended class is not always necessary, it depends upon the scripting engine implementation.

The example location starts with classpath:applications which in physical disk terms for the "oflaDemo" application equates to webapps/oflaDemo/WEB-INF/applications

15.3. III. Creating an application script

15.3.1. 1. Application adapter

Scripting an application adapter is more difficult in some languages than it is in others, because of this I present the Ruby example which works really well and is easy to write and integrate. The application services are easily written in any of the supported languages, but they require a Java interface at a minimum.

i. JRuby application adapter implementation

```
# JRuby
require 'java'
module RedFive
   include_package "org.red5.server.api"
   include_package "org.red5.server.api.stream"
   include_package "org.red5.server.api.stream.support"
   include_package "org.red5.server.adapter"
   include_package "org.red5.server.stream"
end
#
# application.rb - a translation into Ruby of the ofla demo application, a red5 example.
```

```
#
# @author Paul Gregoire
#
class Application &lt; RedFive::ApplicationAdapter
    attr_reader :appScope, :serverStream
        attr_writer :appScope, :serverStream
        def initialize
          #call super to init the superclass, in this case a Java class
          super
          puts "Initializing ruby application"
        end
        def appStart(app)
        puts "Ruby appStart"
                @appScope = app
                return true
        end
        def appConnect(conn, params)
                puts "Ruby appConnect"
                measureBandwidth(conn)
                puts "Ruby appConnect 2"
                if conn.instance_of?(RedFive::IStreamCapableConnection)
                    puts "Got stream capable connection"
                        sbc = RedFive::SimpleBandwidthConfigure.new
                        sbc.setMaxBurst(8388608)
                        sbc.setBurst(8388608)
                        sbc.setOverallBandwidth(8388608)
                        conn.setBandwidthConfigure(sbc)
                end
                return super
        end
        def appDisconnect(conn)
                puts "Ruby appDisconnect"
                if appScope == conn.getScope &amp;&amp; @serverStream != nil
                        @serverStream.close
                end
                super
        end
        def toString
        return "Ruby toString"
        end
    def setScriptContext(scriptContext)
          puts "Ruby application setScriptContext"
    end
    def method_missing(m, *args)
      super unless @value.respond_to?(m)
      return @value.send(m, *args)
    end
end
```

15.3.2. 2. Application services

Here is an example of a Java interface (Yes, the methods are supposed to be empty) which is used in the examples to provide a template for applications which will gather a list of files and return them as a "Map" (key-value pairs) to the caller.

i. Simple Java interface for implementation by scripts

```
package org.red5.server.webapp.oflaDemo;

import java.util.Map;
public interface IDemoService {
        /**
     * Getter for property 'listOfAvailableFLVs'.
     *
     * @return Value for property 'listOfAvailableFLVs'.
     */
    public Map getListOfAvailableFLVs();
    public Map getListOfAvailableFLVs(String string);
}
```

ii. Spring bean definition for a script implementation of the interface

```
<bean id="demoService.service" class="org.springframework.scripting.jruby.JRubyScriptFactory">
   <constructor-arg index="0" value="classpath:applications/demoservice.rb"/>
   <constructor-arg index="1">
      <list>
         <value>org.red5.server.webapp.oflaDemo.IDemoService</value>
      </list>
   </constructor-arg>
</bean>
```

iii.JRuby script implementing the interface

```
# JRuby - style
require 'java'
module RedFive
   include_package "org.springframework.core.io"
   include_package "org.red5.server.webapp.oflaDemo"
end
include_class "org.red5.server.api.Red5"
include_class "java.util.HashMap"
#
# demoservice.rb - a translation into Ruby of the ofla demo application, a red5 example.
#
# @author Paul Gregoire
```

```
#
class DemoService &lt; RedFive::DemoServiceImpl
   attr_reader :filesMap
   attr_writer :filesMap
      def initialize
         puts "Initializing ruby demoservice"
         super
         @filesMap = HashMap.new
      end
 def getListOfAvailableFLVs
            puts "Getting the FLV files"
            begin
          dirname = File.expand_path('webapps/oflaDemo/streams').to_s
                  Dir.open(dirname).entries.grep(/\.flv$/) do |dir|
                     dir.each do |flvName|
                     fileInfo = HashMap.new
                     stats = File.stat(dirname+'/'+flvName)
                     fileInfo["name"] = flvName
                     fileInfo["lastModified"] = stats.mtime
                     fileInfo["size"] = stats.size || 0
                @filesMap[flvName] = fileInfo
                print 'FLV Name:', flvName
                print 'Last modified date:', stats.mtime
                print 'Size:', stats.size || 0
                print '-------'
             end
          end
             rescue Exception =&gt; ex
                   puts "Error in getListOfAvailableFLVs #{errorType} \n"
                   puts "Exception: #{ex} \n"
                   puts caller.join("\n");
             end
             return filesMap
      end
      def formatDate(date)
            return date.strftime("%d/%m/%Y %I:%M:%S")
      end
   def method_missing(m, *args)
     super unless @value.respond_to?(m)
     return @value.send(m, *args)
   end
end
```

iv.Java application implementing the interface, upon which the Ruby code was based (This code is NOT needed when using the script)

```
package org.red5.server.webapp.oflaDemo;
import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.red5.server.api.IScope;
import org.red5.server.api.Red5;
import org.springframework.core.io.Resource;
public class DemoService {
        protected static Log log = LogFactory.getLog(DemoService.class.getName());

 /**
     * Getter for property 'listOfAvailableFLVs'.
Scripting Implementations
     *
     * @return Value for property 'listOfAvailableFLVs'.
     */
    public Map getListOfAvailableFLVs() {
                IScope scope = Red5.getConnectionLocal().getScope();
                Map&lt;String, Map&gt; filesMap = new HashMap&lt;String, Map&gt;();
                Map&lt;String, Object&gt; fileInfo;
                try {
                        log.debug("getting the FLV files");
                        Resource[] flvs = scope.getResources("streams/*.flv");
                        if (flvs != null) {
                                for (Resource flv : flvs) {
                                        File file = flv.getFile();
                                        Date lastModifiedDate = new Date(file.lastModified());
                                        String lastModified = formatDate(lastModifiedDate);
                                        String flvName = flv.getFile().getName();
                                        String flvBytes = Long.toString(file.length());
                                        if (log.isDebugEnabled()) {
                                                log.debug("flvName: " + flvName);
                                                log.debug("lastModified date: " + lastModified);
                                                log.debug("flvBytes: " + flvBytes);
                                                log.debug("-------");
                                        }
                                        fileInfo = new HashMap&lt;String, Object&gt;();
                                        fileInfo.put("name", flvName);
                                        fileInfo.put("lastModified", lastModified);
                                        fileInfo.put("size", flvBytes);
                                        filesMap.put(flvName, fileInfo);
                                }
}
                        Resource[] mp3s = scope.getResources("streams/*.mp3");
                        if (mp3s != null) {
                                for (Resource mp3 : mp3s) {
                                        File file = mp3.getFile();
                                        Date lastModifiedDate = new Date(file.lastModified());
                                        String lastModified = formatDate(lastModifiedDate);
                                        String flvName = mp3.getFile().getName();
                                        String flvBytes = Long.toString(file.length());
                                        if (log.isDebugEnabled()) {
                                                log.debug("flvName: " + flvName);
                                                log.debug("lastModified date: " + lastModified);
                                                log.debug("flvBytes: " + flvBytes);
                                                log.debug("-------");
                                        }
                                        fileInfo = new HashMap&lt;String, Object&gt;();
                                        fileInfo.put("name", flvName);
                                        fileInfo.put("lastModified", lastModified);
                                        fileInfo.put("size", flvBytes);
```

```
                                        filesMap.put(flvName, fileInfo);
                                }
                        }
                } catch (IOException e) {
                        log.error(e);
                }
                return filesMap;
        }

        private String formatDate(Date date) {
                SimpleDateFormat formatter;
                String pattern = "dd/MM/yy H:mm:ss";
                Locale locale = new Locale("en", "US");
                formatter = new SimpleDateFormat(pattern, locale);
                return formatter.format(date);
        }
}
```

v. Flex AS3 method calling the service

```
[Bindable]
public var videoList:ArrayCollection;
public function catchVideos():void{
        // call server-side method
        // create a responder and set it to getMediaList
        var nc_responder:Responder = new Responder(getMediaList, null);
        // call the server side method to get list of FLV's
        nc.call("demoService.getListOfAvailableFLVs", nc_responder);
}
public function getMediaList(list:Object):void{
        // this is the result of the server side getListOfAvailableFLVs
        var mediaList:Array = new Array();
        for(var items:String in list){
            mediaList.push({label:items, size:list[items].size, dateModified:list[items].lastModifi
        }
        // videoList is bindable and the datagrid is set to use this for it's dataprovider
        // wrap it in an ArrayCollection first
        videoList = new ArrayCollection(mediaList);
}
```

15.4. Creating your own interpreter

Level: Advanced

Lets just open this up by saying that I attempted to build an interpreter for PHP this last weekend 02/2007 and it was a real pain; after four hours I had to give up. So what I learned from this is that you must first identify scripting languages which operate as applications, not as http request processors. Heres a test: Can X language be compiled into an executable or be run on the command-line? If yes then it should be trivial to integrate.

15.5. Links with scripting information

Spring scripting

- http://static.springframework.org/spring/docs/2.0.x/reference/dynamic-language.html
- http://rhinoinspring.sourceforge.net/

Java scripting

- http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/
- http://blogs.sun.com/sundararajan/
- https://scripting.dev.java.net/
- http://today.java.net/pub/a/today/2006/04/11/scripting-for-java-platform.html
- http://www.javaworld.com/javaworld/jw-03-2005/jw-0314-scripting_p.html
- http://www.oreillynet.com/onjava/blog/2004/01/java_scripting_half_the_size_h.html
- http://www.robert-tolksdorf.de/vmlanguages.html

Javascript

- http://www.mozilla.org/rhino/
- http://www.mozilla.org/rhino/ScriptingJava.html

Ruby

- http://jruby.codehaus.org/

BeanShell

- http://www.beanshell.org/

Python

- http://www.jython.org/Project/
- http://www.onjava.com/pub/a/onjava/2002/03/27/jython.html
- http://jepp.sourceforge.net/
- http://jpe.sourceforge.net/
- http://jpype.sourceforge.net/

Groovy

- http://groovy.codehaus.org/

In Red5 0.7 the Ant build.xml file contains a build target that creates a 'cluster' folder containing the same setup as described below. Use 'ant dist-cluster' to create the Red5 clustering setup.

Limitations

As of now, the current trunk only supports the clustering configuration for multiple Edges with one Origin. The Edge server only accepts RTMP connection.

16.1. Server Configuration

16.1.1. Configuration Files

There are several configuration files added to support Edge/Origin configuration.

red5-edge.xml, red5-edge-core.xml - used for edge spring bean configuration. They are under conf/.

red5-origin.xml, red5-origin-core.xml - used for origin spring bean configuration. They are under conf/.

16.2. Configure Edge Server

You don't need to deploy your application on Edges.

We strongly recommend you to deploy Edge on a different server from Origin. But it should be OK to deploy the Edge on the same server as Origin.

16.2.1. Edge on a different Server from Origin

Update the configuration of bean "mrtmpClient" in red5-edge-core.xml to point to Origin server:

```
<bean id="mrtmpClient"
    class="org.red5.server.net.mrtmp.MRTMPClient" init-method="start" >
    <property name="ioHandler" ref="mrtmpHandler" />
    <property name="server" value="${mrtmp.host}" />
    <property name="port" value="${mrtmp.port}" />
</bean>
```

Replace red5.xml with red5-edge.xml. Start the server by

```
$ ./red5.sh
```

or

```
$ java -jar red5.jar
```

16.2.2. Edge on the same Server as Origin

You don't need to change red5.xml. Copy red5-edge.xml to $(RED5_ROOT) from $(RED5_ROOT)/conf. Start the server by

```
$ java -jar red5.jar red5-edge.xml
```

or update red5.sh to add a parameter "red5-edge.xml", then

```
$ ./red5.sh
```

16.3. Configure Origin Server

Deploy your application to webapps/. Make sure your 9035 port is not blocked by firewall. The port will be used by Edges to connection Origin.

Update red5.xml with red5-origin.xml. Start the server by

```
$ ./red5.sh
```

or

```
$ java -jar red5.jar
```

16.4. Use Your Appliation

Your RTMP can go through Edges now. Your RTMPT and HTTP can go through Origin as normal.

17.1. JMX Classes

Red5's implementation consists of the following classes and various other MBeans:

org.red5.server.jmx.JMXFactory - Provides access to the platform MBeanServer as well as registration, unregistration, and creation of new MBean instances. Creation and registration is performed using StandardMBean wrappers.

org.red5.server.jmx.JMXAgent - Provides the HTML adapter and registration of MBeans.

org.red5.server.jmx.JMXUtil - Helper methods for working with ObjectName or MBean instances.

17.2. Spring configuration

The Spring configuration for the JMX implementation allows you to configure the "domain" for MBean registration and listener port for the HTML adaptor. The default entries are shown below.

```
<!-- JMX server -->
<!-- JMX server -->
 <bean id="jmxFactory" class="org.red5.server.jmx.JMXFactory">
  <property name="domain" value="org.red5.server"/>
 </bean>
 <bean id="jmxAgent" class="org.red5.server.jmx.JMXAgent" init-method="init">
  <!-- The RMI adapter allows remote connections to the MBeanServer -->
  <property name="enableRmiAdapter" value="true"/>
  <property name="rmiAdapterPort" value="${jmx.rmi.port.registry}"/>
  <property name="rmiAdapterRemotePort" value="${jmx.rmi.port.remoteobjects}"/>
  <property name="rmiAdapterHost" value="${jmx.rmi.host}"/>
  <!-- SSL
   To use jmx with ssl you must also supply the location of the keystore and its password
   when starting the server with the following JVM options:
    -Djavax.net.ssl.keyStore=keystore
        -Djavax.net.ssl.keyStorePassword=password
  -->
  <property name="enableSsl" value="${jmx.rmi.ssl}"/>
  <!-- Starts a registry if it doesnt exist -->
  <property name="startRegistry" value="true"/>
  <!-- Authentication -->
  <property name="remoteAccessProperties" value="${red5.config_root}/access.properties"/>
  <property name="remotePasswordProperties" value="${red5.config_root}/password.properties"/>
  <property name="remoteSSLKeystore" value="${red5.config_root}/keystore.jmx"/>
  <property name="remoteSSLKeystorePass" value="${rtmps.keystorepass}"/>
  <!-- The HTML adapter allows connections to the MBeanServer via a web browser -->
  <property name="enableHtmlAdapter" value="${jmx.http}"/>
  <property name="htmlAdapterPort" value="${jmx.http.port}"/>
  <!-- Mina offers its own Mbeans so you may integrate them here -->
  <property name="enableMinaMonitor" value="true"/>
 </bean>
```

The config settings for the jmxAgent bean is located in the red5.properties, these are:

red5.properties -

```
 # JMX
jmx.rmi.port.registry=9999
jmx.rmi.port.remoteobjects=
jmx.rmi.host=0.0.0.0
jmx.rmi.ssl=false
jmx.http=false
jmx.http.port=8082
```

1. jmx.rmi.port.registry - The RMI registry port. The RMI adapter may only be used if an RMI registry is running. The RMI registry is enabled by default.

1. jmx.rmi.port.remoteobjects - The RMI remote objects export port to specify for access through firewalls. The default port is generated from the RMI stack.

1. jmx.rmi.host - For RMI remote access specify the host to bind to usually the public address.

1. jmx.rmi.ssl - Enable RMI / JMX SSL. SSL is off by default.

2. jmx.http - Enable HTTP RMI adapter. The HTML adapter is disabled by default, but it allows easy management of MBeans from a web browser.

17.3. RMI Authentication

RMI authentication is configured and enabled by default. This is to secure the RMI connection from anonymous clients. The bean properties remoteAccessProperties and remotePasswordProperties set the JMX access and password config files. The access.properties and password.properties config files define the JMX user rights and clear text password. access.properties contains a user and group rights config

access.properties -

```
red5user readwrite
```

Where red5user is the JMX username and readwrite is the rights which is usually left as default. password.properties contains the JMX user and password

password.properties -

```
red5user changeme
```

Where red5user is the JMX username and changeme is the JMX password.

Tip

It is advisable to change the default login, aswell as configure with SSL enabled as the login is cleartext.

17.4. JMX / RMI / SSL

When RMI is enabled with SSL, the bean properties remoteSSLKeystore and remoteSSLKeystorePass are required to load the SSL keystore and the keystore password for the SSL request. The default keystore loaded is the conf/keystore.jmx file which can also share the keystore required for RTMPS connections. The java properties javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword are transparently set. To generate the keystore / and truststore for client / server connections run from the source

```
ant truststore
```

This will generate a keystore.jmx, red5server.cer and truststore.jmx certificate.

17.5. jConsole / JMX Client

JConsole is a utility that ships with the JRE (since 1.5), it allows you to manage local and remote JMX implementations. To enable introspection you must add the following VM parameter to your startup:

```
-Dcom.sun.management.jmxremote
```

17.5.1. Local Management

After the parameter is set and the application initialized you can start jConsole at the command line by typing:

```
$ jconsole
```

A Swing application will appear and you must select the implementation (agent) you wish to manage, for local simply select "org.red5.server.Standalone".

17.5.2. Remote Management

For remote connections with jconsole / JMX clients the command is

```
$ jconsole service:jmx:rmi://host:port/jndi/rmi://host:port/red5
```

17.5.3. SSL Remote Management

For remote ssl connections with jconsole / JMX clients the client is required to load the truststore certificate generated previouslly.

The command for setting the truststore properties

```
$ jconsole -J-Djavax.net.ssl.trustStore=truststore.jmx \
-J-Djavax.net.ssl.trustStorePassword=password \
service:jmx:rmi://host:port/jndi/rmi://host:port/red5
```

17.6. Links

- http://www.onjava.com/pub/a/onjava/2004/09/29/tigerjmx.html?page=1
- http://java.sun.com/developer/JDCTechTips/2005/tt0315.html#2

The Red5 demo applications are available for downloaded on demand using the installer application located at http://localhost:5080/installer.

18.1. Getting Red5 Demo Applications Server-Side and Client-Side Source

1. With your favourite SVN client check out the source code from svn at this address http://red5.googlecode.com/svn/java/example/trunk/ or https://red5.googlecode.com/svn/java/example/trunk/ if you have a google code login.

1. With your favourite SVN client check out the source code from svn at this address http://red5.googlecode.com/svn/flash/trunk/ or https://red5.googlecode.com/svn/flash/trunk/ if you have a google code login.

18.1.1. List Of Available Demo Applications (Server Side)

- SOSample - A simple shared ball demo that makes use of Shared Objects.
- admin - The Red5 administration panel.
- echo - A test application that runs RTMP/AMF datatype tests.
- oflaDemo - Simple video player as shown on the Online Open Source Flash conference.
- bwcheck - Demo application that detects the client bandwidth.
- fitcDemo - Video conference with chat.

18.1.2. List Of Available Demo Applications (Client Side)

- admin - The admin panel client application
- bwcheck - Demo to interface with the bandwidth check application, tests both download and upload rates.
- echo - Simple echo test AMF client
- loadtest - Simple loading testing tool, requesting a file multiple times.
- port-tester - Open port tester application.
- publisher - Simple broadcaster application

18.2. Environment Build Setup

To build the demo applications and add WAR snapshots to the subversion repository, the ant environment requires a SvnAnt task library added to the ant common library directory:

1. Go here: http://subclipse.tigris.org/svnant.html

2. Download the latest SvnAnt

ex: http://subclipse.tigris.org/files/documents/906/43359/svnant-1.2.0-RC1.zip

1. Unzip the archive and place the jar files in your Ant lib directory

```
C:\dev\ant\lib
```

4. Using your svn client or subclipse svn client in eclipse checkout or update the snapshots repository https://red5.googlecode.com/svn/snapshots. It will keep the registry.xml file up to date for modifying later.

1. Add these variables to a build.properties file into user home directory

```
svn.url=http://red5.googlecode.com/svn/snapshots/
svn.login=youruser
svn.password=the google code password
snapshot.path=/www/red5_snapshots/
Where snapshot.path is the path to the checked out snapshots directory.
```

18.3. Building The Demo Application

To build the application and upload the created WAR file to the snapshots repository run the following ant target.

```
$ ant upload-snapshot
```

18.4. Updating The Applications Registry

Once the updated WAR has been uploaded to the snapshots repository, the registry.xml file requires to be updated so the demo applications installer will collect the update.

1. Locate in the console output after uploading snapshot something like Destination: /www/red5_snapshots/admin- r3197 [nullchangeset/3197]-java6.war the file of the new war will be admin- r3197 [nullchangeset/3197]-java6.war.

2. Edit the registry.xml in the snapshots checkout update the webapp entry with the new filename and commit the change

ie

```
<application name="admin">
   <author>Martin M, Dan Rossi</author>
   <desc>Administration console</desc>
   <filename>admin-r3197-java6.war</filename>
</application>
```

Note

Subclipse version for committing changes also made by svnant in the snapshots repository, needs to be version 1.4 which is bound to subversion version 1.5

using this update site http://subclipse.tigris.org/update_1.4.x. Other svn clients also need to be bound to subversion 1.5 or you will get client too old errors.

18.5. Demo Applications Documentation

Following is documentation for the demo applications:

18.5.1. Bandwidth Check Application

This section explains the bandwidth check application and how to use it. The bandwidth check application handles two service method calls to trigger a download or upload rate check and return information to the flash client to determine what video bitrate to use.

18.5.1.1. Source Code

- Server Side - http://code.google.com/p/red5/source/browse/#svn/java/example/trunk/bwcheck
- Client Side - http://code.google.com/p/red5/source/browse/#svn/flash/trunk/bwcheck

18.5.1.2. Bandwidth Check Service Methods

The service method is enabled in the bean with a name bwCheckService.service.

```
<bean id="bwCheckService.service" class="org.red5.demos.bwcheck.BandwidthDetection" />
```

Inside the BandwidthDetection class there are two service methods:

- Trigger a server to client rate check

```
public void onServerClientBWCheck(Object[] params) {
 IConnection conn = Red5.getConnectionLocal();
 ServerClientDetection serverClient = new ServerClientDetection();
 serverClient.checkBandwidth(conn);
}
```

- Trigger a client to server rate check

```
public Map<String, Object> onClientBWCheck(Object[] params) {
 ClientServerDetection clientServer = new ClientServerDetection();
 return clientServer.onClientBWCheck(params);
}
```

18.5.1.3. ServerClientDetection

The ServerClientDetection class detects server to client bandwidth. 3 set of payload data arrays are intialized, the first with 1200 keys, and the next two with 12000 keys ie

```
for (int i = 0; i < 12000; i++) {
  payload_1[i] = Math.random();
}

  p_client.setAttribute("payload_1", payload_1);
```

The start microtime is recorded, along with an initial number of bytes sent to the client.

To initiate the handshake with the client method **onBWCheck** is called with parameters

- count - the number of times a result has been received from the client
- sent - the number of times the client method onBWCheck has been called
- timePassed - The interval time in milliseconds since the beginning of the bandwidth checking has occured.
- latency -
- cumLatency - the value of the increased passes from server to client.

```
private void callBWCheck(Object payload)
 {
  IConnection conn = Red5.getConnectionLocal();


  Map<String, Object> statsValues = new HashMap<String, Object>();
  statsValues.put("count", this.count);
  statsValues.put("sent", this.sent);
  statsValues.put("timePassed", this.timePassed);
  statsValues.put("latency", this.latency);
  statsValues.put("cumLatency", this.cumLatency);
  statsValues.put("payload", payload);

  if (conn instanceof IServiceCapableConnection) {
   ((IServiceCapableConnection) conn).invoke("onBWCheck", new Object[]{statsValues}, this);
  }
 }
```

An initial payload is sent with a size of 1200 keys, of the second pass, if the pass count is less than 3 and the time interval passed is less than 1 second progressively increase the payload packet sent with a size of 12000 keys.

On the next pass if its between 3 and less than 6 times and less than 1 second, send the 3rd payload packet.

On the next pass if its greater than 6 times and less than 1 second, send the 4th payload packet.

Once the times passed reaches the amount of times sent, send the client the calculated rate, calculated by the following

```
this.deltaDown = (endStats.getWrittenBytes() - beginningValues.get("b_down")) * 8 / 1000; // bytes to
            this.deltaTime = ((now - beginningValues.get("time")) - (latency * cumLatency)) / 1000; /

            if (Math.round(deltaTime) <= 0) {
    this.deltaTime = (now - beginningValues.get("time") + latency) / 1000;
   }
   this.kbitDown = Math.round(deltaDown / deltaTime); // kbits / sec

            if (kbitDown < 100) this.kbitDown = 100;

            log.info("onBWDone: kbitDown: {} deltaDown: {} deltaTime: {} latency: {} ", new Object[]{

            this.callBWDone();
```

This will call a client method **onBWDone**

- kbitDown - the kbits down value
- deltaDown -
- deltaTime -
- latency - The latency delay calculated between server and client

```
private void callBWDone()
 {
  IConnection conn = Red5.getConnectionLocal();

  Map<String, Object> statsValues = new HashMap<String, Object>();
  statsValues.put("kbitDown", this.kbitDown);
  statsValues.put("deltaDown", this.deltaDown);
  statsValues.put("deltaTime", this.deltaTime);
  statsValues.put("latency", this.latency);

  if (conn instanceof IServiceCapableConnection) {
   ((IServiceCapableConnection) conn).invoke("onBWDone", new Object[]{statsValues});
  }
 }
```

18.5.1.3.1. Client Side Download Detection

Client side callback methods are setup to enable the detection.

```
public function onBWCheck(obj:Object):void
```

```
 {
   dispatchStatus(obj);
 }

 public function onBWDone(obj:Object):void
 {
  dispatchComplete(obj);
 }
```

And then the information is obtainable on the Object argument

```
public function onServerClientComplete(event:BandwidthDetectEvent):void
 {
  txtLog.data += "\n\n kbit Down: " + event.info.kbitDown + " Delta Down: " + event.info.deltaDown +
  txtLog.data += "\n\n Server Client Bandwidth Detect Complete";
  txtLog.data += "\n\n Detecting Client Server Bandwidth\n\n";
  ClientServer();
 }
```

18.5.1.4. ClientServerDetection

The ClientServerDetection class helps detect client to server bandwidth. The server side method **onClientBWCheck** is called with some information to help the client to determine the bandwidth.

- cOutBytes - The bytes read from the client
- cInBytes - The bytes sent to the client
- time -

```
public Map<String, Object> onClientBWCheck(Object[] params) {
  final IStreamCapableConnection stats = this.getStats();

  Map<String, Object> statsValues = new HashMap<String, Object>();
  Integer time = (Integer) (params.length > 0 ? params[0] : 0);
  statsValues.put("cOutBytes", stats.getReadBytes());
  statsValues.put("cInBytes", stats.getWrittenBytes());
  statsValues.put("time", time);

  log.info("cOutBytes: {} cInBytes: {} time: {}", new Object[]{stats.getReadBytes(), stats.getWritter

  return statsValues;

 }
```

19.1. Overview

As of version 0.8, the Red5 Testing framework has been modified and updated. The unit-tests have been updated to pass, and a automated system-testing and continuous integration framework has been added. This document attempts to explain the thoughts and architecture involved so as to facilitate review by the Red5 core team.

19.2. How to Start Testing Without Reading This Chapter

To see the results of the Red5 continuous build server (this URL may change), go to http://build.theyard.net/.

The build server runs after every check-in, as well as every night. We test against JDK5 and JDK6.

To run all the Red5 unit tests yourself, check out the Red5 tree, and run:

```
$ ant run-tests
```

To see the results, open this file in a browser

```
doc/test/index.html
```

To run all the Red5 system tests, make sure you don't already have red5 running then in one terminal type:

```
$ ant run-tests-server
```

Then open this file in a browser. When the security dialog comes up, give it access to your camera, and select the box to remember the setting (if you want it to auto-run in the future):

```
test/fixtures/red5-selftest.swf
```

You can find all log files generate by the red5 test server in:

```
bin/testcases/testreports/dist/log
```

You can find the documentation for what the system tests do here (and can add to it by just putting ASDoc style comments in any System tests you add):

Current Flash System Tests [http://build.theyard.net/job/red5_flash_selftest_trunk_flex3.1/javadoc/]

And that's it.

19.3. Who Should Read This Chapter In Depth?

This chapter is targeted at people who are:

1. Modifying Red5 code directly and want to make sure their code works.
2. Interested in how Red5 is working to improve quality, and has some experience with software testing.
3. People who have found a bug in Red5, and want to submit a patch with a Unit Test or System Test that will catch regressions.
4. Goblins.

19.4. Red5 Testing Strategy

The Red5 Testing Strategy has 5 components to it:

\#	Type	Description
1	Code	Write great code; we've done this from day one and see no reason to stop now.
2	Unit Tests	
3	Functional Tests	Write functional tests to simulate network interactions

This is not yet implemented. We plan to do this in the future using RTMPClient, but any help the community can give here is appreciated. ||

4	System Tests	Use flash-based system-tests (using AsUnit [http://asunit.org/]) that make sure end-to-end interaction with Adobe's flash player works as expected. For example, we test that we can connect from Flash via RTMP, we can play back pre-recorded FLV files, and we can publish from a Camera.
5	Continuous Building	Build and run all tests every time someone checks in to make sure all tests still pass. Currently the continuous server can be found here (but it will move): http://build.theyard.net/

19.5. Red5 Testing Props

Major props go to the following folks:

1. The red5 development team because, well, test frameworks don't mean shit if you don't got good stuff to test.

2. Thijs, who set up the initial Red5 build server, and showed me how to get get Apache and Tomcat working nicely together.

19.6. Unit Testing

19.6.1. Purpose

The purpose of a unit test is to make sure a java object operates according to its specification, regardless of how it is plugged in with other objects. For a good overview of check out this Wikipedia Unit Testing web page [http://en.wikipedia.org/wiki/Unit_testing]. For example, you can (and we do) have Unit Tests that test we're encoding and decoding data from AMF codec's correctly using Mock Objects, that test whether or not we can load information from Spring, and whether or not we can inject Meta Data into FLV viles correctly.

19.6.2. Technology

Red5 uses the JUnit [http://junit.org/] unit testing framework. If you're not familiar with that suite, please check it out. It is the de-facto standard for Java Unit Testing.

19.6.3. Running Tests

To run tests, checkout the latest server build, and run the following ant command:

```
$ ant run-tests
```

To see the results, you can open the doc/test/index.html file once the tests are finished.

```
firefox doc/test/index.html
```

19.6.4. Creating New Tests

Writing unit tests in the JUnit [http://junit.org] framework is beyond the scope of this document, but you can find help at the JUnit site. The Red5 unit test framework support JUnit 4.0, but can run JUnit 3.x-style tests as well. To create a new unit test, just create a new JUnitclass source file to the right path under test, and end the source filename with the string "Test.java". For example, to test org.red5.server.api.ANewClass, you would create the following java file under the "test" directory:

```
test/org/red5/server/api/ANewClassTest.java
```

Once you do that, the compile process should pick up your new file and run the tests automatically.

By default, the run-tests runs all unit tests in the following directory:

```
bin/testcases/testreports
```

19.6.5. Running unit tests from eclipse

In theory every should work, but you may need to set the directory eclipse runs the test from. Make sure it is set to:

```
bin/testcases/testreports
```

19.6.6. Guidelines for New Unit Tests

Unit tests help make the code base stronger, but that said we do need to make sure that unit tests meet certain guidelines so we can have a useful build process. Those guidelines are:

- Unit Tests MUST be self-contained (i.e. each test should run independently).
- Unit Tests MUST not require a Red5 server to be running.
- Unit Tests MAY assume that no Red5 instance is currently running while they run (and so may fire up Red5 objects that bind to ports if appropriate).
- Unit Tests MUST NOT require thread-timing specific to your machine to run (or fail) consistently.
- Unit Tests MUST run successfully 100% in order to be checked in. That means, no checking in tests that fail with "not implemented".
- Unit Tests SHOULD try to avoid introducing new dependencies, but if you must use one (for example http://multithreadedtc.googlecode.com/ can be useful), identify it when you submit a JUnit test case and we'll review whether or not to add to the ivy test dependencies.
- Unit Tests MUST document what they do, and how to tell if they really worked, in JavaDoc comments above each test method.
- Unit Tests SHOULD be written using JUnit 4.x annotations

19.6.7. Submitting New Unit Tests

We really want new Unit Tests, so if you have a Unit Test that meets the above guidelines we'd love to consider it. To submit it, do the following:

- Review the guidelines above. Really.
- If you're fixing a bug:
  - Create a unit test and make sure it fails 100% of the time before you fix the bug.
  - Fix the bug.
  - Ensure that unit test succeeds 100% of the time after you fix the bug.

- If you're testing a new feature:
  - Write your new feature.
  - Create a diff file with your patch using "svn diff" from tip of tree.
  - Ensure that unit test succeeds 100% of the time after you fix the bug.
- File a new issue in Trac in the "Developer Tools" section. Please attach the diff files and the contents of any new files, and specify the following:
  - What the test tests.
  - Brief overview of how it works.
- Someone (probably Art) will review it and get back to you on changes that may be needed, or will commit it.

19.6.8. Suggesting New Unit Tests

We really do want new unit tests, and would love suggestions. But bear in mind that Red5 is a 100% volunteer project and most people who work on it have full time day jobs (their time spent on red5 is a labor of love). So don't be hurt if your suggestion for a test is not picked up on.

That said, a great way to suggest an area to test is to go ahead and write the system test yourself\! Send it to the list, and it'll probably get a warm reception.

19.7. Integration Testing

We currently don't have a integration testing framework [http://en.wikipedia.org/wiki/Integration_testing], but when I next return to this area, I'll try adding one. The basic idea for this (which I love) is to make a framework based on the RTMPClient.

19.7.1. Purpose

The purpose of integration testing is to start to plug together different simpler modules (that hopefully have been unit tested) to see if they play nice together. See this Integration Testing Wikipedia Pagefor an overview of the concepts.

19.8. System Testing

19.8.1. Purpose

When all is set and done, and you've Unit tested everything, and did integration testing by mixing together different components, you're still not done. At some point, a user is going to pick up your application, and start using it. And if you haven't tested from that end to your code and back, well chances are something will break.

That's where System Testing [http://en.wikipedia.org/wiki/Functional_testing] comes in. In the System Test we try to do some basic end-to-end tests to see if our code performs as expected from the end-user's perspective.

19.9. Technology

For Flash system testing (a.k.a selftest), we use the ASUnit|http://asunit.org/ [http://junit.org/] framework, which is very similar to JUnit. You can find the current Flash self test here.

We also use The Yard Flash Libraries [http://theyard.googlecode.com/] to abstract away some components of Flash connecting and stream playing, but you're not required to use those libraries if you submit new flash unit tests.

19.10. Running Tests

The Red5 system tests require a special test server to be running. This is just a mostly empty red5 server with one special application installed:

```
http://localhost/selftest
```

That selftest Red5 application has the following service exposed under the name "echo":

```
red5.server.services.IEchoService.java:
[http://code.google.com/p/red5/source/browse/java/server/trunk/test/org/red5/server/service/IEchoSe
```

Every method on that Java Interface is callable from Flash by using the prefix "echo.". For example, "echo.echoNumber" will call the echoNumber method over RTMP/AMF.

There are two ways to run the system test:

Attended - Run a Test Server

```
$ ant run-tests-server
```

Start up the Flash Self Test application

```
flashplayer tests/fixtures/red5-selftest.swf
```

This method assumes someone is watching the test.

Unattended -

```
$ ant run-tests-systemtest
```

This only works on Linux. It starts up a red5 server, runs the system test in the background, and then collects all log artifacts in the directory "output" relative to the current directory. It

will also take snapshop pictures of the desktop as running if ImageMagick's import tool is installed. ||

System tests run the server with RED5_HOME set to bin/testcases/testreports/dist, and runs the flash clients from the directory bin/testcases/testreports/fixtures.

Lastly, you should ensure red5 is not currently running on the server you run a system test on. However, because the system tests use their own version of Red5, you don't need to worry about them clobbering anything in your own Red5 installation.

The System Tests use a series of scripts located in:

```
test/scripts
```

to automatically start-up and shutdown red5, as well as find the necessary flash logs from different parts of the system. The main one of interest is:

```
test/scripts/red5-flash-player-headless
```

It assumes it's running under a Windowing system (e.g. XWindows) with a Bourne Shell, and then starts a clean red5 server, runs the Flash system tests, and cleans up afterwards.

19.11. Creating New Tests

Writing unit tests in theAsUnit framework is beyond the scope of this document, but you can find help at the AsUnit site.

But to create a new system test, you can start with the Flash selftest application:svn checkout http://red5.googlecode.com/svn/flash/trunk/selftest red5_selftestTo create a new AsUnit test just create a newclass source file to the right path under test, and end the source filename with the string "Test.as". For example:

```
test/org/red5/server/decodingComplexObjectOverAMFTest3.as
```

Once you do that, you'll need to modify the AllTests.as file in the directory to add your new test.

We use The Yard flash libraries|http://code.google.com/p/theyard/ [http://ofb.net/~aclarke/theyard/flashutils-0.1.0/api/] to abstract away some of the complexities of connecting to and manipulating NetConnection and NetStream objects. See the Yard flash library documentation [The]. You don't have to use them for new tests, but they can make things a lot easier (for example, by taking care of connecting for you).

To see documentation of existing tests, run:

```
ant doc
```

19.12. A Sample System Test

Here's a very straightforward System Test submitted by trebor (at) vlideshow.com.

This test connects to the test server and calls the "echo.echoString" method to pass a String to Red5, and then make sure we get the same array back. It tests both AMF0 and AMF3 using the same code path because they should be the same.

```
EchoStringTest.as: AMF0 and AMF3 Strings sent over RTMP Test
[http://code.google.com/p/red5/source/browse/flash/trunk/selftest/test/src/org/red5/server/io/EchoStr
```

19.13. Guidelines for New System Tests

Unfortunately Flash ActionScript is not as forgiving as Java is about cleaning up after a test is finished, so the guidelines for writing System Tests are somewhat more involved. Also, these tests MUST be runnable in a "unattended" mode \-\- meaning requiring no human interaction, to the bar is higher.

- System Tests MUST not require any human interaction \-\- i.e. if a human can't give permission for something, it should fail without blocking.
- System Tests MAY draw on the flash screen but MUST remove any artifacts when done
- System Tests MUST be self-contained (i.e. each test should run independently).
- System Tests MAY assume that a Red5 instance is running on localhost, on port 1935, and that the selftest application is available.
- System Tests MAY assume that the selftest application has the Echo service installed.
- System Tests MUST clean up fully after themselves. That is, they must disconnect and remove any event handlers..
- System Tests MUST run successfully 100% in order to be checked in. That means, no checking in tests that fail with "not implemented".
- System Tests SHOULD try to avoid introducing new dependencies, but if you must use one (for example http://theyard.googlecode.com/ can be useful), identify it when you submit a test case and we'll review whether or not to add to the ivy test dependencies.
- System Tests MUST document what they do, and how to tell if they really worked, in AsDoc comments above each test method.

19.14. Submitting New System Tests

We really want new System Tests, so if you have a System Test that meets the above guidelines we'd love to consider it. To submit it, do the following:

- Review the guidelines above. Really.
- If you're fixing a bug:

** Create a system test and make sure it fails 100% of the time before you fix the bug. ** Fix the bug, and run a new test server. ** Ensure that unit test succeeds 100% of the time after you fix the bug.

- If you're testing a new feature:

** Write your new feature. ** Create a unit test and make it it succeeds 100% of the time with the new feature.

- Create a diff file with your patch using "svn diff" from tip of tree.
- File a new issue in Trac in the "Developer Tools" section. Please attach the diff files and the contents of any new files, and specify the following:

** What the test tests ** Brief overview of how it works

- Someone (probably Art) will review it and get back to you on changes that may be needed, or will commit it.

If your change is accepted, we'll integrate it into the Flash self-test, and update the Java Server trunk to use the new Flash selftest as our system test.

19.15. Suggesting New System Tests

We really do want new tests, and would love suggestions. But bear in mind that Red5 is a 100% volunteer project and most people who work on it have full time day jobs (their time spent on red5 is a labor of love). So don't be hurt if your suggestion for a test is not picked up on.

That said, a great way to suggest an area to test is to go ahead and write the system test yourself\! Send it to the list, and it'll probably get a warm reception.

19.16. Continuous Integration

19.16.1. Overview

The last step of our testing framework is to run a continuous build. See this Wikipedia Pagefor some of the principles involved.

The basic idea is to do a checkout, run all unit, functional and system tests, and then notify the person who checked in, and any others that are interested, about the current state of the build. The idea is that it is easier to fix bugs when they are introduced, than if they are found days or weeks later.

19.16.2. Technology

We use Hudson as our continuous build server running inside a Tomcat instance (running as the Hudson, not root, user) that is forwarded to by Apache2. This currently runs on an Amazon EC2 small instance hosted at:

```
[http://build.theyard.net/]
```

E-Mail notification of bad builds are sent to the last person who checked in, and to the red5-builds (at) googlecode.com group.

We run the following builds continuously:

- We build the java/server/trunk against JDK 1.6 on Linux i386 (Ubuntu) and run all units tests
- If this is successful, we run all system tests under JDK 1.6 on Linux i386 (Ubuntu) .
- We build the java/server/trunk against JDK 1.5 on Linux i386 (Ubuntu) and run all unit tests.
- If this is successful, we run all system tests under JDK 1.5 on Linux i386 (Ubuntu) .

19.16.3. How To Run The Continuous Build

The Continuous Build server will run any time you check something into the Java Server. It also runs once every night.

If you're on the Red5 dev team and want to set up new job, or log-in to hudson directly, talk to Art Clarke and he'll hook you up.

19.16.4. How to Submit New Jobs for Continuous Building

For now, send a request to red5devs@osflash.org and we'll evaluate it.

19.17. How you can help with Continuous Building

If you're willing do donate a i386 Amazon EC2 instance or an i86_64 Amazon EC2 instance, we're in need of both to do testing on. The current set up is temporary.

19.17.1. How to Set up a Continuous Build Server

NOTE: THIS SECTION IS MEANT FOR SERIOUSLY ADVANCED RED5 USERS. 99.999999% of people shouldn't even read this.

Glad you asked. We used an Amazon EC2 instance to get us started. Specifically this AMI from Eric Hammon at alestic.com.

We then created a script that makes that image into one that can run Red5's continuous build server. See:

```
http://red5.googlecode.com/svn/build/remote/trunk/ec2/
```

To set up an AWS EC2 instance to build and auto-test red5, do the following:

1. Learn how to use Amazon EC2.
2. Start up an instance of Ubuntu 8.04 LTS Hardy: ami-1cd73375
3. Check out the Red5 remote build branch:

```
$ svn checkout http://red5.googlecode.com/build/remote/trunk/
```

1. Copy the ec2/ec2-explode directory to your new Amazon EC2 instance:

```
cd ec2/ec2-explode
./ec2-implode ../ec2.tgz
scp -i YOUR_AWS_KEYPAIR root@YOUR_AWS_PUBLIC_IP:/tmp
```

1. Log into your AWS EC2 instance:

```
$ ssh -i YOUR_AWS_KEYPAIR -l root YOUR_AWS_PUBLIC_IP
```

1. Prepare your ec2-explode package:

```
cd tmp
  tar xzvf ec2.tgz
```

1. Run the ec2-explode script:

```
./ec2-explode
```

You will have to accept the Sun JDK license, choose a password for your XAuthority file (don't worry \-\- the X ports aren't opened, you just need that to run a headless X Server to make the Flash System Tests run), and enter the data necessary to send mail from your machine.

1. You'll need to patch up your Apache 2 files to reflect your domain name:

```
rename /etc/apache2/sites-enabled/build.theyard.net /etc/apache2/sites-enabled/YOUR-APACHE-SITE
vi /etc/apache2/sites-enabled/YOUR-APACHE-SITE
service apache2 restart
```

1. Go to your website and make sure hudson is running:

```
http://YOUR-AWS-PUBLIC-IP/
```

It's probably a good idea to change the default "hudson" password as well. It defaults to: 10.

```
fmskiller
```

This is the password for the Hudson UI, not the password on the hudson linux account. By default the linux hudson account doesn't allow log in using passwords.

Available in Red5 is a Plugin architecture system to enable to extend features into Red5 for an entire server or application. Plugins are loaded on startup and then configured on a per application basis.

Tip

Currently only available via SVN trunk.

20.1. Loading

Plugins are compiled into jar files which will be loaded and parsed by the server on startup.

Path to load the plugins is in:

```
/path/to/red5/plugins
```

20.2. Configuring

Plugins can be configured via the application adaptor by setting a property inside the red5-web.xml config file. The plugins property has one child node which is a list of plugins to load for the application

```
<bean id="web.handler" class="org.red5.demos.oflaDemo.Application">
<property name="plugins">
            <list>
...
```

Inside the list can be configured multiple plugins beans

```
<bean class="org.red5.server.plugin.PluginDescriptor">
 <property name="pluginName" value="authPlugin"/>
    <property name="pluginType" value="org.red5.server.plugin.auth.AuthPlugin"/>
    <property name="method" value="getRed5AuthenticationHandler"/>
    <property name="methodReturnType" value="org.red5.server.plugin.auth.Red5AuthenticationHandler"/>
</bean>
```

- pluginName - the name of the plugin compiled into the plugin
- pluginType - the fully qualified name of the plugin class ie org.red5.server.plugin.auth.AuthPlugin
- method - method is a getter for a factory method as the plugin may provide multiple features ie getRed5AuthenticationHandler
- methodReturnType - methodReturnType is the fully qualified name of the plugin factory class to provider a certain feature ie org.red5.server.plugin.auth.Red5AuthenticationHandler

Plugins are able to be configured with config properties using a property setter which is a spring hashmap list of values ie

```
<bean class="org.red5.server.plugin.PluginDescriptor">
                    <property name="pluginName" value="securityPlugin"/>
                    <property name="pluginType" value="org.red5.server.plugin.security.SecurityPlugin"
                    <property name="method" value="getPlaybackSecurityHandler"/>
                    <property name="methodReturnType" value="org.red5.server.plugin.security.PlaybackS
                    <property name="properties">
                        <map>
                            <entry>
                                <key><value>htmlDomains</value></key>
                                <value>file:///path/to/allowedHTMLdomains.txt</value>
                            </entry>
                            <entry>
                                <key><value>swfDomains</value></key>
                                <value>file:///path/to/allowedSWFdomains.txt</value>
                            </entry>
                        </map>
                    </property>
                </bean>
```

Where using spring map syntax each property key / value is within an entry tag

```
<entry>
          <key><value>htmlDomains</value></key>
                                <value>file:///path/to/allowedHTMLdomains.txt</value>
                            </entry>
```

Tip

Due to some issues with spring context paths loading in plugins, full absolute paths are required to files to be loaded ie file:///path/to/allowedHTMLdomains.txt

20.3. Developing

To begin developing a new plugin it's best to download already built ones from SVN to use as a template ie

```
http://red5.googlecode.com/svn/java/plugins/trunk/securityplugin/
```

To configure some ant properties to compile the plugin correctly, inside build.properties update the following

```
red5.root=/www/red5_server_xuggle_timestamp_fixes
```

```
main-class=org.red5.server.plugin.security.SecurityPlugin
```

Where red5.root is the path to red5 and main-class is the fully qualified name of the plugin.

At the top of the ant build script, update the project name to the name of the plugin which will be used to generate the plugin jar file

```
<project name="securityplugin" basedir="." default="all" xmlns:ivy="antlib:org.apache.ivy.ant">
```

Tip

Make sure the following ant property is set to either the main-class property or static to the plugin fully qualified name. If this is not setup correctly the Plugin loader will detect there is no manifest and not load the plugin. <attribute name="Red5-Plugin-Main-Class" value="${main-class}"/>

20.3.1. Plugin Main Class

The plugin main class requires to extend the Red5Plugin base class which the plugin loader will then use this as the main class for loading the plugin and factory methods.

```
public class SecurityPlugin extends Red5Plugin {

 private static Logger log = Red5LoggerFactory.getLogger(SecurityPlugin.class, "plugins");

 private static Serializer serializer = new Serializer();

 private MultiThreadedApplicationAdapter application;


 public void doStart() throws Exception {
  log.debug("Start");
 }

 public void doStop() throws Exception {
  log.debug("Stop");
 }

 public String getName() {
  return "securityPlugin";
 }

 public void setApplication(MultiThreadedApplicationAdapter app) {
  log.trace("Setting application adapter: {}", app);
  this.application = app;
 }

 //methods specific to this plug-in

 public PlaybackSecurityHandler getPlaybackSecurityHandler() {
  PlaybackSecurityHandler ph = null;
  try {
   ph = (PlaybackSecurityHandler) Class.forName("org.red5.server.plugin.security.PlaybackSecurityHandl
```

```
  ph.setApplication(application);
 } catch (Exception e) {
  log.error("PlaybackSecurityHandler could not be loaded", e);
 }
 return ph;
}

public PublishSecurityHandler getPublishSecurityHandler() {
 PublishSecurityHandler ps = null;
 try {
  ps = (PublishSecurityHandler) Class.forName("org.red5.server.plugin.security.PublishSecurityHandler
  ps.setApplication(application);
 } catch (Exception e) {
  log.error("PublishSecurityHandler could not be loaded", e);
 }
 return ps;
}

public SharedObjectSecurityHandler getSharedObjectSecurityHandler() {
 SharedObjectSecurityHandler sh = null;
 try {
  sh = (SharedObjectSecurityHandler) Class.forName("org.red5.server.plugin.security.SharedObjectSecur
  sh.setApplication(application);
 } catch (Exception e) {
  log.error("SharedObjectSecurityHandler could not be loaded", e);
 }
 return sh;
}

//common methods

/**
 * Invokes the "onStatus" event on the client, passing our derived status.
 *
 * @param conn
 * @param status
 */
public static void writeStatus(IConnection conn, StatusObject status) {
 //make a buffer to put our data in
 IoBuffer buf = IoBuffer.allocate(128);
 buf.setAutoExpand(true);
 //create amf output
 Output out = new Output(buf);
 //mark it as an amf object
 buf.put(AMF.TYPE_OBJECT);
 //serialize our status
    status.serialize(out, serializer);
    //write trailer
 buf.put((byte) 0x00);
 buf.put((byte) 0x00);
 buf.put(AMF.TYPE_END_OF_OBJECT);
 //make the buffer read to be read
 buf.flip();

 //create an RTMP event of Notify type
 IRTMPEvent event = new Notify(buf);

 //construct a packet
 Header header = new Header();
 Packet packet = new Packet(header, event);

 //get our stream id
 int streamId = BaseRTMPHandler.getStreamId();
 //set channel to "data" which im pretty sure is 3
 header.setChannelId(3);
 header.setTimer(event.getTimestamp()); //0
 header.setStreamId(streamId);
```

```
  header.setDataType(event.getDataType());

  //write to the client
  ((RTMPConnection) conn).write(packet);
 }

}
```

The getter method getName is required to be set so the plugin can be identified and loaded correctly using the plugin config in the application

```
public String getName() {
  return "securityPlugin";
}
```

```
public void doStart() throws Exception {
  log.debug("Start");
 }

 public void doStop() throws Exception {
  log.debug("Stop");
 }

 public String getName() {
  return "securityPlugin";
 }

 public void setApplication(MultiThreadedApplicationAdapter app) {
  log.trace("Setting application adapter: {}", app);
  this.application = app;
 }
```

Other methods are doStart , doStop and setApplication setter method which is required to set a reference to the loaded application.

The plugin factory method is required to be configured and return a reference to the factory class which is then loaded in the application config. A reference to the application can also be set if desired so the factory method class can manipulate methods and properties on the application.

```
public PlaybackSecurityHandler getPlaybackSecurityHandler() {
  PlaybackSecurityHandler ph = null;
  try {
   ph = (PlaybackSecurityHandler) Class.forName("org.red5.server.plugin.security.PlaybackSecurityHandl
   ph.setApplication(application);
  } catch (Exception e) {
   log.error("PlaybackSecurityHandler could not be loaded", e);
```

```
  }
  return ph;
 }
```

20.3.2. Factory Method Class

The factory method class is what gets configured to load and is where the features to run happen. It is required to extend the ApplicationLifeCycle class as well as implement the IRed5PluginHandler interface.

```
public abstract class SecurityBase extends ApplicationLifecycle implements IRed5PluginHandler {
...
```

Setter methods for both application and properties is required so the class is able to get a reference to the application as well as have property configs set.

```
 public void setApplication(MultiThreadedApplicationAdapter app) {
  application = app;
 }

 public void setProperties(Map<String, Object> props) {
  properties = props;
 }
}
```

An init method is required to be overridden to enable the factory class to start and then enable features and manipulate the application

```
 @Override
 public void init() {
  if (properties.containsKey("htmlDomains")) {
   htmlDomains = properties.get("htmlDomains").toString();
  }
  if (properties.containsKey("swfDomains")) {
   swfDomains = properties.get("swfDomains").toString();
  }

  allowedHTMLDomains = readValidDomains(htmlDomains, "HTMLDomains");

  // Populating the list of domains which are allowed to host a SWF file
  // which may connect to this application
  allowedSWFDomains = readValidDomains(swfDomains, "SWFDomains");

  // Logging
  if (HTMLDomainsAuth) {
   log.debug("Authentication of HTML page URL domains is enabled");
  }
  if (SWFDomainsAuth) {
   log.debug("Authentication of SWF URL domains is enabled");
```

```
  }

  log.debug("...loading completed.");

  //now register with the application
  application.registerStreamPlaybackSecurity(this);
 }
```

Part III. App Server

Application Server section including plugins

- Chapter 21, *JEE Container Plug-ins*
- Chapter 20, *Plugins*
- Chapter 22, *SSAS Support - Basis*

21.1. The Containers

There are three JEE (Java Enterprise Edition) containers available for embedding within your Red5 server. The Tomcat container was previously embedded by default, but it has since been extracted to allow the users to select or create their own.
We offer the following containers: Tomcat Server [http://tomcat.apache.org/] pre-compiled [http://red5.googlecode.com/svn/repository/red5/tomcatplugin-1.0.jar] source [http://red5.googlecode.com/svn/java/plugins/trunk/tomcat/] Winstone Server [http://winstone.sourceforge.net/] pre-compiled [http://red5.googlecode.com/svn/repository/red5/winstoneplugin-1.0.jar] source [http://goo.gl/JumDi/] None of the containers is enabled by default; an empty jee-container.xml file exists within the Red5 configuration directory to prevent startup issues. To enable a here [http://www.thefunnyquotessayings.com/cool-hilarious-funny-quotes-sayings/] you must extract the plugin's jee-container.xml file and place it the Red5 configuration directory and restart your server.

21.2. Fixing plug-in startup errors

The most likely cause of any start up issues you may experience is their missing dependent libraries. But if "beans" are not found, extract the jee-container.xml file from plugin jar and place it in the "conf" directory. If you see an error about a "Loader" class not being found, this will be because the container

you have chosen has not been deployed to the "plugins" directory. Ensure that your selected containers jar is in the plugins directory directly under your red5 home directory cash advance loans [http://www.cashadvance-loans.net/]. The exception will read like so for a missing tomcat plugin jar.

```
org.springframework.beans.factory.CannotLoadBeanClassException: Cannot
find class [org.red5.server.tomcat.TomcatLoader] for bean with name
'tomcat.server' defined in class path resource [jee-container.xml]; nested exception is
java.lang.ClassNotFoundException: org.red5.server.tomcat.TomcatLoader
```

21.2.1. Tomcat Plugin

The following jars are required by Tomcat, place these in the red5/plugins directory funny jokes [http://itshumour.blogspot.com/2011/07/funny-marriage-jokes.html]:

```
catalina-6.0.26.jar
jasper-6.0.26.jar
jasper-jdt-6.0.26.jar
jasper-el-6.0.26.jar
tomcat-coyote-6.0.26.jar
tomcat-juli-slf4j-1.5.0.jar
```

21.2.2. Jetty Plugin

The following jars are required by Jetty quotes to live by [http://tinyurl.com/72llty5] , place these in the red5/plugins directory:

```
jetty-continuation-7.1.6.v20100715.jar
jetty-deploy-7.1.6.v20100715.jar
jetty-http-7.1.6.v20100715.jar
jetty-io-7.1.6.v20100715.jar
jetty-security-7.1.6.v20100715.jar
jetty-server-7.1.6.v20100715.jar
jetty-servlet-7.1.6.v20100715.jar
jetty-util-7.1.6.v20100715.jar
jetty-webapp-7.1.6.v20100715.jar
jetty-xml-7.1.6.v20100715.jar
```

21.2.3. Winstone Plugin

######## ########## ######## [http://megaupl0ad.com/########-########-##########-##-########/] The following jar is required by Winstone , place these in the red5/plugins directory: xbox 360 parts [http://www.repairpartstock.com/]

```
winstone-lite-0.9.10.jar
```

The foundation classes for SSAS support are now available, the actual SSAS API (partial) to be provided as soon as time allows. In a nutshell, to use the layer, you place your FMS scripts in a webapp configured to use FMSAdpater. From there on in, the scripts drive the behaviour of the Red5 application. Before hopes are raised too high too soon, bear in mind that the code attached actually provides the proposed basis for Red5 scripting applications - it is from this code base that SSAS 'simulation' will be provided - this is the 'hello world' of the underlying implementation. Rather than go into detail here (for the moment), the source code attached has been commented and should provide some insight.

Contents:

ScriptingApplication. A generic encapsulation of common operations associated with obtaining and evaluating programs written in JSR-223 specification supported languages. Pending critical review, it is hoped that ScriptingApplication will become the adopted means to support scripted Red5 applications.

ECMAScriptApplication. A specialization of ScriptingApplication, ECMAScriptApplication builds on the functionality of ScriptingApplication. In addition it provides a streamlined interface to define Host objects, gobal properties, constants and methods. It has been designed to take advantage of the underlying Mozilla Rhino implementation that ships with JavaSE6. The reason to support the 'phobos' implementation, rather than building Rhino from scratch is primarily to avoid introducing potential instabilities to the code base, and the phobos implementation is generally considered stable. Unlike the SSAS interpreter demoed at FITC, which was based on a (since deprecated) bespoke Rhino build, the forthcoming SSAS interpreter will be based on ECMAScriptApplication and the phobos implementation.

ScriptingApplicationAdapter. ScriptingApplicationAdapter extends ApplicationAdapter and is a generic encapsulation of the common operations of hooking up scripting application functionality to the Java based Red5 API. At present, the imlementation only supports single scope applications. Multiscope scripting application support is in development.

ECMAScriptApplicationAdapter. ECMAScriptApplicationAdapter subclasses ScriptingApplicationAdapter to support the means for implementors to develop bespoke ECMAScript Host Object API's to run on top of the Red5 Java API. FMSApplicationAdapter will be a specialization of ECMAScriptApplicationAdapter. In addition to these classes a number of proposed conventions have been developed, primarily with regard to defining ECMAScript host objects. Any of you familiar with the Mozilla Rhino host object API will be familiar with its eccentricities. Example source code provided details the proposed conventions for adoption by Red5 developers when developing scripting support in ECMAScript derivitive languages.

Instructions. Obtain Red5 from the repository and build the project in Eclipse as per the norm. Download the attached files (source.zip and scriptingadapter.zip). Place the class file contents of source.zip into the src folder of your Red5 project directory. Place the contents of scriptingadapter.zip into your webapps folder. Read through the source and you should be good to go\!

If you want some more information, feel free to unzip ScriptingApplication(standalone).zip - a standalone development implementation of ECMAScriptApplication (that is, does not

require Red5) demonstrating a simple use case. A word of warning, however, some of the comments are incomplete. More meaningful examples will follow soon.

Please leave comments, feature requests, bug reports, experiences and criticisms on this page of the wiki as this project is developed further.

Part IV. Tutorials

Tutorials

- Chapter 23, *Red5 and Acegi Security*
- Chapter 24, *Edge Origin Clustering Configuration*
- Chapter 25, *What To Do When Red5 Freezes*
- Chapter 27, *Ivy setup with Eclipse*
- Chapter 28, *Java 6 Update 4 Target For OSX 10.5*
- Chapter 29, *Jruby scripting explained*
- ???
- Chapter 30, *Creating Patches From SVN*
- Chapter 31, *Ping*
- Chapter 32, *RTMP Client*
- Chapter 33, *Scopes and Contexts*
- Chapter 34, *Red5 and Spring JDBC*
- Chapter 35, *Deploying Red5 to Tomcat*
- Chapter 36, *Using Wireshark On OSX*

The current implementation of Red5 doesn't support a way for RoleBased Security.

Acegi Security System is on the list to be implemented but it wasn't so far. So i thought, for now, how could we use Acegi with Spring and Red5?

= Introduction =

Acegi Security [http://www.acegisecurity.org] is a powerful, flexible security solution for enterprise software, with a particular emphasis on applications that use Spring. Using Acegi Security provides applications with comprehensive authentication, authorization, instance-based access control, channel security and human user detection capabilities.

The main advantage of Ageci is the easy implementation and a standard approach of solving security matters. Rolebased with ACL permissions.

= The Start = I use seperate files for the setup of acegi in spring with Red5. The mean guide for this tutorial is the acegi tutorial available in the war of acegi security bin download. The current config allows to use any red5-*.xml. So create a file named _red5-security.xml_ in your WEB-INF Folder. You can also use your own red5-security.properties config handler for variable definitions.

In this example we use the simple Memorybased Authentication, you could also use Jdbc or any other provided by acegi.

You have to create a file called users.properties

```
admin=secretpassword,ROLE_SUPERVISOR
```

You can add your users by adding new lines with this syntax.

Our red5-security.xml should look like this

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spri
<beans>
 <bean id="placeholderConfig" class="org.springframework.beans.factory.config.PropertyPlaceholde
    <property name="location" value="/WEB-INF/red5-security.properties" />
 </bean>
 <bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
   <list>
    <ref local="daoAuthenticationProvider"/>
   </list>
  </property>
 </bean>
 <bean id="daoAuthenticationProvider" class="org.acegisecurity.providers.dao.DaoAuthenticationPr
  <property name="userDetailsService" ref="userDetailsService"/>
  <property name="userCache">
```

```
  <bean class="org.acegisecurity.providers.dao.cache.EhCacheBasedUserCache">
   <property name="cache">
    <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
     <property name="cacheManager">
      <bean class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
     </property>
     <property name="cacheName" value="userCache"/>
    </bean>
   </property>
  </bean>
 </property>
</bean>

<bean id="userDetailsService" class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
 <property name="userProperties">
  <bean class="org.springframework.beans.factory.config.PropertiesFactoryBean">
   <property name="location" value="/WEB-INF/users.properties"/>
  </bean>
 </property>
</bean>
</beans>
```

As you see we use a DataObjectAccess Provider compatible with a list of different Providers. We use a simple Cache based provider (EhCacheBasedUserCache) for faster authentication. (Doesn't matter, its still memory based) The service that provides the users information is the _userDetailsService_ bean. The property _userProperties_ specify the relative path to our _user.properties_ file.

Acegi security allows us to use more providers at the same time.

= Implementation in our program =

```
@Override
public boolean appConnect(IConnection arg0, Object[] arg1) {
 if (arg1.length==1)
 {
  if (arg1[0]!=null)
  {
   //we expect a format like this {name: "admin", password: "secret"}
   final HashMap m=(HashMap)arg1[0];
   UsernamePasswordAuthenticationToken t=new UsernamePasswordAuthenticationToken(m.get("

   ProviderManager mgr=(ProviderManager)masterScope.getContext().getBean("authenticationMan
   try {
    //authenticate the user against our proivder
    // IMPORTANT: the returning UsernamePasswordAuthenticationToken
    //            is not the same as our UsernamePasswordAuthenticationToken
    t=(UsernamePasswordAuthenticationToken)mgr.authenticate(t);
```

```
  }
  catch(BadCredentialsException ex)
  {
   //the information provided was wrong
   rejectClient("Wrong login information");
  }
  if (t.isAuthenticated())
  {
   arg0.getClient().setAttribute("authInformation", t);

   // The client is authenticated
   // You can use this in your functions called by the client
   // or event the StreamPublish Security handler
   log.debug("YESS!!! AUTHENTICATED!!!!!");
  }
 }
}
return super.appConnect(arg0, arg1);
```

So this is the easy way of implementation.

= Better approach = So, the above implementation is very easy and, I dont like such code in bigger apps. So, how can we provide a generic way of security, even if we have to work in an environment where we can provide anonymous authentication?

The solution is to write your own ClientRegistry or to change the Application Code

```
  public boolean connect(IConnection conn, IScope scope, Object[] params) {

log.debug("Connect to core handler ?");

      // Get session id
      String id = conn.getSessionId();

// Use client registry from scope the client connected to.
IScope connectionScope = Red5.getConnectionLocal().getScope();

      // Get client registry for connection scope
      IClientRegistry clientRegistry = connectionScope.getContext()
  .getClientRegistry();

      // Get client from registry by id or create a new one
      IClient client = clientRegistry.hasClient(id) ? clientRegistry
  .lookupClient(id) : clientRegistry.newClient(params);

// We have a context, and a client object.. time to init the conneciton.
conn.initialize(client);
```

```
 // we could checked for banned clients here
 return true;
}
```

This is the main code of our Corehandler : Application. So if we override the connect, we could integrate it directly.

The problem is, that we have to copy the code because otherwise we can not inject our client ID.

So lets take a look at the client registry

```
public IClient newClient(Object[] params) throws ClientNotFoundException,
  ClientRejectedException {
 IClient client = new Client(nextId(), this);
 addClient(client);
 return client;
}
```

We can now extend the ClientRegistry to AuthClientRegistry.

```
public class AuthClientRegistry extends ClientRegistry {

 protected static Log log = LogFactory.getLog(AuthClientRegistry.class.getName());

 public AuthClientRegistry() {
  // TODO Auto-generated constructor stub
  super();
 }
 @Override
 public IClient newClient(Object[] params) throws ClientNotFoundException, ClientRejectedExceptio
   //We can do our authentication here.
}
```

Edit: this part is modified according Joachim Bauch

Now we have to integrate our own ClientRegistry. We have to register it on Startup as a seperate bean at our WebApp. We have to modify red5-web.xml

```
<bean id="authClientRegistry" class="path.to.my.AuthClientRegistry" singleton="true" />
<bean id="web.context" class="org.red5.server.Context">
    <property name="scopeResolver" ref="red5.scopeResolver" />
    <property name="clientRegistry" ref="authClientRegistry" />
    <property name="serviceInvoker" ref="global.serviceInvoker" />
    <property name="mappingStrategy" ref="global.mappingStrategy" />
```

```
</bean>
```

Now simply extend the Client class to AuthClient. AuthClient serves some more methods according authentication.

We will now create instead of a Client() a new AuthClient()

```
IClient client =new AuthClient(authenticationInformation);
```

Its not the smoothest solution, but because many of us are using the username as an ID we have oppertunity to simply lookup all ids with the client registry, without creating extra hash maps. But from then its very important to do a second lookup if the id does not already exist before we create our new client.

In the application we could use a simple casting.

```
if (client is AuthClient)
{
    ((AuthClient)client).getAuthInfo().isAuthenticated(); //as a sample usage
}
```

Feature: We have not to lookup in the provided HashMap for the Authentication and we could even extend it for bigger programms.

23.1.1. Legal?

Ok, i dont know if this is a really _legal_ method of injecting clients because i dont know if its changing some logic of red5. Maybe someone of the core developers give a comment on this approach. But i can promise, it works and i use it in production. ;-)

Hope it help.

cu nomIad

This page describes the steps to configure and deploy your application on Red5 clustering [Documentation/Clustering/EdgeOriginSolutiononTerracotta].

In Red5 0.7 the Ant build.xml file contains a build target that creates a 'cluster' folder containing the same setup as described below. Use 'ant dist-cluster' to create the Red5 clustering setup.

= Limitations =

As of now, the current trunk only supports the clustering configuration for multiple Edges with one Origin. The Edge server only accepts RTMP connection.

= Server Configuration =

24.1. Configuration Files

There are several configuration files added to support Edge/Origin configuration.

red5-edge.xml, red5-edge-core.xml \-\- used for edge spring bean configuration. They are under conf/.

red5-origin.xml, red5-origin-core.xml \-\- used for origin spring bean configuration. They are under conf/.

24.2. Configure Edge Server

You don't need to deploy your application on Edges.

We strongly recommend you to deploy Edge on a different server from Origin. But it should be OK to deploy the Edge on the same server as Origin.

24.2.1. Edge on a different Server from Origin

Update the configuration of bean "mrtmpClient" in red5-edge-core.xml to point to Origin server:

```
<bean id="mrtmpClient"
    class="org.red5.server.net.mrtmp.MRTMPClient" init-method="start" >
    <property name="ioHandler" ref="mrtmpHandler" />
    <property name="server" value="YourOrigin" />
    <property name="port" value="${mrtmp.port}" />
</bean>
```

Replace red5.xml with red5-edge.xml. Start the server by

```
./red5.sh

or
```

```
java \-jar red5.jar 
```

24.2.2. Edge on the same Server as Origin

You don't need to change red5.xml. Copy red5-edge.xml to $(RED5_ROOT) from $(RED5_ROOT)/conf. Start the server by

```
java \-jar red5.jar red5-edge.xml

or update red5.sh to add a parameter "red5-edge.xml", then

./red5.sh 
```

24.3. Configure Origin Server

Deploy your application to webapps/. Make sure your 9035 port is not blocked by firewall. The port will be used by Edges to connection Origin.

Update red5.xml with red5-origin.xml. Start the server by

```
./red5.sh

or

java \-jar red5.jar
```

= Use Your Appliation =

Your RTMP can go through Edges now. Your RTMPT and HTTP can go through Origin as normal.

If you get server deadlock (freeze) here is what you can do to help us fix the problem.

1. Ensure you are running the latest trunk version. This is so we can be sure the problem has not already been fixed.

2. Make sure you start the server using red5.sh or red5.bat. This wil ensure the server is running under its own pid.

3. When the server freezes run the jstack command. You can find the process id using `ps -ef` (linux), `ps -aux` (osx) or task manager (win).

```
jstack <pid>
```

4. Create a Trac ticket, attach the output from jstack, and describe your environment and application.

This document explains in details how to setup a derby database connection using Hibernate [http://www.hibernate.org/] and caching. This document assumes you are using Red5 0.8 and the latest Hibernate Release 3.3.1.

Note

This update requires very latest Red5 svn trunk

Document below reference demo application http://red5.electroteque.org/dev/demos/hibernateTest.zip.

Sources for this application is here http://red5.electroteque.org/dev/demos/hibernateTest-src.zip

The demo application is built using the Red5Plugin, simply import the project into eclipse.

http://www.hibernate.org/ DAO takes a similar approach to the Spring DAO JDBC setup but adds first and second level caching and example is described here and Spring JDBC Hibernate [docs:Red5].

26.1. Required Libraries

- aspectjweaver.jar - Found in Spring with dependancies distribution.
- c3p0-0.9.1.2.jar
- dom4j-1.6.1.jar - Hibernate dependancy, found in distribution
- javassist-3.4.GA.jar - Hibernate dependancy, found in distribution
- antlr-3.1.1.jar - Hibernate dependancy, found in distribution. Already included with Red5.
- commons-collections-3.2.1.jar - Hibernate dependancy, found in distribution. Already included with Red5.
- jta-1.1.jar - Hibernate dependancy, found in distribution. Already included with Red5.
- slf4j-api-1.5.2.jar - Hibernate dependancy, found in distribution. Already included with Red5.
- ehcache-1.6.0-beta1.jar - The latest ehcache which removes backport concurrent dependancy.
- jsr107cache-1.0.jar - EHCache 1.6 dependancy.
- derby-10.4.2.0.jar
- hibernate3.jar - Get the latest which is 3.3.1.
- spring-aop-2.5.5.jar - Already included with Red5.
- spring-orm-2.5.5.jar - Already included with Red5.
- spring-jdbc-2.5.5.jar - Found in spring distribution.
- spring-tx-2.5.5.jar - Found in spring distribution.

Note

There is still an issue getting aspectjweaver.jar to resolve in the classpath which is needed for the AOP style of the transaction setup therefore it is required to be included in the classpath to the Red5 startup script like so

```
export RED5_CLASSPATH="${RED5_HOME}/boot.jar${P}${RED5_HOME}/conf${P}${CLASSPATH
```

26.2. Configuration

The following configurations explain how to configure an existing Red5 application configuration files. To learn about creating the application first visit Create New Applications In Red5 [Documentation/UsersReferenceManual/Red5CoreTechnologies/Chapter1].

26.2.1. red5-hibernate.xml

1. Create a config file called red5-hibernate.xml.
2. Add the following code

```
<xslthl:directive><?xml version="1.0" encoding="UTF-8"?></xslthl:directive>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-

    <bean id="hibernateProperties" class="org.springframework.beans.factory.config.PropertiesFactoryBe
  <property name="properties">
   <props>
    <prop key="hibernate.format_sql">true</prop>
    <prop key="hibernate.use_sql_comments">true</prop>
    <prop key="hibernate.generate_statistics">true</prop>
    <prop key="hibernate.cache.use_structured_entries">true</prop>

                <prop key="hibernate.bytecode.use_reflection_optimizer">true</prop>

                <!--  <prop key="hibernate.cache.provider_class">org.hibernate.cache.HashtableCachePro
    <prop key="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</prop>
    <prop key="hibernate.cache.provider_configuration_file_resource_path">hibernateTest-ehcache.xml</p
    <prop key="hibernate.cache.use_query_cache">true</prop>
    <prop key="hibernate.cache.use_second_level_cache">true</prop>
    <prop key="hibernate.dialect">org.hibernate.dialect.DerbyDialect</prop>
    <prop key="connection.autocommit">true</prop>
    <prop key="hibernate.show_sql">true</prop>
    <prop key="hibernate.generate_statistics">true</prop>
    <prop key="hibernate.hbm2ddl.auto">update</prop>
    <prop key="hibernate.show_sql">true</prop>
    <prop key="hibernate.c3p0.minPoolSize">5</prop>
    <prop key="hibernate.c3p0.maxPoolSize">20</prop>
    <prop key="hibernate.c3p0.timeout">600</prop>
    <prop key="hibernate.c3p0.max_statement">50</prop>
```

```
   <prop key="hibernate.c3p0.min_size">0</prop>
   <prop key="hibernate.c3p0.max_size">20</prop>
   <prop key="hibernate.c3p0.timeout">30</prop>
   <prop key="hibernate.c3p0.testConnectionOnCheckout">false</prop>
   <!--  <prop key="hibernate.query.substitutions">true 'T', false 'F'</prop>
   <prop key="hibernate.hbm2ddl.auto">update</prop>
   <prop key="hibernate.show_sql">true</prop>
   <prop key="hibernate.c3p0.minPoolSize">5</prop>
   <prop key="hibernate.c3p0.maxPoolSize">20</prop>
   <prop key="hibernate.c3p0.timeout">600</prop>
   <prop key="hibernate.c3p0.max_statement">50</prop>
   <prop key="hibernate.c3p0.testConnectionOnCheckout">false</prop>-->
  </props>
 </property>
</bean>

   <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       <property name="driverClassName" value="${db.driver}"/>
       <property name="url" value="${db.jdbcurl}"/>
       <property name="username" value="${db.username}"/>
 <property name="password" value="${db.password}"/>
   </bean>

   <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
 <property name="dataSource"><ref local="dataSource"/></property>
 <property name="hibernateProperties">
  <ref bean="hibernateProperties" />
 </property>
 <!-- Must references all OR mapping files. -->
 <property name="mappingResources">
  <list>
         <value>org/red5/server/webapps/hibernateTest/dao/Users.hbm.xml</value>
     </list>
 </property>
</bean>


   <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManage
      <property name="sessionFactory"><ref bean="sessionFactory"/></property>
   </bean>



<bean id="usersDao" class="org.red5.server.webapps.hibernateTest.dao.UsersDaoImpl">
 <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>


<bean id="usersService" class="org.red5.server.webapps.hibernateTest.dao.UsersServiceImpl">
 <property name="usersDao" ref="usersDao"/>
</bean>

<aop:config>
    <aop:pointcut id="usersServiceMethods" expression="execution(''' org.red5.server.webapps.hibernat
    <aop:advisor advice-ref="txAdvice" pointcut-ref="usersServiceMethods"/>
  </aop:config>

  <tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
      <tx:method name="get*" propagation="REQUIRED"/>
      <tx:method name="add*" propagation="REQUIRED"/>
      <tx:method name="delete*" propagation="REQUIRED"/>
    </tx:attributes>
 </tx:advice>


 <bean id="jmxExporter"
```

```
        class="org.springframework.jmx.export.MBeanExporter">
        <property name="beans">
            <map>
                <entry key="Hibernate:name=statistics">
                    <ref local="statisticsBean" />
                </entry>
            </map>
        </property>
    </bean>

    <bean id="statisticsBean" class="org.hibernate.jmx.StatisticsService">
        <property name="statisticsEnabled">
            <value>true</value>
        </property>
        <property name="sessionFactory"><ref local="sessionFactory"/></property>
    </bean>


</beans>
```

If you don't want to use AOP and setup the Spring 1.0 way it is like

```
<bean id="usersService"
      class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager" ref="transactionManager"/>
    <property name="target">
      <bean class="org.red5.server.webapps.hibernateTest.dao.UsersServiceImpl">
        <property name="usersDao" ref="usersDao"/>
      </bean>
    </property>
    <property name="transactionAttributes">
      <props>
        <prop key="get*">PROPAGATION_REQUIRED</prop>
        <prop key="add*">PROPAGATION_REQUIRES_NEW</prop>
        <prop key="delete*">PROPAGATION_REQUIRED</prop>
      </props>
    </property>
  </bean>
```

26.2.2. red5-web.xml

Note

There is no need to import configs anymore into red5-web.xml assuming it is prefixed with red5-* in the name.

26.2.3. red5-web.properties

We setup the database connection properties in the red5-web.properties file

```
db.driver=org.apache.derby.jdbc.EmbeddedDriver
db.jdbcurl=jdbc:derby:/path/to/db/server
db.username=dbuser
db.password=dbpass
```

26.2.4. hibernateTest-ehcache.xml

Note

This is needed to be placed into the classes directory to be loaded into the classpath

Add the following ehcache config to your classes location

```
<ehcache>
 <diskStore path="webapps/hibernateTest/cache" />

 <defaultCache
  maxElementsInMemory="4"
  eternal="false"
  timeToIdleSeconds="120"
  timeToLiveSeconds="120"
  overflowToDisk="false"
  diskPersistent="false"
  diskExpiryThreadIntervalSeconds="120"
  memoryStoreEvictionPolicy="LFU"
 />

 <cache name="org.hibernate.cache.StandardQueryCache"
      maxElementsInMemory="100"
      eternal="false"
      timeToIdleSeconds="120"
      timeToLiveSeconds="120"
      overflowToDisk="false"/>

  <cache name="org.hibernate.cache.UpdateTimestampsCache"
      maxElementsInMemory="5000"
      timeToIdleSeconds="120"
      timeToLiveSeconds="120"
      eternal="true"/>

 <cache name="org.red5.server.webapps.hibernateTest.dao.Users"
  maxElementsInMemory="1000"
  eternal="false"
  timeToIdleSeconds="120"
       timeToLiveSeconds="600"
  overflowToDisk="false" />
</ehcache>
```

26.2.5. StreamPoints.hbm.xml

This is the mapping file that hibernate will load to map objects to the database table.

```
<xslthl:directive><?xml version="1.0"?></xslthl:directive>


<xslthl:doctype><!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" ></xslthl:doctype>
```

```
<hibernate-mapping>
    <class
        name="org.red5.server.webapps.hibernateTest.dao.Users"
        table="users"
        dynamic-update="false"
        dynamic-insert="false">

        <cache usage="read-write" />

        <id
            name="id"
            column="id"
            type="java.lang.Long"
            unsaved-value="-1">
         <generator class="increment" />
        </id>


        <property
            name="userName"
            type="string"
            update="false"
            insert="true"
            column="username"
            length="100"
            not-null="true"

        />


    </class>


</hibernate-mapping>
```

26.3. Hibernate DAO Classes

26.3.1. Users.java

```
public class Users implements Serializable {

    private String userName;
    private Long id;


 public Users()
 {
 }

 public void setId(Long id)
 {
  this.id = id;
 }

 public Long getId() {
  return id;
 }

 public void setUserName(String value)
 {
```

```
  this.userName = value;
 }

  public String getUserName() {
   return userName;
  }

}
```

26.3.2. IUsersDAO.java

```
package org.electroteque.dao;

import java.util.List;

public interface IStreamPointsDao {

 List getStreamPoints();
 StreamPoints getStreamPoint(StreamPoints streamPoint);
 void deleteStreamPoint(StreamPoints streamPoints);
 void addStreamPoint(StreamPoints streamPoints);
}
```

26.3.3. IStreamPointsService.java

```
package org.red5.server.webapps.hibernateTest.dao;

import java.util.List;


public interface IUsersDao {

 List getUsers();
 Users getUser(Users users);
 void deleteUser(Users users);
 void addUser(Users users);
}
```

26.3.4. UsersDaoImpl.java

```
package org.red5.server.webapps.hibernateTest.dao;

import java.util.List;


import org.hibernate.HibernateException;
import org.hibernate.Session;

import org.hibernate.criterion.Example;
```

```
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;


public class UsersDaoImpl extends HibernateDaoSupport implements IUsersDao {


 public Users getUser(Users users) {
  Example example = Example.create(users);
  return (Users)this.getSessionFactory().getCurrentSession().createCriteria(Users.class)
  .add(example).list().iterator().next();
 }

 public List getUsers() {

      return this.getSessionFactory().getCurrentSession()
            .createQuery("FROM Users ORDER BY username")
            .list();
 }


 public void deleteUser(Users users)
 {
  this.getSessionFactory().getCurrentSession().delete(getUser(users));
 }

 public void addUser(Users users)
 {
  this.getSessionFactory().getCurrentSession().save(users);
 }

}
```

26.3.5. UsersServiceImpl.java

```
package org.red5.server.webapps.hibernateTest.dao;

import java.util.List;


public class UsersServiceImpl implements IUsersDao {


 private IUsersDao usersDao;

    public void setUsersDao(IUsersDao users) {
        this.usersDao = users;
    }

    public List getUsers() {
     return this.usersDao.getUsers();
    }

    public Users getUser(Users users) {
     return this.usersDao.getUser(users);
    }

 public Users getStreamPoint(Users users) {
  return this.usersDao.getUser(users);
 }

 public void deleteUser(Users users)
 {
```

```
  this.usersDao.deleteUser(users);
 }

 public void addUser(Users users)
 {
  this.usersDao.addUser(users);
 }


}
```

26.4. Usage

```
for (Iterator iter = this.usersService.getUsers().iterator(); iter.hasNext();) {
   Users element = (Users) iter.next();
   log.debug("User ID: {}, User Name: {}", new Object[]{element.getId(), element.getUserName()});
}

Users users = new Users();
users.setUserName("newuser");
this.usersService.addUser(users);
this.usersService.deleteUser(users);
```

26.4.1. Links

- Hibernate - http://www.hibernate.org
- C3P0 - http://sourceforge.net/projects/c3p0
- AspectJ - http://www.eclipse.org/aspectj/
- Backport Util Concurrent - http://backport-jsr166.sourceforge.net/ (required by EHCache)
- EHCache - http://ehcache.sourceforge.net/
- Derby - http://db.apache.org/derby/
- Spring Framework - http://www.springframework.org/
- Hibernate Connection Pooling - http://www.informit.com/articles/article.aspx?p=353736&seqNum=4
- Hibernate C3P0 Connection Pooling - http://www.hibernate.org/214.html
- Hibernate Code Generator tools for eclipse and ant - http://docs.jboss.org/tools/2.1.0.Beta1/hibernatetools/html/
- Spring ORM - http://static.springframework.org/spring/docs/2.5.x/reference/orm.html

The build system requires an Eclipse plugin for Ivyto be installed and setup in each development environment to make project dependencies easier for developers and the build system to manage. *The* *IvyDE* plugin may be installed through the standard Eclipse Install/Update (Help->Software Updates->Find and Install...) mechanism by adding the Ivy update site http://www.apache.org/dist/ant/ivyde/updatesite . This plugin allows management of dependencies through use of an "ivy.xml" file. While it does not completely take the place of the Eclipse classpath file, it does move the majority of library management to Ivy. Ivy will pull dependencies from an internal repository and then if not found it will look in our google code repository.

Ivy is dependent upon a configuration (ivysettings.xml) file to define repository locations. After installing the IvyDE Eclipse plugin within Eclipse go to *"Window->Preferences->Ivy"* set the* *URL to point to the "ivysettings.xml" file. This file tells Ivy where to locate our dependency libraries. After pointing to this configuration file the IvyDEplugin will be able to retrieve all defined dependencies. Note that if the Ivy plugin option to "Do a retrieve after resolve" is checked then dependencies will be copied to the folder defined in the Pattern text box after being downloaded to your local cache. This is not necessary since the Ivy plugin will point to your local Ivy cache in the classpath and the build scripts are designed to copy the dependencies as needed.If you do not have a need to execute Ant build scripts from within Eclipse, but would like to ignore build file problems you may set the preference to ignore build file problems under "Window->Preferences->Ant->Editor->Problems tab".

If you want to execute the Ant build scripts from within Eclipse you will first need to change your Eclipse Ant runtime to point to Ant 1.7 or newer.

After installing the IvyDEplugin into Eclipse it may give error popups when trying to retrieve inter-project dependencies. This is due to the dependency being defined on a packaged version of the project. If the dependency is upon a "latest.integration" revision and you haven't deployed the dependent project to your local repository then you'll get the popup. This error is also caused when the dependency is upon a "latest.release" revision and a release has not yet been published to the internal archive. You can either ignore the popups, which will not cause any problems in development since a dependency on the raw project should be defined in your Eclipse classpath, or choose to publish the dependency locally by executing the "publish.local" Ant build target of the project. This may be remedied in the future when/if our continuous integration server publishes integration snapshots to our internal archive.

Lastly, here are a couple related screen casts which will help:

- How to setup the IvyDE and how to remove the older version http://gregoire.org/2008/08/30/ivyde-part-1/
- How get libraries to resolve properly http://gregoire.org/2008/09/06/ivyde-part-2/

27.1. Update

Ivy Repository keeps changing, the new update site has been updated however, there is some issues with the downloaded jar therefore please download this jar for the ivyide plugin http://code.google.com/p/red5/source/browse/repository/ivy/ivyde-eclipse.jar and place it in /path/to/eclipse/plugins/org.apache.ivyde.eclipse_2.0.0.beta1 for now.

To Refresh Ivy dependancies:

```
ant ivyclear
ant dist
```

To resolve the dependancies run as described in part 2 of Paul's blog

```
ant -Divy.conf.name="java6, eclipse" dist
```

The very recent update of Java 6 for OSX 10.5 has some problems with the Java Preferences tool. These changes do not have any effect therefore the symlinks need to be done manually.

Once the update had happened it changed the Java 6 SDK target back to Java 5 SDK and had to be changed back manually.

```
cd /System/Library/Frameworks/JavaVM.framework/Versions/
rm CurrentJDK
ln -s 1.6 CurrentJDK
```

Restart terminal then run the java version check to confirm it's back to 1.6

```
javac -version
```

Should be

```
javac 1.6.0_13
```

And then

```
java -version
```

Should be

```
java version "1.6.0_13"
Java(TM) SE Runtime Environment (build 1.6.0_13-b03-211)
Java HotSpot(TM) 64-Bit Server VM (build 11.3-b02-83, mixed mode)
```

29.1. Overview

I am pretty new to red5 and had some trouble getting any of my jruby apps to work with it. The only other example that I could find was the scripting how-to guide and I personally think it is geared more towards seasoned red5 developers. So this will be my attempt at making jruby scripting and scripting in general with red5 a little easier for newbs such as my self. This tutorial assumes that you have the latest version of red5 from trunk installed and working correctly. I also use apache's ant to build the more complicated example.

29.2. Basic Application

The first example is going to assume that you have no java services defined and only have the main application(main.rb). This will cover the required directory structure and required files(took me a long time to figure this out). This may seem very simple to more seasoned red5 developers but it is major hang-up for new users to the scripting features in red5. Also, I am using java version 1.6.0_02 but this first example should work with any version that supports scripting. However, I have read that there are issues with older versions and red5 scripting.

29.2.1. Webapp Directory Structure

As you may already be aware, all applications should be located in the red5/webapps directory. This seems really simple now that I am aware of what is required but at first I had no clue what my basic directory structure should be. Also, please note that we will use a slightly different process for applications with java services and I will explain that in more detail later on in the tutorial. In this tutorial red5 will represent the red5 root directory. I will also use the webapp name of red5Jruby throughout this tutorial.

The basic configuration files can be found in red5/doc/templates/myapp . To make things easier you can copy the contents of that directory to red5/webapps/yourApp

```
red5/    #red5 root directory
...webapps/    #red5 web applications directory
.....red5Jruby/    #our application
.........WEB-INF/     #contains all of our application's  files i.e. configs and application sources
.............classes/    #contains our complied java classes and scripting sources
...................applications/   #contains our scripting sources
.........................main.rb     #our jruby implementation of the red5 application adapter
.............lib/    #our application's lib directory
.................slf4j-api-1.4.3.jar   #core api. This jar should be located in red5/lib. You will need to copy it
.............web.xml&amp;nbsp; #our application settings. This file is located in red5/doc/templates/myapp/W
.............red5-web.xml #our application's web handler and services definitions
.............red5-web.properties #our application's vitualHosts and context path
```

The configuration files should be pretty easy to understand. The only one that I am going to cover in this tutorial is the red5-web.xml. Which contains our web handler and services definitions. Below is an excerpt from the red5-web.xml that defines our application web handler. Please see the scripting how-to guide for more information.

```
#red5/webapps/red5Jruby/WEB-INF/red5-web.xml
<!-- Our application web handler -->
      <bean id="web.handler" class="org.springframework.scripting.jruby.JRubyScriptFactory">
 <constructor-arg index="0" value="classpath:applications/main.rb"/>
 <constructor-arg index="1">
  <list>
   <value>org.red5.server.api.IScopeHandler</value>
   <value>org.red5.server.adapter.IApplication</value>
  </list>
 </constructor-arg>
</bean>
```

29.2.2. Ruby application adapter

I am going to use the main.rb from the olfaDemo as it is a good basic example. In later tutorials I will explain the application adapter in more detail. The focus of this tutorial is to get a basic application setup and working.

```
# JRuby - style
require 'java'
module RedFive
   include_package "org.red5.server.api"
 include_package "org.red5.server.api.stream"
 include_package "org.red5.server.api.stream.support"
 include_package "org.red5.server.adapter"
 include_package "org.red5.server.stream"
end

#
# application.rb - a translation into Ruby of the ofla demo application, a red5 example.
#
# @author Paul Gregoire
#
class Application < RedFive::ApplicationAdapter

   attr_reader :appScope, :serverStream
 attr_writer :appScope, :serverStream

 def initialize
   #call super to init the superclass, in this case a Java class
   super
   puts "Initializing ruby application"
 end

 def appStart(app)
       puts "Jruby application started"
```

```
 @appScope = app
 return true
end

def appConnect(conn, params)
 puts "Ruby appConnect"
 measureBandwidth(conn)
 puts "Ruby appConnect 2"
 if conn.instance_of?(RedFive::IStreamCapableConnection)
     puts "Got stream capable connection"
  sbc = RedFive::SimpleBandwidthConfigure.new
  sbc.setMaxBurst(8388608)
  sbc.setBurst(8388608)
  sbc.setOverallBandwidth(8388608)
  conn.setBandwidthConfigure(sbc)
 end
 return super
end

def appDisconnect(conn)
 puts "Ruby appDisconnect"
 if appScope == conn.getScope && @serverStream != nil
  @serverStream.close
 end
 super
end

def toString
 return "Ruby toString"
end

   def setScriptContext(scriptContext)
   puts "Ruby application setScriptContext"
   end

   def method_missing(m, *args)
     super unless @value.respond_to?(m)
     return @value.send(m, *args)
   end

end
```

That is all that is required for a basic jruby application in red5.

29.3. More complicated example

Next I will explain a more complicated example that has defined services. I will use the oflaDemo as the example and will include a build file that will compile our application. As

I said before, the purpose of this tutorial is to show you how to create and setup a basic application. I will go more into detail in later tutorials.

First we will need to create a new directory under red5/webapps. I will also refer to this one as red5Jruby. Once this is completed you will need to obtain the sources for the oflaDemo. If you complied red5 from the latest trunk version, they should be located in red5-trunk/webapps/oflaDemo. Now copy the contents of the oflaDemo directory into our new red5Jruby directory. Once that is completed you will then need to change the references to oflaDemo to red5Jruby in the configuration files. At this time you can also remove the handler and services definitions for java and the other scripting languages from the red5-web.xml.

You should now have a directory structure that looks like this:

```
red5/
...webapps/
......WEB-INF/
.........src/ #our application sources; both java and jruby
...........applications/ #our scripting sources
...........org/
.............red5/
...............server/
.................webapp/
...................oflaDemo/  #you will need to rename this directory to red5Jruby
.....................Application.java
.....................DemoService.java
.....................IDemoService.java
.....................DemoServiceImpl.java
.........logback.xml
.........build.xml
.........build.properties
```

You will now need to change the following package definition to suit our new application in all of the java sources.

```
package org.red5.server.webapp.oflaDemo;
```

to

```
package org.red5.server.webapp.red5Jruby;
```

Also, you will need to change the logger definition in red5/webapps/red5Jruby/WEB-INF/logback.xml to match our application ie org.red5.server.webapp.red5Jruby.

29.3.1. Build sources

I pieced this together from red5/build.xml the comments should explain it pretty well

build.xml

```
<?xml version="1.0"?>
<project name="Complie Red5 webapp" basedir="." default="build">

<!-- Project properties -->
 <property name="root.dir" value="../../../"/>
 <property name="root.classes.dir" value="${root.dir}/bin"/>
 <property name="root.lib.dir" value="${root.dir}/lib"/>
 <property name="src.dir" value="src"/>
 <property name="classes.dir" value="classes"/>
 <property name="lib.dir" value="lib"/>
 <property name="target.jar" value="red5Jruby.jar"/>
<!-- End Project properties -->

<!-- webapps Classpath -->
 <path id="webapps.classpath">
    <fileset dir="${root.dir}">
    <filename name="red5.jar"/>
   </fileset>
   <fileset dir="${root.lib.dir}">
    <filename name="*.jar"/>
   </fileset>
   <pathelement location="${root.classes.dir}"/>
 </path>
<!-- End webapps Classpath -->

<!-- Clean webapp -->
 <target name="clean" description="Cleans WEBAPP dir">
  <echo message="Deleting the classes and lib directories...." />
  <delete dir="${classes.dir}"/>
  <delete dir="${lib.dir}"/>
 </target>
<!-- End Clean webapp -->

<!-- Prepare webapp -->
 <target name="prepare" depends="clean">
  <echo message="Creating the classes and lib directories...." />
  <mkdir dir="${lib.dir}"/>
  <mkdir dir="${classes.dir}"/>
  <mkdir dir="${classes.dir}/applications"/>
 </target>
<!-- End Prepare webapp -->

<!-- Build webapp -->
 <target name="build" depends="prepare">
  <echo message="Compiling application sources...." />
  <javac
```

```
   sourcepath=""
   srcdir="${src.dir}"
   destdir="${classes.dir}"
   classpathref="webapps.classpath"
      optimize="true"
      verbose="false"
      fork="true"
      nowarn="true"
      deprecation="false"
      debug="true"
      compiler="modern"
      source="1.6"
      target="1.6"
  />
  <echo message="Copying scripting sources..." />
  <copy todir="${classes.dir}/applications">
   <fileset dir="${src.dir}/applications"/>
  </copy>

  <copy todir="${classes.dir}" file="${src.dir}/logback.xml" overwrite="true"/>
  <copy todir="${lib.dir}">
   <fileset dir="${root.lib.dir}">
    <include name="slf4j-api-1.4.3.jar"/>
    <include name="logback-core-0.9.8.jar"/>
    <include name="logback-classic-0.9.8.jar"/>
   </fileset>
  </copy>
 </target>
<!-- End Build webapp -->


<!-- Create Jar -->
 <target name="jar" description="Make Archive" depends="build">
  <echo message="Creating Jar" />
  <jar destfile="${lib.dir}/${target.jar}">
   <fileset dir="${classes.dir}">
    <include name="**"/>
   </fileset>
  </jar>
 </target>
<!-- End Create Jar -->

</project>
```

You can use this same build file for future applications. However, you will need to change the target.jar property from red5Jruby.jar to yourAppName.jar

build.properties(not really used in this example)

```
# Base build properties
#

#javac options
# http://ant.apache.org/manual/CoreTasks/javac.html
# sun javac
build.compiler=javac
# jikes
#build.compiler=jikes

# generic compiler options
build.verbose=false
build.fork=true
build.deprecation=false
build.nowarn=true

# optimize only works with a few compilers
build.optimize=true

# Change this var to build to a different Java version
java.target_version=1.6

# jikes options
build.compiler.emacs=false
build.compiler.fulldepend=false
build.compiler.pedantic=false
```

We are almost done now\! All we have left is to acutally build the application and start/restart the server. You can do this by navigating to red5/webapps/red5Jruby/WEB-INF and typing the following into the command prompt

```
ant jar
```

This will compile the application sources, create the required application directory structure and create a jar containing the contents of the newly created classes directory. At this point you should be able to restart the server and you should see the following in the terminal

```
Initializing ruby application
Ruby appStart
```

Any suggestions or clarifications on any topics covered in this tutorial are more then welcome.

Enjoy, Thomas Johnson(tomfmason)

Have yet to work it out doing it from subversion in Eclipse. Inside a terminal at the top level of the red5 directory run something like this to create a patch of a changed file for submitting into Trac.

```
svn diff src/org/red5/server/util/FileUtil.java > FileUtil.txt
```

This will create a patch file which can then be submitted as a file upload into a ticket.

Ping is the most mysterious message in RTMP and till now we haven't fully interpreted it yet. In summary, Ping message is used as a special command that are exchanged between client and server. This page aims to document all known Ping messages. Expect the list to grow.

The type of Ping packet is 0x4 and contains two mandatory parameters and two optional parameters. The first parameter is the type of Ping and in short integer. The second parameter is the target of the ping. As Ping is always sent in Channel 2 (control channel) and the target object in RTMP header is always 0 which means the Connection object, it's necessary to put an extra parameter to indicate the exact target object the Ping is sent to. The second parameter takes this responsibility. The value has the same meaning as the target object field in RTMP header. (The second value could also be used as other purposes, like RTT Ping/Pong. It is used as the timestamp.) The third and fourth parameters are optional and could be looked upon as the parameter of the Ping packet. Below is an unexhausted list of Ping messages.

- type 0: Clear the stream. No third and fourth parameters. The second parameter could be 0. After the connection is established, a Ping 0,0 will be sent from server to client. The message will also be sent to client on the start of Play and in response of a Seek or Pause/Resume request. This Ping tells client to re-calibrate the clock with the timestamp of the next packet server sends.
- type 1: Tell the stream to clear the playing buffer.
- type 3: Buffer time of the client. The third parameter is the buffer time in millisecond.
- type 4: Reset a stream. Used together with type 0 in the case of VOD. Often sent before type 0.
- type 6: Ping the client from server. The second parameter is the current time.
- type 7: Pong reply from client. The second parameter is the time when the client receives the Ping.

New Changes have been made to the RTMPClient in 0.8 which allows for more flexibility.

RTMPClient can be either extended or composed into another class. Both ways need to implement some callback interfaces for handling callback methods for stream events and exception handling

- http://dl.fancycode.com/red5/api/org/red5/server/api/service/IPendingServiceCallback.html - For handling connection events
- http://dl.fancycode.com/red5/api/org/red5/server/net/rtmp/RTMPClient.INetStreamEventHandler.html - For handling stream events
- http://api.red5.nl/org/red5/server/net/rtmp/ClientExceptionHandler.html - For handling connection exceptions
- http://api.red5.nl/org/red5/server/api/event/IEventDispatcher.html - For handling the NetStream events

32.1. Extending RTMPClient

```
import org.red5.server.api.service.IPendingServiceCallback;
import org.red5.server.net.rtmp.ClientExceptionHandler;
import org.red5.server.net.rtmp.INetStreamEventHandler;
import org.red5.server.api.event.IEventDispatcher;

public class RTMPClientExtended extends RTMPClient implements INetStreamEventHandler, IPendingServiceC
{
....
}
```

32.2. RTMPClient Service Callback Handlers

The handlers need to be setup to a class like so

```
rtmpClient.setServiceProvider(this);
rtmpClient.setExceptionHandler(this);
rtmpClient.setCallbackProvider(this);
rtmpClient.setStreamEventDispatcher(this);
```

32.3. Callback Result Handlers

32.3.1. Callback Result Handler

The resultReceived method determines connections status, it works similar to the Flash status event

```
synchronized public void resultReceived(IPendingServiceCall call) {
      Object result = call.getResult();

  if (result instanceof ObjectMap)
  {

   if (StatusCodes.NC_CONNECT_SUCCESS.equals(code))
   {
    log.info("Connected");
    state = RTMPStreamState.STREAM_CREATING;
    createStream(this);

   }
               }
.....
                else {

    if ("createStream".equals(call.getServiceMethodName())) {

   state = RTMPStreamState.PLAYING;

   if (result instanceof Integer) {
    Integer streamIdInt = (Integer) result;
    streamId = streamIdInt.intValue();
    log.debug("createStream result stream id: " + streamId);
    play(streamIdInt, fileName, start, duration);
   } else {
    disconnect();
    state = RTMPStreamState.STOPPED;
   }

    }
 }
}
```

32.3.2. Callback Stream Status Handler

Where createStream(this); sets up the NetStream for playing back and streaming. The next call will trap creatStream and then be able to play back the file by first collecting the streamId.

The stream event status is setup like so

```
public void onStatus(Object obj)
{
   ObjectMap map = (ObjectMap) obj;
   String code = (String) map.get("code");
   String description = (String) map.get("description");
   String details = (String) map.get("details");

   if (StatusCodes.NS_PLAY_START.equals(code))
   {
      .......

   }
}
```

32.3.3. Callback Error Handler

The client exception handler for handling the connection errors is like so

```
public void handleException(Throwable throwable)
{
  log.error("{}",new Object[]{throwable.getCause()});
}
```

32.3.4. Dispatch Event Handler

For handling the NetStream events during streaming playback which lets you trap the packets of the stream it is like so

```
public void dispatchEvent(IEvent event) {
        if (!(event instanceof IRTMPEvent)) {
            log.debug("skipping non rtmp event: " + event);
            return;
        }
        IRTMPEvent rtmpEvent = (IRTMPEvent) event;
        /*
        if (log.isDebugEnabled()) {
            log.debug("rtmp event: " + rtmpEvent.getHeader() + ", "
                    + rtmpEvent.getClass().getSimpleName());
        } */
        if (!(rtmpEvent instanceof IStreamData)) {
            log.debug("skipping non stream data");
            return;
        }
        if (rtmpEvent.getHeader().getSize() == 0) {
            log.debug("skipping event where size == 0");
            return;
        }
        ITag tag = new Tag();
        tag.setDataType(rtmpEvent.getDataType());
        if (rtmpEvent instanceof VideoData) {
            videoTs += rtmpEvent.getTimestamp();
            tag.setTimestamp(videoTs);
        } else if (rtmpEvent instanceof AudioData) {
            audioTs += rtmpEvent.getTimestamp();
            tag.setTimestamp(audioTs);
        }

        ByteBuffer data = ((IStreamData) rtmpEvent).getData().asReadOnlyBuffer();
        tag.setBodySize(data.limit());
        tag.setBody(data);
        //log.debug(data.toString());
        try {
            writer.writeTag(tag);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
 }
```

#!div class=noteboxnot
The connect timeout is still hardcoded in RTMPClient unless the entire code is redone in a extended

32.4. Examples

32.4.1. Playback Example

```
synchronized public void resultReceived(IPendingServiceCall call) {
      Object result = call.getResult();

  if (result instanceof ObjectMap)
  {

   if (StatusCodes.NC_CONNECT_SUCCESS.equals(code))
   {
    log.info("Connected");
    state = RTMPStreamState.STREAM_CREATING;
    createStream(this);

   }
               }
.....
                else {

    if ("createStream".equals(call.getServiceMethodName())) {

   state = RTMPStreamState.PLAYING;

   if (result instanceof Integer) {
    Integer streamIdInt = (Integer) result;
    streamId = streamIdInt.intValue();
    log.debug("createStream result stream id: " + streamId);
                              play(streamId, "stream.flv", 0);
   } else {
    disconnect();
    state = RTMPStreamState.STOPPED;
   }

    }
 }
}
```

32.4.2. Publish Example

```
synchronized public void resultReceived(IPendingServiceCall call) {
      Object result = call.getResult();

  if (result instanceof ObjectMap)
  {

   if (StatusCodes.NC_CONNECT_SUCCESS.equals(code))
   {
    log.info("Connected");
    state = RTMPStreamState.STREAM_CREATING;
    createStream(this);
```

```
   }
              }
.....
               else {

    if ("createStream".equals(call.getServiceMethodName())) {

   state = RTMPStreamState.PLAYING;

   if (result instanceof Integer) {
    Integer streamIdInt = (Integer) result;
    streamId = streamIdInt.intValue();
    log.debug("createStream result stream id: " + streamId);
                               publish("publishtest", streamId, "live", this);
   } else {
    disconnect();
    state = RTMPStreamState.STOPPED;
   }

    }
 }
}
```

```
public void onStatus(Object obj)
{
   ObjectMap map = (ObjectMap) obj;
   String code = (String) map.get("code");
   String description = (String) map.get("description");
   String details = (String) map.get("details");

   if (StatusCodes.NS_PUBLISH_START.equals(code))
   {
 state = RTMPStreamState.PUBLISH_START;
 log.debug("{} for {}", new Object[]{code,details});

        service = new FLVService();
  service.setSerializer(new Serializer());
  service.setDeserializer(new Deserializer());

  log.info("Started Publishing");

               // Read In File And Publish The Data !!
  try {

   File f = new File(dir + "/" + fileName);
   log.debug("test: {}", f);

   IFLV flv = (IFLV) service.getStreamableFile(f);
   flv.setCache(NoCacheImpl.getInstance());

   ITagReader reader = flv.getReader();

   FileStreamSource src = new FileStreamSource(reader);

   while (src.hasMore())
   {
    IRTMPEvent event = src.dequeue();
    RTMPMessage rtmpMsg = new RTMPMessage();
                rtmpMsg.setBody(rtmpEvent);

                publishStreamData(streamId, rtmpMsg);
   }
```

```
   this.getRTMPClient().unpublish(streamId);

  } catch (Exception ex) {
   log.error(ex.getCause().toString());
  }

   }
}
```

32.4.3. Shared Object Example

```
synchronized public void resultReceived(IPendingServiceCall call) {
      Object result = call.getResult();

  if (result instanceof ObjectMap)
  {

   if (StatusCodes.NC_CONNECT_SUCCESS.equals(code))
   {
    log.info("Connected");
    state = RTMPStreamState.STREAM_CREATING;
    createStream(this);

                                IClientSharedObject obj = getSharedObject("server", true);
          obj.addSharedObjectListener(this);

                                Map<String,Object> map = new HashMap<String,Object>();
                                map.put("key","value");

         obj.beginUpdate();
        obj.setAttributes(map);
       obj.endUpdate();
       obj.clear();


   }
              }
.....
               else {

    if ("createStream".equals(call.getServiceMethodName())) {

   state = RTMPStreamState.PLAYING;

   if (result instanceof Integer) {
    Integer streamIdInt = (Integer) result;
    streamId = streamIdInt.intValue();
    log.debug("createStream result stream id: " + streamId);

   } else {
    disconnect();
    state = RTMPStreamState.STOPPED;
   }

    }
 }
}
```

32.4.4. Sharedobject Listeners

```
public void onSharedObjectClear(ISharedObjectBase so)
 {
  log.debug("Shared Object Clear");
 }

 public void onSharedObjectConnect(ISharedObjectBase so)
 {
  log.debug("Shared Object Connect");
 }

 public void onSharedObjectDelete(ISharedObjectBase so, String key)
 {
  log.debug("Shared Object Delete");
 }

 public void onSharedObjectDisconnect(ISharedObjectBase so)
 {
  log.debug("Shared Object Disconnect");
 }

 public void onSharedObjectSend(ISharedObjectBase so, String method, List params)
 {
  log.debug("Shared Object Send");
 }

 public void onSharedObjectUpdate(ISharedObjectBase so, IAttributeStore values)
 {
  log.debug("Shared Object Update");
 }

 public void onSharedObjectUpdate(ISharedObjectBase so, Map<String,Object> values)
 {
  log.debug("Shared Object Updae");
 }

 public void onSharedObjectUpdate(ISharedObjectBase so, String key, Object value)
 {
  log.debug("Shared Object Updae");
 }
```

32.4.5. Callback Example

```
synchronized public void resultReceived(IPendingServiceCall call) {
      Object result = call.getResult();

  if (result instanceof ObjectMap)
  {

   if (StatusCodes.NC_CONNECT_SUCCESS.equals(code))
   {
    log.info("Connected");
    state = RTMPStreamState.STREAM_CREATING;
    createStream(this);

                                invoke("service.CallService", new Object[]{"arg1","arg2"}, this);
```

```
   }
               }
.....
                else {

    if ("createStream".equals(call.getServiceMethodName())) {

   state = RTMPStreamState.PLAYING;

   if (result instanceof Integer) {
    Integer streamIdInt = (Integer) result;
    streamId = streamIdInt.intValue();
    log.debug("createStream result stream id: " + streamId);

   } else {
    disconnect();
    state = RTMPStreamState.STOPPED;
   }

    }
 }
}
```

33.1. Overview

This page explains **none, #990000 , Scope) [[Color(]]** and **none, #990000 , Context) [[Color(]]** in Red5. The Scope model in Red5 that supports Application model is an extension to the Application model in FMS. The Context model in Red5 has no counterpart in FMS. These two concepts are Red5 specific.

33.2. Concept of Scope

Resources are managed in Red5 in a tree. Each node of a tree is called a scope. If the scope is a leaf node, it is called a BasicScope and if the scope contains child scopes, it is called a Scope. There're two pre-defined BasicScopes in Red5, SharedObject Scope and BroadcastStream Scope.

Each Application has its own Scope hierarchy and the root scope is WebScope. There's a global scope defined in Red5 that aims to provide common resource sharing across Applications namely GlobalScope. The GlobalScope is the parent of all WebScopes. Other scopes in between are all instances of Scope. Each scope takes a name. The GlobalScope is named "default". The WebScope is named per Application context name. The Scope is named per path name. The SharedObject Scope is named per SharedObject's name. The BroadcastStream Scope is named per Stream's name.

Except GlobalScope and BasicScopes, all Scopes can be connected by a client. A Scope object might be created as a result of a connection request from a client. For example, a client could issue a request to connect to oflaDemo/room0 when the room0 scope does not exist. After the establishment of the connection, room0 is created. If the url contains many intermediate scopes, all these scopes will be created. For example, oflaDemo/lobby0/room0 is requested and neither lobby0 or room0 exist. lobby0 and room0 will be created accordingly. Then the connection is tied to room0 scope.

A typical scope hierarchy could be like this:

```
GlobalScope(default) --> WebScope(oflaDemo) --> Scope(room0) --> BroadcastStream(live0), Shar
                                            --> Scope(room1) --> SharedObject(so0)
                     --> WebScope(fitcDemo) --> Scope(room1) --> BroadcastStream(live0)
```

For standalone version, the global scope object is defined in webapps/red5-default.xml. For WAR version, the global scope object is defined together with web scope object in conf/war/applicationContext.xml.

33.3. Scope Functionality

Apart from the tree structure iteration, Scope provides the resource resolving and service registration capabilities. It also manages a set of connection objects that are currently connected to it. A ScopeHandler can be injected into a Scope so that the activity of Scope can be monitored, including the creation, destruction and connection of Scope.

33.4. Context

A Context is stuck to a Scope object and provides additional services to the scope object. Context objects can be obtained by calling IContext.getScope(). Context wraps the spring application context so that the services can be declared as spring beans and looked up from Context. Other services include "clientRegistry", "serviceInvoker", "persistenceStore", "mappingStrategy" and resource resolver that backs the resource resolver provided in Scope. For more information of Context, please refer to org.red5.server.api.IContext.

Context can be inherited. This means a Scope may not define a context and instead use its parent's context directly. Only GlobalScope and WebScope use their own Context object.

Hi,

Im writing this, because in the actual mailing list there are some requests on DB Connections.

There was a solution posted some time ago but there is a better way to solve this common problem.

In this article I will explain how to implement a good way for accessing a database without Hibernate.

Actually we are working with the Spring framework [http://www.springframework.org]. The cool thing about Spring is that we can implement "shared used objects" on the server.

= The fast way =

```
new com.mysql.jdbc.Driver();
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost/test?" + //get the connection
                                    "user=u&password=pass");
ResultSet rs = null;
Statement stmt = null;
try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT * from tatest");

    if (rs.next())
    {
        log.debug(rs.getString(2));
    }
} finally {
     if (rs != null) {
            try {
                rs.close();
            } catch (SQLException sqlEx) {  }

          rs = null;
       }
    }
}
```

= A better aproach =

Ok, the solution above works quite fast if you test it, but Red5 is a webserver that serves content for many people. We need to implement ConnectionPooling that speeds up the connectiontime and reduces object creation. Additional we want to gerenalize our code so it can be used with any other database.

For this solution we can use Spring. We will store our "datasource" as a bean available for every connection (its threadsave).

We start to setup our project. We create an application described in the "New Applications.txt" [http://www.joachim-bauch.de/tutorials/red5/HOWTO-NewApplications.txt] by Joachim. In the WEB-INF folder we create an additional folder called "lib". This directory serve our .jar files we need to use.

We need following libraries:

Setup an eclipse project described in the other tutorials. spring-jdbc.jar spring-dao.jar commons-dbcp-1.2.1.jar (mysql-connector-java-5.0.5-bin.jar

(UPDATE20110126: spring-dao.jar -> org.springframework.transaction-3.0.5.RELEASE.jar; commons-dbcp-1.4; mysql-connector-java-5.1.14)

We will now add following lines to our red5-web.xml.

```
<bean id="idDataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource"
 <property name="driverClassName" value="${db.driver}" />
 <property name="url" value="${db.url}" />
 <property name="username" value="${db.username}" />
 <property name="password" value="${db.password}" />
</bean>
```

In the red5-web.properties we add the following

```
db.driver=com.mysql.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/test
db.username=user
db.password=pass
```

This is the standard setup for our database access. (You have to modify this values according your DB configuration)

Now we can write our main code in the Red5 ApplicationAdapter

```
public String getSampleString()
{
 //Getting the datasource bean
 Object o=Red5.getConnectionLocal().getScope().getContext().getBean("idJdbcTemplate");
 JdbcTemplate t=(JdbcTemplate)o;

 //make a simple select and use a mapper for mapping it on your objects
 final List l=t.query("SELECT * FROM tatest;",new RowMapper(){
  public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
   return new MappedRow(rs.getInt(1),rs.getString(2));
  }
 });
```

```
 //finally loop through all values and concat them
 final Iterator<MappedRow> i=l.iterator();
 String s="";
 while(i.hasNext())
 {
  s+=i.next().getName()+";"; //dirty, just for less code
 }
 //return the result for an function
 return s;

}
```

Here is the code for the MappedRow

```
public class MappedRow {
 protected int id;
 protected String name;

 public MappedRow(int _id,String _name) {
  id=_id;
  name=_name;
 }
 public int getId() {
  return id;
 }
 public String getName() {
  return name;
 }
}
```

Finally we can use a ConnectionPool able datasource like org.apache.commons.dbcp.BasicDataSource

Very easy implementation, just delete the idDataSource bean and replace it with this

```
<bean id="idDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
 <property name="driverClassName"><value>${db.driver}</value></property>
 <property name="url"><value>${db.url}</value></property>
 <property name="username"><value>${db.username}</value></property>
 <property name="password"><value>${db.password}</value></property>
 <property name="poolPreparedStatements"><value>true</value></property>
 <property name="maxActive"><value>10</value></property>
 <property name="maxIdle"><value>10</value></property>
</bean>
```

Finally we are able with this code to work database independent, with connection pooling and the use of a threadsave bean. The result is reduced and generalized code. Automatic object mapping and xml config.

cu nomIad

Updated for 0.8.0

35.1. Preface

This document describes how to deploy Red5 to Tomcat as web application archive (WAR). The standard Red5 deployment consists of a standalone Java application with an embedded JEE container (Jetty or Tomcat) running as a system service, whereas the WAR version runs inside of a JEE container.

We have a preference for Tomcat, but it is possible to use the WAR with another JEE container.

35.2. Deployment

The Tomcat war deployer scans the webapps directory for wars periodically. When a war is found that has not yet been deployed, the deployer will expand the war file into a directory based on the filename of the war. A war named myapp.war would be expanded into a directory named myapp; depending upon your installation the full path would look similar to this C:\Tomcat-6.0.18\webapps\myapp.

Red5 server is packaged into a file named ROOT.war, this filename has a special connotation on most JEE application servers and is normally the default or root web context. The root web context is responsible for servicing requests which do not contain a path component. A url with a path component looks like http://www.example.com/myapp wheres root web application url would resemble this http://www.example.com/. An additional configuration file the context descriptor, is located in the META-INF directory for each web context. Applications that are not accessed via HTTP, do not require a web / servlet context. The root war file contains nearly everything that is in a standalone server build except for embedded server classes and select configuration files.

The ROOT.war may be renamed to red5.war, but this has not been tested and would require changes to some of the configuration files. Remember that this name and its resolution within the JEE container is primarily for HTTP requests and should not affect RTMP communications.

35.2.1. Context descriptors

A Context XML descriptor is a fragment of XML data which contains a valid Context element which would normally be found in the main Tomcat server configuration file (conf/server.xml). For a given host, the Context descriptors are located in $CATALINA_HOME/conf/[enginename]/[hostname]/. Note that while the name of the file is not tied to the webapp name, when the deployer creates descriptors from the context.xml files contained in the war; their names will match the web application name.

Context descriptors allow defining all aspects and configuration parameters of a context, such as naming resources and session manager configuration. It should be noted that the docBase specified in the Context element can refer to either the .WAR or the directory which will be created when the .WAR is expanded or the .WAR itself.

35.3. Red5 Configuration

Configuration of the Red5 server consists of a few context parameters in the web.xml, a default context file, a bean ref file, and a Spring web context file for each application that will utilize Red5 features. Web applications that use only AMF to communicate with Red5 do not require a configuration entry in the servers application context. The application context which is managed via Spring is only available to applications that are contained within the root war; due to the way that the web application classloaders work. In addition, Red5 uses a context counterpart called a Scope which serves as a container for the context, handler, server core instance, and a few other objects. A scope is similar to the application model in FMS.

The initial entry point or startup servlet for Red5 is the WarLoaderServlet and it is configured as a servlet listener in the web.xml as shown below. Functionally this servlet takes the place of the Bootstrap class in a standard Red5 server

```
<listener>
 <listener-class>org.red5.server.war.WarLoaderServlet</listener-class>
</listener>
```

This listener is responsible for starting and stopping Red5 upon receipt of context initialized and context destroyed container events. The war loader is similar in function to the Spring ContextLoaderListener servlet but is specialized for Red5.

35.3.1. Spring contexts

There are two types of contexts used by Red5, "default" and "web"; there may be only one default context but any number of web contexts.

35.3.2. Default context

The default context is synonymous with the global application context and is responsible for providing objects and resources at the top or global level. Spring beans in this level are configured via the defaultContext.xml and beanRefContext.xml which are located in the ROOT classes directory (ex. C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes). The bean ref file defines the default.context bean which as an instance of org.springframework.context.support.ClassPathXmlApplicationContext. Two other configuration files red5-common.xml and red5-core.xml are used to construct the default context; these files are derived from the standalone configuration files of the same names, the primary difference is that the server embedding sections have been removed.

The default context is referenced in the web.xml via the parentContextKey parameter:

```
<context-param>
 <param-name>parentContextKey</param-name>
 <param-value>default.context</param-value>
</context-param>
```

This parameter is used by the ContextLoader to locate the parent context, which in turn allows the global resources to be located. The context loader is used by the WarLoaderServlet to initialize the web contexts.

The scope counterpart to the global context is the global scope and it is referenced in the web.xml via the globalScope parameter:

```
<context-param>
 <param-name>globalScope</param-name>
 <param-value>default</param-value>
</context-param>
```

35.3.3. Web context

Web context definitions are specified in Spring configuration files suffixed with -web.xml; If your application is named oflaDemo then its configuration file would be named oflaDemo-web.xml. The Spring web context files should not be confused with JEE context descriptors as they are only used for red5 web contexts and the later are used by Tomcat. Each web context must have a corresponding configuration file, the configuration files are specified using an ant-style parameter in the web.xml as shown below.

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>WEB-INF/classes/*-web.xml</param-value>
</context-param>
```

Context configuration files specify the resources that are used to notify the application about joining / leaving clients and provide the methods that a client can call. Additionally, the configuration files specify the scope hierarchy for these classes.

Every context configuration must contain a minimum of three entries - a context, scope, and handler. The only exception to this rule is the root web application since it does not have a handler application, in this case the global handler is used.

Context - Each context must have a unique name assigned since all the contexts exist within a single Spring application context. The root web context is named web.context, additional contexts suffix this base name with their web application name; for example oflaDemo would be named web.context.oflaDemo. A context is specified in the web context file as shown below.

```
<bean id="web.context" class="org.red5.server.Context">
 <property name="scopeResolver" ref="red5.scopeResolver" />
 <property name="clientRegistry" ref="global.clientRegistry" />
 <property name="serviceInvoker" ref="global.serviceInvoker" />
 <property name="mappingStrategy" ref="global.mappingStrategy" />
</bean>
```

Scope - Every application needs at least one scope that links the handler to the context and the server. The scopes can be used to build a tree where clients can connect to every node and share objects inside this scope (like shared objects or live streams). You can consider the scopes as rooms or instances.

The root scope has the name web.scope, additional scope names should follow the naming convention specified for contexts. A scope for oflaDemo would be named web.scope.oflaDemo so that it will not conflict with other contexts.

A scope bean has the following properties: 1.server - This references the server red5.server 2.parent - The parent for this scope is normally global.scope 3.context - Context for this scope, use the web.context for root and web.context.oflaDemo for oflaDemo 4.handler - Handler for this scope, which is similar to a main.asc in FMS. 5.contextPath - The path to use when connecting to this scope. 6.virtualHosts - A comma separated list of host names or IP addresses this scope listens on. In the war version we do not control the host names, this is accomplished by Tomcat. A value of “*” which means “all” is expected here.

The root scope definition looks like this:

```
<bean id="web.scope" class="org.red5.server.WebScope" init-method="register">
 <property name="server" ref="red5.server" />
 <property name="parent" ref="global.scope" />
 <property name="context" ref="web.context" />
 <property name="handler" ref="global.handler" />
 <property name="contextPath" value="/" />
 <property name="virtualHosts" value="*" />
</bean>
```

The contextPath is similar to the docBase in the JEE context file for each web application. Where the docBase is used to locate resources by HTTP, the contextPath is use to find resources via RTMP. Your applications may add additional elements after the configured path to dynamically create extra scopes. The dynamically created scopes all use the same handler but have their own properties, shared objects and live streams.

Handler - Every context needs a handler to provide the methods called by connecting clients. All handlers are required to implement org.red5.server.api.IScopeHandler, however you may implement additional interfaces for controlling access to shared objects or streams. A sample implementation is provided with Red5 that may be used as your base class: org.red5.server.adapter.ApplicationAdapter. Please refer to the javadoc for this class for additional details.

As an example the scope handler for the oflaDemo is shown:

```
<bean id="web.handler.oflaDemo" class="org.red5.demos.oflaDemo.Application"/>
```

The id attribute is referenced by the oflaDemo scope definition:

```
<bean id="web.scope.oflaDemo" class="org.red5.server.WebScope" init-method="register">
 <property name="server" ref="red5.server" />
 <property name="parent" ref="global.scope" />
 <property name="context" ref="web.context.oflaDemo" />
 <property name="handler" ref="web.handler.oflaDemo" />
 <property name="contextPath" value="/oflaDemo" />
 <property name="virtualHosts" value="*" />
</bean>
```

If you don't need any special server-side logic, you can use the default application handler provided by Red5:

```
<bean id="web.handler" class="org.red5.server.adapter.ApplicationAdapter" />
```

External applications An external application refers to a web application that accesses Red5 outside of the ROOT web application. Whether these applications exist within the same JVM instance or not, they may only access Red5 via RTMP or the AMF tunnel servlet. The tunnel servlet is configured in the web.xml for each application that requires AMF communication with Red5, an example is shown below:

```
    <context-param>
        <param-name>tunnel.acceptor.url</param-name>
        <param-value>http://localhost:8080/gateway</param-value>
    </context-param>

    <context-param>
        <param-name>tunnel.timeout</param-name>
 <param-value>30000</param-value>
</context-param>

<servlet>
 <servlet-name>gateway</servlet-name>
 <servlet-class>org.red5.server.net.servlet.AMFTunnelServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>gateway</servlet-name>
 <url-pattern>/gateway</url-pattern>
</servlet-mapping>
```

The tunnel servlet class must be on the classpath of the application under which it is executed. In addition to the tunnel servlet the org.red5.server.net.servlet.ServletUtils class is required along with the following library jars:

```
commons-codec-1.3.jar
commons-httpclient-3.0.1.jar
mina-core-2.0.0-M6.jar
```

```
slf4j-api-1.5.6.jar
jcl-over-slf4j-1.5.6.jar
jul-to-slf4j-1.5.6.jar
log4j-over-slf4j-1.5.6.jar
logback-core-0.9.14.jar
logback-classic-0.9.14.jar
red5-remoting-0.8.0.jar (contains ServletUtils class)
```

These jars should be placed in the WEB-INF/lib directory of your application. ex. C:\Tomcat-6.0.18\webapps\myapp\WEB-INF\lib

35.4. Creating and deploying your application

In the following section, two applications will be covered. The first will be a web application that communicates with Red5 via AMF or RTMP and has its own handler, referred to as “RemoteApp”. The second will consist an SWF that communicates with Red5 via RTMP, this application will be called “LocalApp”. Any IDE may be used to create these applications as long as it supports Java; the Eclipse IDE is suggested. SWF files outlined in the examples were created using AS3 in Flex.

35.4.1. Remote application

This example will provide you with the minimum amount of configuration needed for a remote Red5 application. The following resources will be created: * JEE web application * Client SWF * Red5 handler class * Spring web context

Steps 1.Create a web application named RemoteApp in your IDE. 2.Obtain a red5 jar, which may be downloaded from http://red5.googlecode.com/svn/repository/red5/red5-0.8.0.jar or built from source with the command “ant jar”. This library is needed if you extend the ApplicationAdapter for your scope handler. 3.Obtain the remoting jar, this may be accomplished by building yourself from the command line with “ant remotejar” or by downloading it from http://red5.googlecode.com/svn/repository/red5/red5-remoting-0.8.0.jar. This library provides the AMF tunnel servlet. 4.Place the library jars in your project library directory and add them to your build classpath. 5.Compile the Java and Flex source. 6.Create a directory named RemoteApp in the Tomcat webapps directory. ex. C:\Tomcat-6.0.18\webapps\RemoteApp 7.Copy the contents of the web directory to the RemoteApp directory. 8.From the bin directory copy the RemoteApp.swf to the webapps\RemoteApp directory. 9.Copy the lib directory and its contents to the WEB-INF, excluding the red5.jar file. 10.Copy the whole example directory and the RemoteApp-web.xml file from the bin directory to the classes directory under ROOT. ex. C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes 11.Restart tomcat 12.Open your browser and go to: http://localhost:8080/RemoteApp/RemoteApp.html 13.Click on the RTMP or HTTP connect buttons. For a successful test you should see a server response of “Hello World”.

35.4.2. Local application

A simple application that resides entirely within the ROOT web application. This example consists of a Spring web context, handler class, and a client SWF.

Steps 1.Create a web application named LocalApp in your IDE. 2.Obtain a red5.jar, which may be downloaded from http://red5.googlecode.com/svn/repository/red5/red5-0.8.0.jar

or built from source with the command “ant jar”. This library is needed if you extend the ApplicationAdapter for your scope handler. 3.Place the library jar in your project library directory and add it to your build classpath. 4.Compile the Java and Flex source. 5.Copy the LocalApp.html and LocalApp.swf from the bin directory to the ROOT directory. ex. C:\Tomcat-6.0.18\webapps\ROOT 6.Copy the whole example directory and the LocalApp-web.xml file from the bin directory to the classes directory under ROOT. ex. C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes 7.Restart tomcat 8.Open your browser and go to: http://localhost:8080/LocalApp.html 9.Click on the connect button. For a successful test you should see a server response of “Hello World”.

35.4.3. Example Source

The example application source is available in Subversion at http://red5.googlecode.com/svn/java/example/trunk

35.5. Additional web configuration

AMF gateway - This servlet provides communication with server applications using AMF.

```
<servlet>
 <servlet-name>gateway</servlet-name>
 <servlet-class>org.red5.server.net.servlet.AMFGatewayServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>gateway</servlet-name>
 <url-pattern>/gateway</url-pattern>
</servlet-mapping>
```

RTMPT - This servlet implements an RTMP tunnel via HTTP, this is normally used to bypass firewall issues.

```
<servlet>
 <servlet-name>rtmpt</servlet-name>
 <servlet-class>org.red5.server.net.rtmpt.RTMPTServlet</servlet-class>
 <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>rtmpt</servlet-name>
 <url-pattern>/fcs/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>rtmpt</servlet-name>
 <url-pattern>/open/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>rtmpt</servlet-name>
 <url-pattern>/idle/*</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
 <servlet-name>rtmpt</servlet-name>
 <url-pattern>/send/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>rtmpt</servlet-name>
 <url-pattern>/close/*</url-pattern>
</servlet-mapping>
```

Security - The following entries are used to prevent retrieval of sensitive information.

```
<security-constraint>
 <web-resource-collection>
  <web-resource-name>Forbidden</web-resource-name>
  <url-pattern>/WEB-INF/*</url-pattern>
 </web-resource-collection>
 <auth-constraint />
</security-constraint>

<security-constraint>
 <web-resource-collection>
  <web-resource-name>Forbidden</web-resource-name>
  <url-pattern>/persistence/*</url-pattern>
 </web-resource-collection>
 <auth-constraint />
</security-constraint>

<security-constraint>
 <web-resource-collection>
  <web-resource-name>Forbidden</web-resource-name>
  <url-pattern>/streams/*</url-pattern>
 </web-resource-collection>
 <auth-constraint />
</security-constraint>
```

35.6. Renaming the ROOT war to Red5

In certain circumstances you may need to rename the war so that it does not override the ROOT web application of your container. There are just a few steps to make this work:

1. Open the web.xml for red5

tomcat/webapps/red5/WEB-INF/web.xml

2. Change the webAppRootKey parameter to /red5

```
 <context-param>
  <param-name>webAppRootKey</param-name>
  <param-value>/red5</param-value>
 </context-param>
```

3. Open the root-web.xml file

tomcat/webapps/red5/WEB-INF/classes/root-web.xml

4. Change the contextPath and virtualHosts in the web.scope bean to look like this:

```
<property name="contextPath" value="/red5" />
<property name="virtualHosts" value="*" />
```

35.7. Troubleshooting

If you have problems with deployment or if your application does not start, follow these steps prior to posting a bug. Directory examples use a typical windows based path structure.

1. Stop the Tomcat server

2. Locate your Tomcat installation directory C:\Program Files\Apache\Tomcat

3. Delete the "work" directory C:\Program Files\Apache\Tomcat\work

4. Delete the "Catalina" directory from the "conf" directory C:\Program Files\Apache\Tomcat \conf\Catalina

5. Delete the expanded war directories, if they exist C:\Program Files\Apache\Tomcat \webapps\ROOT C:\Program Files\Apache\Tomcat\webapps\echo C:\Program Files \Apache\Tomcat\webapps\SOSample

6. Ensure your WAR files are in the webapps directory C:\Program Files\Apache\Tomcat \webapps\ROOT.war C:\Program Files\Apache\Tomcat\webapps\echo.war C:\Program Files\Apache\Tomcat\webapps\SOSample.war

7. Restart Tomcat

If you still experience problems, gather the following information and post an issue on Trac after you do a quick search to see if others have experienced the same problem.

1. Java version 2. Tomcat version 3. Operating system 4. Red5 version (0.8, Trunk, Revision 2283, etc...)

35.8. Definitions

AMF – A binary format based loosely on the Simple Object Access Protocol (SOAP). It is used primarily to exchange data between an Adobe Flash application and a database, using a Remote Procedure Call. Each AMF message contains a body which holds the error or response, which will be expressed as an ActionScript Object.

Ant – Software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, and is best suited to building Java projects.

AS3 – A scripting language based on ECMAScript, used primarily for the development of websites and software using the Adobe Flash Player platform.

Flex – Software development kit and an IDE for a group of technologies initially released in March of 2004 by Macromedia to support the development and deployment of cross platform, rich Internet applications based on their proprietary Macromedia Flash platform.

RTMP – Real Time Messaging Protocol (RTMP) is a proprietary protocol developed by Adobe Systems that is primarily used with Adobe Flash Media Server to stream audio, video, and data over the internet to the Adobe Flash Player client. RTMP can be used for Remote Procedure Calls. RTMP maintains a persistent connection with an endpoint and allows real-time communication. Other RPC services are made asynchronously with a single client/server request/response model, so real-time communication is not necessary.

RTMPT – RTMP using HTTP tunneling.

SWF – Proprietary vector graphics file format produced by the Flash software from Adobe. Intended to be small enough for publication on the web, SWF files can contain animations or applets of varying degrees of interactivity and function. SWF is also sometimes used for creating animated display graphics and menus for DVD movies, and television commercials.

Tomcat – A web container, or application server developed at the Apache Software Foundation (ASF). Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own internal HTTP server.

http://www.wireshark.org/

It appears when using Wireshark on OSX the libpcap requires access to a BPF device for capturing. The ownership / permissions require to be changed each time to allow access to the network interfaces in Wireshark. Run this command upon each time requiring to use Wireshark or add it to a command as a startup item.

```
sudo chown $USER /dev/bpf*
```

Part V. Examples

Red5 Example Applications

- Chapter 37, *File Upload Servlet Example*
- Chapter 14, *Security*
- Chapter 38, *Server Side Playlists*

http://red5.electroteque.org/dev/demos/uploadTest.zip

An example file upload servlet examples which uses spring MVC and a flash file uploader. Files are configured to go into the set streams directory.

The only requirement is to modify the setting to where uploads are stored in the red5-web.properties.

```
webapp.uploadPath=streams/
```

http://red5.electroteque.org/dev/demos/playlistTest.zip

or

http://red5.electroteque.org/dev/demos/playlistTest.war

To compile run

```
ant
```

It will produce a war in the dist directory, copy it to the webapps directory and the war deployer will deploy the webapp.

The playlist is setup like this so change the settings here

```
playlist.repeat=true
playlist.path=streams/
playlist.pattern=*.flv
playlist.streamName=webtv
playlist.runOnStart=true
```

```
playlist.runOnStart
```

Will tell the application to start the stream on application startup, which will make it act like a real web tv stream. Otherwise the stream will startup on client connection.

Appendix A. RTMP Specification

http://osflash.org/documentation/rtmp

- http://wiki.gnashdev.org/RTMP
- http://wiki.gnashdev.org/RTMP_Messages_Decoded
- http://osflash.org/_media/rtmp_spec.jpg
- http://www.acmewebworks.com/Downloads/openCS/TheAMF.pdf

Appendix B. RTMPT Specification

This document describes the RTMPT tunneling protocol as implemented by the Red5 Open Source Flash Server. Please note that this document is _not_ an official specification by Macromedia but hopefully helps other people to write software that makes use of RTMPT.

RTMPT basically is a HTTP wrapper around the RTMP protocol that is sent using POST requests from the client to the server. Because of the non-persistent nature of HTTP connections, RTMPT requires the clients to poll for updates periodically in order to get notified about events that are generated by the server or other clients.

During the lifetime of a RTMPT session, four possible request types can be sent to the server which will be described below.

B.1. URLs

The URL to be opened has the following form::

http://server/<comand>/[<client>/]<index>

`<command>`

denotes the RTMPT request type (see below)

`<client>`

specifies the id of the client that performs the requests (only sent for established sessions)

`<index>`

is a consecutive number that seems to be used to detect missing packages

B.2. Request / Response

All HTTP requests share some common properties:

- They use HTTP 1.1 POST.
- The content type is `application/x-fcs`.
- The connection should be kept alive by the client and server to reduce

  network overhead.

The HTTP responses also share some properties:

- The content type is `application/x-fcs`.
- For all established sessions the first byte of the response data controls

  the polling interval of the client where higher values mean less polling requests.

B.3. Polling interval

The server always starts with a value of 0x01 after data was returned and increases it after 10 emtpy replies. The maximum delay is 0x21 which causes a delay of approximately 0.5 seconds between two requests.

Red5 currently increases the delay in the following steps: 0x01, 0x03, 0x05, 0x09, 0x11, 0x21.

B.4. Initial connect (command "open")

This is the first request that is sent to the server in order to register a client on the server and start a new session. The server replies with a unique id (usually a number) that is used by the client for all future requests.

Note: the reply doesn't contain a value for the polling interval! A successful connect resets the consecutive index that is used in the URLs.

B.5. Client updates (command "send")

The data a client would send to the server using RTMP is simply prefixed with a HTTP header and otherwise sent unmodified.

The server responds with a HTTP response containing one byte controlling the polling interval and the RTMP data if available.

B.6. Polling requests (command "idle")

If the client doesn't have more data to send to the server, he has to poll for updates to receive streaming data or events like shared objects.

B.7. Disconnect of a session (command "close")

If a client wants to terminate his connection, he sends the "close" command which is replied with a 0x00 by the server.

Appendix C. AMF Specification

C.1. AMF0 Specification

http://download.macromedia.com/pub/labs/amf/amf0_spec_121207.pdf

C.2. AMF3 Specification

http://download.macromedia.com/pub/labs/amf/amf3_spec_121207.pdf http://osflash.org/documentation/amf3

Appendix D. Data Type Mappings

Table D.1. Flash to Java Data Mappings

Number	any of the Java numeric types
int/uint *	any of the non-floating point Java numeric types
String	String
Boolean	Boolean/boolean
Array	List
Object	org.red5.io.utils.ObjectMap [http://dl.fancycode.com/red5/api/org/red5/io/utils/ObjectMap.html]
Date	java.util.Date
XML	org.w3c.dom.Document
ByteArray *	org.red5.io.amf3.ByteArray [http://dl.fancycode.com/red5/api/org/red5/io/amf3/ByteArray.html]
IExternalizable *	org.red5.io.amf3.IExternalizable [http://dl.fancycode.com/red5/api/org/red5/io/amf3/IExternalizable.html]
ArrayCollection *	org.red5.compatibility.flex.messaging.io.ArrayCollection [http://dl.fancycode.com/red5/api/org/red5/compatibility/flex/messaging/io/ArrayCollection.html]
ObjectProxy *	org.red5.compatibility.flex.messaging.io.ObjectProxy [http://dl.fancycode.com/red5/api/org/red5/compatibility/flex/messaging/io/ObjectProxy.html]
null	null
custom class **	custom class

Note

Please note that Red5 performs automatic parameter conversion, e.g. if you pass a number to a method that takes a String as parameter, it is automatically converted.

1. * - Only available in Flash Player 9 or newer (AMF3)

2. ** - You can map the class to serialize to in Red5 by adding

Appendix E. FLV

http://osflash.org/flv

http://www.adobe.com/devnet/flv/pdf/video_file_format_spec_v9.pdf

Appendix F. H264

This Section contains some of the answers from the mailing list about h.264. Please read this first before sending another question to the mailing list.

Paul Gregoire Posted initial Support in his Blog 6. October 2008 18:34: Steven and I got h264 working ! http://gregoire.org/2008/10/06/red5-h264/

Questions:

F.1. Does Red5 plan to support H.264/ACC streams in the next release, maybe 0.7.1?

You can check out Paul's mp4/f4v branch here: http://red5.googlecode.com/svn/java/server/branches/paulg_mp4/

and according to the plan it should be part of the 0.7.1 release - afaik.

And I'll let Paul speak for himself :-) http://osflash.org/pipermail/red5_osflash.org/2008-January/018483.html

Paul:I feel like I'm done it myself a million times :-) I'm currently active in the mp4 / h.264 branch and I would love to have it be part of the 0.7.1 release. The problem for me is that I'm not sure when I will be done and I would like to 0.7.1 go live very soon; it is more likely that 0.7.2 will include the release of h.264.

In actuality it is now scheduled for the 0.9.0 release but it may make it into 0.8.0. The 0.7.1 version will not be released - Paul

F.2. why are .mp4 files also listed in oflaDemo webapp grid as possible streams?

The "correct" extension in Adobe's case is F4V for Mpeg4 / h.264 encoded content. If you want a video that works, use the sample that comes with FMS3: AdobeBand_1500K_H264.mp4 Like I stated in my email/post, I have clean-up to do in addition to making the video config packet dynamic (the one that contains the decoder settings).

If you specify the flv extension that is what you will get, it will not include h264 content. FLV is expected to be any of the following video codecs: vp6, screenvideo, or h263. The audio will also not support AAC / MP4A for the flv extension. The f4v, mp4, and mov extensions will be provided by the MP4Reader class. The f4v file may contain avc1, vp6e, h264, mp4 for video and aac, mp4a, mp3, pcm for audio (there may be more but thats what I recall). Along side the MP4Reader is an audio only reader for AAC/MP4A files in the same style as the MP3Reader; it will be released at the same time and is for audio only files.

F.3. howto convert to h.264 using ffmpeg?

Here is a ffmpeg encoding line to provide mp4 with h264 codec readable with flash player 9 , FLVTOOL useless.

```
options="-vcodec libx264 -b 512k -bf 3 -subq 6 -cmp 256 -refs 5 -qmin 10 \
 -qmax 51 -qdiff 4 -coder 1 -loop 1 -me hex -me_range 16 -trellis 1 \
 -flags +mv4 -flags2 +bpyramid+wpred+mixed_refs+brdo+8x8dct \
 -partitions parti4x4+parti8x8+partp4x4+partp8x8+partb8x8 -g 250 \
 -keyint_min 25 -sc_threshold 40 -i_qfactor 0.71"

 ffmpeg -y -i "$X" -an -pass 1 -threads 2 $options "$tmpfile"

 ffmpeg -y -i "$X" -acodec libfaac -ar 44100 -ab 96k -pass 2 \
 -threads 2 $options "$tmpfile"

 qt-faststart "$tmpfile" "$outfile"
```

F.4. Does anyone have a link to an explaination of h264 licensing?

http://www.mpegla.com/avc/avc-licensees.cfm

http://www.flashmediaserver-blog.de/2008/05/21/54/ (skip the german text of the top) the list (in short) from their:

H264 /commercial if a Enduser chooses to pay title-by-title o no fee for videos shorter then 12 minutes o else the lower of: 0,02 US$ per Titel or 2% of the Price that the Enduser pays if the Enduser pays within Subscription contract: o < 100.000 Subscribers per year --> no Fee o 100.000 - 250.000 Subscribers per Y --> 25.000 US$/Yr o 250.000 - 500.000 Subscribers per Y --> 50.000 US$/Y o etc.

this might be interesting for germans (we as only folk on the world have a fee for free televeision called GEZ):

> dann gibt es noch ein weiteres Lizenzmodell, dass aber nur g?ltig ist, wenn Geb?hren des Endusers an anderer Stelle bereits bezahlt werden, wie z.B. bei den GEZ-Geb?hren.(ist aber nur f?r ?ffentlich rechtliche interessant):-) (there a differenet licensing model aswell but only available if the Enduser pays there Fee?s some where else like in german GEZ)

for me the stream.seek functionality doesent work for .mp4 files did someone tested yet? Yes, seek is one of the things that doesn't work yet.

Will I be able to multiplex H.264 video and AAC audio into the current FLV container format?

A: Adobe encourages customers to use the new MPEG-4-based file format. The new file format is designed to work with the features of these codecs.

from: http://labs.adobe.com/wiki/index.php/Flash_Player:9:Update:H.264#Q:_Will_I_be_able_to_multiplex_H.264_video_and_AAC_audio_into_th

F.5. Someone already created a demo?

http://s1blue2.waltertak.com:15080/demos/

This demo is a simple AS3 Flash client playing a selection of FLV and H.264 content. The FLV content is to show the huge difference in quality.

Some H.264 content isn't played correct ; the playhead runs too fast resulting in fast-forwarding.

The large "720" and "1080" content requires upto 2 megabytes per second of bandwidth to be played smoothly.

Seeking through H.264 doesn't work yet (as of 2008/10/08) .

F.6. Some more Info about Seek and possible solutions

http://h264.code-shop.com/trac/wiki/FlashPlayer

Seek isn't working yet with H.264 content (as of 2008/10/08) .

F.7. Are audio files supported?

Yes, the following file types may be played:

- AAC
- M4A
- F4A

They are delivered in the same manner as an MP3 file, but with a different reader.

F.8. How do I request an h264 file?

These request methods are supported:

- mp4:myfile
- mp4:myfile.f4v
- mp4:myfile.mp4
- mp4:myfile.mov
- mp4:myfile.3gp
- mp4:myfile.3g2
- myfile.f4v
- myfile.mp4
- myfile.mov

- myfile.3gp
- myfile.3g2

F.9. Does red5 support h264 live streaming?

h.264 currently concerns only VOD. Paul: Right now, RTP/RTSP is not supported. When the MP4/h.264 features are complete, you will get RTP/RTSP for Free!

F.10. Links

- http://today.java.net/pub/a/today/2006/08/22/experiments-in-streaming-java-me.html
- http://rtspproxy.berlios.de/

Appendix G. Speex Codec

Speex [http://www.speex.org/] is a new voice codec supported in Flash Player 10 and above. With the new release of flash player 10 this document explains a little about the new codec with this release.

G.1. Setting Up Flex SDk / Flex Builder

Flex SDK / Flex Builder needs to be setup to target flash player 10 playerglobal.swc which has the new features enabled.

Taken from the Adobe Open Source area http://opensource.adobe.com/wiki/display/flexsdk/Targeting+Flash+Player+10+Beta

G.1.1. Get Flex 3 SDK

- Download Flex SDK 3.0.3 or above. In Flex 3 extract into the sdks directory.
- Make sure that you have a Flash 10 playerglobal.swc at FLEX_SDK/frameworks/libs/player/10/playerglobal.swc.

G.1.2. Config Flex Config To Target Flash Player 10

- Modify FLEX_SDK/frameworks/flex-config.xml. Edit <target-player>, replacing 9.0.115 with 10.0.0:

  {{{ <target-player>10.0.0</target-player> }}}

- In <external-library-path>, edit the path-element for playerglobal.swc, replacing 9 with 10:

  {{{ <external-library-path> <path-element>libs/player/10/playerglobal.swc</path-element> </external-library-path> }}}

- Do the same with <library-path:

  {{{ <library-path> <path-element>libs</path-element> <path-element>libs/player/10</path-element> <path-element>locale/{locale}</path-element> </library-path> }}}

- In flex builder, right click the project and choose properties, choose Flex Compiler, choose the "Configure Flex SDKs" link on the right, choose add then choose the location of the newly downloaded SDK and give it a name.
- Back in the Flex Compiler config choose "Use a specific SDK and select in the list the Flex SDK configured previously.

G.2. Code Example

Setup the Microphone class to use the new Speex Codec.

```
_microphone = Microphone.getMicrophone();
if (_microphone) {
    //_microphone.setLoopBack(true);
    _microphone.codec = SoundCodec.SPEEX;
    _microphone.encodeQuality = 10;
    _microphone.rate = 44;
    _microphone.framesPerPacket = 2;
    _microphone.gain = 50;
    _microphone.setUseEchoSuppression(true);
    ns.attachAudio(_microphone);
} else {
    throw new Error("Audio Not Connected");
}
```

G.3. FFMpeg and Speex

According to Art Clarke subversion revision r15028 [http://svn.mplayerhq.hu/ffmpeg/trunk/?pathrev=15028] of the FFMpeg source code has the capabilities of decoding the Speex codec from FLV files.

G.4. Links

* http://www.speex.org/

* http://help.adobe.com/en_US/AS3LCR/Flash_10.0/flash/media/SoundCodec.html

* http://help.adobe.com/en_US/AS3LCR/Flash_10.0/flash/media/Microphone.html