



Open Source Flash Server

Red5 - Reference Documentation

Version 0.8

Copyright © Red5 Open Source Flash Server

Steven Gong, Paul Gregoire, Daniel Rossi

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

Preface	viii
1. Introduction	1
2. What's new in Red5 0.8 RC1	2
2.1. 0.8 Public Beta Release	2
I. Getting Started	3
3. Frequently Asked Questions	4
3.1. Questions	4
3.1.1. GENERAL	4
3.1.2. DOCUMENTATION	4
3.1.3. CONFIGURATION	4
3.1.4. STREAMING	4
3.1.5. CODECS	5
3.1.6. DATABASE	5
3.1.7. SCRIPTING	5
3.1.8. SHARED OBJECTS	5
3.1.9. LEGAL STUFF	5
3.1.10. Red5 WAR version	5
3.1.11. MISC	6
3.1.12. TROUBLESHOOTING	6
3.2. Answers	6
3.2.1. GENERAL	6
3.2.2. DOCUMENTATION	8
3.2.3. CONFIGURATION	8
3.2.4. STREAMING	8
3.2.5. CODECS	9
3.2.6. DATABASE	10
3.2.7. SCRIPTING	10
3.2.8. SHARED OBJECTS	10
3.2.9. LEGAL STUFF	11
3.2.10. Red5 WAR version	11
3.2.11. MISC	11
3.2.12. TROUBLESHOOTING	14
4. Configuration Files	15
4.1. Directory "conf"	15
4.1.1. jetty.xml	15
4.1.2. keystore	15
4.1.3. log4j.properties	15
4.1.4. realm.properties (Jetty)	15
4.1.5. tomcat-users.xml (Tomcat)	15
4.1.6. red5.globals	16
4.1.7. red5.properties	16
4.1.8. red5.xml	16
4.1.9. red5-common.xml	16
4.1.10. red5-core.xml	17
4.1.11. red5-rtmpt.xml	18
4.1.12. web.xml (Tomcat)	18
4.1.13. web-default.xml (Jetty)	18
4.2. Webapp config directory	18
4.2.1. red5-web.xml	18

5. Migration Guide	20
5.1. Application callbacks	20
5.1.1. Interface IScopeHandler	20
5.1.2. Class ApplicationAdapter	20
5.1.3. Accepting / rejecting clients	21
5.2. Current connection and client	21
5.3. Additional handlers	22
5.3.1. Handlers in configuration files	22
5.3.2. Handlers from application code	23
5.4. Calls to client methods	23
5.5. SharedObjects	24
5.5.1. Serverside change listeners	25
5.5.2. Changing from application code	25
5.6. Persistence	27
5.7. Periodic events	28
5.8. Remoting	29
5.8.1. Remoting server	29
5.8.2. Remoting client	30
5.9. Streams	30
6. Red5 Libraries	31
6.1. Spring scripting support	31
6.2. Groovy	31
6.3. Beanshell	31
6.4. Ruby	31
6.5. Jython / Python	31
6.6. Java 5 Libraries	31
6.7. Script related JSR's	31
6.8. Javascript / Rhino	32
7. Building Red5	33
7.1. Build Environment Setup	33
7.1.1. Ant	33
7.1.2. Java	33
7.1.3. Red5	33
7.2. Building	34
7.2.1. Getting Red5 Source	34
7.2.2. Getting Red5 Demo Applications Source	34
7.2.3. Getting Red5 Flash Demo Source	34
7.2.4. Running the ant build	34
7.2.5. Current Ant Targets	34
7.2.6. Ant and Ivy	36
7.3. How to build with eclipse	36
7.3.1. Recommended Eclipse Plugins	36
7.3.2. Importing the Red5 Project	37
7.3.3. Updating the Red5 source	37
7.3.4. Debugging Red5 in Eclipse	38
7.3.5. Ant, Ivy and Eclipse	39
8. Releasing Red5	40
9. System Requirements For Red5	41
II. Red5 Core Technologies	42

10. Create new applications in Red5	43
10.1. The application directory	43
10.2. Configuration	43
10.2.1. webAppRootKey	43
10.3. Handler configuration	43
10.3.1. Context	43
10.3.2. Scopes	44
10.4. Handlers	45
10.5. Logging	45
11. Deploying Red5 To Tomcat	46
11.1. Preface	46
11.2. Deployment	46
11.3. Context descriptors	46
11.4. Red5 Configuration	46
11.4.1. Spring contexts	47
11.4.2. Default context	47
11.4.3. Web context	48
11.4.4. External applications	50
11.5. Creating and deploying your application	51
11.5.1. Remote application	51
11.5.2. Local application	52
11.5.3. Example Source	52
11.6. Additional web configuration	52
11.7. Troubleshooting	54
11.8. Definitions	55
11.9. Bibliography	56
12. Customize Stream Paths	57
12.1. Filename generator service	57
12.2. Custom generator	57
12.3. Activate custom generator	58
12.4. Change paths through configuration	58
13. Security	60
13.1. Stream Security	60
13.1.1. Stream playback security	60
13.1.2. Stream publishing security	61
14. Scripting Implementations	62
14.1. I. Select a scripting implementation	62
14.2. II. Configuring Spring	62
14.3. III. Creating an application script	64
14.3.1. 1. Application adapter	64
14.3.2. 2. Application services	66
14.4. IV. Creating your own interpreter	69
14.5. V. Links with scripting information	70
15. Clustering	71
15.1. Limitations	71
15.2. Server Configuration	71
15.2.1. Configuration Files	71
15.3. Configure Edge Server	71
15.3.1. Edge on a different Server from Origin	71

15.3.2. Edge on the same Server as Origin	72
15.4. Configure Origin Server	72
15.5. Use Your Appliation	72
16. Management	73
16.1. JMX Classes	73
16.2. Spring configuration	73
16.3. RMI Authentication	74
16.4. JMX / RMI / SSL	75
16.5. jConsole / JMX Client	75
16.5.1. Local Management	75
16.5.2. Remote Management	75
16.5.3. SSL Remote Management	75
16.6. Links	76
17. List of Custom bean definitions	77
17.1. how to use the custom settings	77
17.2. Bean Definitions	77
18. Red5 Demo Applications	78
18.1. Getting Red5 Demo Applications Server-Side and Client-Side Source	78
18.2. List Of Available Demo Applications (Server Side)	78
18.3. List Of Available Demo Applications (Client Side)	78
18.4. Environment Build Setup	78
18.5. Building The Demo Application	79
18.6. Updating The Applications Registry	79
18.7. Bandwidth Check Application	80
18.7.1. Source Code	80
18.7.2. Bandwidth Check Service Methods	80
18.7.3. ServerClientDetection	80
18.7.4. ClientServerDetection	83
19. Testing Red5	84
19.1. Overview	84
19.2. How to Start Testing Without Reading This Chapter	84
19.3. Who Should Read This Chapter In Depth?	85
19.4. Red5 Testing Strategy	85
19.5. Red5 Testing Props	86
19.6. Unit Testing	86
19.6.1. Purpose	86
19.6.2. Technology	86
19.6.3. Running Tests	86
19.6.4. Creating New Tests	86
19.6.5. Running unit tests from eclipse	87
19.6.6. Guidelines for New Unit Tests	87
19.6.7. Submitting New Unit Tests	87
19.6.8. Suggesting New Unit Tests	88
19.7. Integration Testing	88
19.7.1. Purpose	88
19.8. System Testing	88
19.8.1. Purpose	88
19.9. Technology	89

19.10. Running Tests	89
19.11. Creating New Tests	90
19.12. A Sample System Test	91
19.13. Guidelines for New System Tests	91
19.14. Submitting New System Tests	92
19.15. Suggesting New System Tests	92
19.16. Continuous Integration	92
19.16.1. Overview	92
19.16.2. Technology	93
19.16.3. How To Run The Continuous Build	93
19.16.4. How to Submit New Jobs for Continuous Building	93
19.17. How you can help with Continuous Building	93
19.17.1. How to Set up a Continuous Build Server	93
A. RTMPT Specification	96
A.1. Overview	96
A.2. URLs	96
A.3. Request / Response	96
A.4. Polling interval	97
A.5. Initial connect (command "open")	97
A.6. Client updates (command "send")	97
A.7. Polling requests (command "idle")	97
A.8. Disconnect of a session (command "close")	97
B. Changelog	98
B.1. Red5 0.7.1 (unreleased)	98
B.2. Red5 0.7.0 (2008-02-23)	98
B.3. Red5 0.6.3 (2007-09-17)	99
B.4. Red5 0.6.2 (2007-06-17)	100
B.5. Red5 0.6.1 (2007-05-23)	101
B.6. Red5 0.6 (2007-04-23)	102
B.7. Red5 0.6rc3 (2007-04-11)	103
B.8. Red5 0.6rc2 (2007-02-12)	104
B.9. Red5 0.6rc1 (2006-10-30)	106
B.10. Red5 0.5 (2006-07-25)	107
B.11. Red5 0.5rc1 (2006-07-11)	107
B.12. Red5 0.4.1 (2006-05-01)	108
B.13. Red5 0.4 (2006-04-20)	108
B.14. Red5 0.3 (2006-02-21)	108
B.15. Red5 0.2 (2005-10-21)	109

Preface

preface

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)

The project is currently at 0.8.0 RC1. To find about our timescales and planned features checkout the roadmap and read our FAQ. If you want to know more about RED5, keep browsing this manual.

Lastly the documentation is under development at the moment. If you have unanswered questions after using it, please check out our mailing list [http://osflash.org/mailman/listinfo/red5_osflash.org]

2.1. 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 3, *Frequently Asked Questions*
- Chapter 4, *Configuration Files*
- Chapter 5, *Migration Guide*
- Chapter 6, *Red5 Libraries*
- Chapter 7, *Building Red5*
- Chapter 8, *Releasing Red5*
- Chapter 9, *System Requirements For Red5*

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 JIRA.
- Ideas for new features can be talked about in the discussion space

3.1. Questions

3.1.1. GENERAL

- What is Red5? [WHATISRED5]
- What does Red5 stand for? [WhatdoesRed5standfor]
- Is there a migration guide from FMS to Red5?
[IsthereamigrationguidefromFMStoRed5]
- applications? How do I create new applications? [Howdolcreatenew]
- What are configuration files? [Whatareconfigurationfiles]
- Is there a mailing list? [Isthereamailinglist]
- What is the mailing list etiquette? (TODO)
- What Ports does Red5 use? [WhatPortsdoesRed5use]
- How can I help? I'm interested in helping the project. How can I help?
[Iminterestedinhelpingtheproject.]
- Who is on the Red5 Team? [WhoisontheRed5Team]
- Are there any benchmarks? (TODO)

3.1.2. DOCUMENTATION

- Where is the official documentation? [Whereistheofficialdocumentation]
- Can I get the documentation in PDF format?
[CanlgetthedocumentationinPDFformat]
- Where can I find the latest javadocs? [WherecanIfindthelatestjavadocs]

3.1.3. CONFIGURATION

- How to disable Socket policy checking for 443 (rtmps and https)?
[HowtodisableSocketpolicycheckingfor443rtmpsandhttps]

3.1.4. STREAMING

- How do I stream to/from custom directories?
[Howdolstreamtofromcustomdirectories]

- How to detect the end of recording? [Howtodetecttheendofrecording]
- How can I record RTMP streams from Red5? [HowcanIrecordRTMPstreamsfromRed5]
- Does Red5 support multicast streaming? [DoesRed5supportmulticaststreaming]
- Can Red5 stream using UDP? [CanRed5streamusingUDP]

3.1.5. CODECS

- What_Codecs_does_Red5_Support? [WhatCodecsdoesRed5Support]
- _is_RTMFP_and_when_will_it_be_available_in_Red5? What is RTMFP and when will it be available in Red5? [What]

3.1.6. DATABASE

- What databases are supported? [Whatdatabasesaresupported]
- Can I use Hibernate with Red5? [CanIuseHibernatewithRed5]

3.1.7. SCRIPTING

- What scripting languages are available? [Whatscriptinglanguagesareavailable]
- Does Red5 support Actionscript 1? [DoesRed5supportActionscript1]
- Does Red5 support Actionscript 3? [DoesRed5supportActionscript3]

3.1.8. SHARED OBJECTS

- How do you setup a Remote SharedObject? [HowdoyousetupaRemoteSharedObject]
- How can I set a Remote SharedObject from the server [HowdoyousetupaRemoteSharedObject] (TODO)
- How can I make a Remote SharedObject persistant on the server? [HowcanImakeaRemoteSharedObjectpersistantontheserver]
- What are remote SharedObject slots?

3.1.9. LEGAL STUFF

- Licence Information [LicenceInformation]
- Is Red5 Legal? [IsRed5Legal]
- Codec Licenses [CodecLicenses] (TODO)
- Third Party Licenses [ThirdPartyLicenses] (TODO)

3.1.10. Red5 WAR version

- Is there any documentation on the Red5 war version? [IsthereanydocumentationontheRed5warversion]

3.1.11. MISC

- Is there an IRC channel? [IsthereanIRCchannel]
- Are there forums? [Arethereforums]
- Are there any frameworks that I can start with? [ArethereanyframeworksthatIcanstartwith]
- What is Paperworld3D? [WhatisPaperworld3D]
- What is Jedai? [WhatisJedai]
- there free tools Are there free tools [Are] (TODO)
- Are there development tools? [Aretheredevelopmenttools]
- there video tutorials Are there video tutorials [Are] (TODO)
- Are there any examples on the web? [Arethereanyexamplesontheweb]
- Is there professional support? [Isthereanyprofessionalsupport]
- Are there hosting solutions? [Aretherehostingsolutions]
- What Red5 groups can I join? [WhatRed5groupscanIjoin]

3.1.12. TROUBLESHOOTING

- Why am I receiving "closing due to long handshake? [Whyamlreceivingclosingduetolonghandshake]

3.2. Answers

3.2.1. GENERAL

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

---- What does Red5 stand for?

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

Is there a migration guide from FMS to Red5?

Yes: Migration Guide [Documentation/UsersReferenceManual/GettingStarted/03-Migration-Guide]

How do I create new applications?

Creating New Applications [Documentation/UsersReferenceManual/Red5CoreTechnologies/01-Creating-New-Applications]

What are configuration files?

see: Configuration Files In Red5 [Documentation/UsersReferenceManual/GettingStarted/02-Configuration-Files]

Is there a mailing list?

- Announcements mailinglist [http://osflash.org/mailman/listinfo/red5-announce_osflash.org]
- Users mailinglist [http://osflash.org/mailman/listinfo/red5_osflash.org]
- Developers mailinglist [http://osflash.org/mailman/listinfo/red5devs_osflash.org]
- Tickets mailinglist [<http://groups.google.com/group/red5tickets>]
- SVN Commits mailinglist [http://osflash.org/mailman/listinfo/red5commits_osflash.org]

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

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

You can create a new JIRA ticket for any contributions you want to make, attach the files there or link it. make sure you signup on the mailinglist as well..

Who is on the Red5 Team?

The Red5 Project (red5 AT osflash.org)

3.2.1.1. Project Managers

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

3.2.1.2. 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)

3.2.1.3. 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)

3.2.2. DOCUMENTATION

Where is the official documentation?

Users Reference Manual [Documentation/UsersReferenceManual]

'Can I get the documentation in PDF format?

TODO

Where can I find the latest javadocs?

<http://red5.newviewnetworks.com/hudson/docs/> <http://api.red5.nl>

3.2.3. CONFIGURATION

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.

3.2.4. STREAMING

How do I stream to/from custom directories?

Customize Stream Paths [UsersReferenceManual/Red5CoreTechnologies/Chapter3]

How to detect the end of recording ?

see: <http://red5.newviewnetworks.com/hudson/docs/org/red5/server/api/stream/IStreamAwareScopeHandler.html>

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>

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 reliable for other platforms either such as Windows Media Player. These solutions usually fall back to unicasting when clients cannot receive multicasted media. In regards to Unicasting, Red5 already has this functionality. In addition, we have an edge-origin solution sometimes referred to as stream-repeating.

Can Red5 stream using UDP?

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

3.2.5. CODECS

What Codecs does Red5 Support?

Video codecs: ScreenVideo On2 VP6 Sorenson H.263 H264

Audio codecs: ADPCM NellyMoser MP3 Speex AAC

[span(**What is RTMFP and when will it be available in Red5?**, style=color: #006699 [nullticket/006699];)]

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.

3.2.6. DATABASE

What databases are supported?

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

Can I use Hibernate with Red5? Yes.

Red5 And Hibernate Example [<http://trac.red5.org/wiki/Documentation/Tutorials/Red5AndHibernate>]

3.2.7. SCRIPTING

What scripting languages are available?

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

Does Red5 support Actionscript 1?

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

Does Red5 support Actionscript 3?

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

3.2.8. SHARED OBJECTS

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

How can I make a Remote SharedObject persistant on the server?

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

3.2.9. LEGAL STUFF

Licence Information

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

For an easier explanation, please see: <http://jira.red5.org/confluence/display/docs/Red5+License+%28LGPL%29>

Is Red5 Legal?

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

3.2.10. Red5 WAR version

Is there any documentation on the Red5 war version?

read: Deploying To Tomcat [Documentation/UsersReferenceManual/Red5CoreTechnologies/02-Deploying-To-Tomcat]

3.2.11. MISC

Is there an IRC channel?

Yes: #red5 on irc.freenode.net

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

Are there any examples on the web?

Below is a list of applications that use Red5. Feel free to add your own !

- [<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> videoconf module by Sebastian Wagner.
- <http://spread.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, online 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 using 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 recording 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

Is there any professional support?

Companies Listed:

- Infrared5 (www.infrared5.com)
- Red5Server (<http://www.red5server.com>)

Are there hosting solutions?

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

Are there forums?

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

What is Jedai?

see: <http://jedai.googlecode.com>

Are there any frameworks that I can start with?

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

Are there development tools?

see: <http://www.red5.org/projects/red5plugin/>

What is Paperworld3D?

see: <http://www.paperworld3d.org>

What Red5 groups can I join?

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

3.2.12. TROUBLESHOOTING

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.

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

4.1. Directory "conf"

4.1.1. 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.

4.1.2. keystore

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

4.1.3. 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/>.

4.1.4. 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

4.1.5. 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>

4.1.6. 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".

4.1.7. red5.properties

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

4.1.8. 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="shutdown" />
... cut for brevity ...
</bean>
```

4.1.9. 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></property>
        <!-- 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></property>
        <!-- A predicate for filtering Collections of Items based on their expiration time -->
        <!-- property name="policy"><value>com.whirlycott.cache.policy.ExpirationTimePredicate</value></property>
        <!-- property name="backend"><value>com.whirlycott.cache.impl.ConcurrentHashMapImpl</value></property>
        <property name="backend"><value>com.whirlycott.cache.impl.FastHashMapImpl</value></property>
    </bean>
```

4.1.10. 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="bT
```

- Tomcat

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

4.1.11. 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.

4.1.12. 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>

4.1.13. 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.

4.2. Webapp config directory

4.2.1. 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.

5.1. 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.

5.1.1. 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.

5.1.2. 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:

FCS / FMS	Red5
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 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.

5.1.2.1. 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 connect to partially the same scopes, only a few join methods might be called, e.g. `rhttp://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.

5.1.3. 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/adaptor/ApplicationAdapter.html> class. Any parameter passed to `rejectClient` is available as the `application` property of the status object that is returned to the caller.

5.2. 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 possibility 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();
    // ...
}
```

5.3. 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.

5.3.1. 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.

5.3.2. 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.

5.4. 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.

5.5. 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>.

5.5.1. 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 <key> of the shared object <so>
        // was changed to <value>.
    }
    public void onSharedObjectDelete(ISharedObject so, String key) {
        // The attribute <key> of the shared object <so> was deleted.
    }
    public void onSharedObjectSend(ISharedObject so,
                                   String method, List params) {
        // The handler <method> of the shared object <so> was called
        // with the parameters <params>.
    }
    // 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())

```

5.5.2. 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" />
```

5.6. 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.

5.7. 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 invocation 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);
}
```

5.8. 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.

5.8.1. 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
```

5.8.2. 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);
```

5.9. Streams

TODO: How can streams be accessed from an application?

6.1. 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

6.2. Groovy

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

6.3. Beanshell

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

6.4. Ruby

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

6.5. Jython / Python

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

6.6. Java 5 Libraries

The following are need for Java 5 only:

6.7. Script related JSR's

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

6.8. Javascript / Rhino

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

7.1. Build Environment Setup

7.1.1. 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`

7.1.2. 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\jdk\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
```

7.1.3. 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 here [<http://www.netbeans.org>] Download Eclipse here [<http://www.eclipse.org>]

7.2. Building

7.2.1. 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>.

7.2.2. 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.

7.2.3. 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.

7.2.4. 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
```

7.2.5. 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-htmlesingle` - 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.

7.2.6. 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
```

More information here setup with Eclipse [docs:Ivy]

7.3. 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>

7.3.1. 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 setup with Eclipse [docs:Ivy]
- Spring IDE - <http://springide.org/project/wiki>
- Subclipse SVN Plugin - <http://subclipse.tigris.org/>

7.3.2. 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.

- Import the checked out working copy.

1. Start Eclipse
2. From the File menu select "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

1. Make sure red5 is selected in the projects area and hit "Finish"
2. Eclipse should automatically build the project, you can force a build from the "project"
 menu and selecting "build project"

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

1. Start Eclipse
2. From the File menu select "import"
3. In the Import dialog box select SVN and then select the item "Checkout Projects from SVN" and hit next
4. 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.

1. Click Finish.
2. Eclipse should automatically build the project, you can force a build from the "project"
 menu and selecting "build project"

7.3.3. Updating the Red5 source

1. In eclipse right click the Red5 source project

2. Locate to "Team" and then "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, select Project -> Clean.

7.3.4. Debugging Red5 in Eclipse

1. Click the arrow next to the Debug icon menu and then click "Open Debug Dialog".
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.

1. Select the Arguments tab
2. Place this into Program Arguments

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

1. Place this into VM Arguments

```
-cp ./conf
```

1. 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
```

1. Click Apply and Close.
2. Right click the project and choose Build Path -> Configure Build Path.
3. In the Source Tab, choose Add Folder and select the src/conf directory.
4. Make sure "Allow output folders for source folders" is selected.
5. Under red5_server/src/conf, select Output Folder and choose edit.
6. Select Specific output, select the root directory and choose "create new folder".
7. Select conf, the output folder for the Red5 configs will now be placed into red5_server/conf.
8. With the imported red5 project selected click the debug icon and it will launch the server.

1. 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.ServerSocketChannellImpl.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 and I will write this up later today once I speak with Luke.

7.3.5. 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.

More information here [Documentation/Tutorials/IvySetupWithEclipse](#)

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

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 or better, Intel Duo Core 2, PentiumD) \ 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 and above, Opteron 2Ghz and above) \ 2 - 4 GB Available memory or above \ 1GB Ethernet Card with big pipe network \ 200 MB of available disk space (10K RPM and above SATA II \ SCSI RAID 1-5) \ Network Storage Cluster Solution for mass content storage (ie Isolon, Dell MD1000)
Software Requirements	Java JRE 1.5 or 1.6 \ Service Scripts (Java Service Wrapper, FireDaemon Pro)

Part II. Red5 Core Technologies

Red5 core intro here

- Chapter 10, *Create new applications in Red5*
- Chapter 11, *Deploying Red5 To Tomcat*
- Chapter 12, *Customize Stream Paths*
- Chapter 13, *Security*
- Chapter 14, *Scripting Implementations*
- Chapter 15, *Clustering*
- Chapter 16, *Management*
- Chapter 17, *List of Custom bean definitions*
- Chapter 18, *Red5 Demo Applications*
- Chapter 19, *Testing Red5*

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]

11.1. 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.

11.2. 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` whereas 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.

11.3. 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.

11.4. 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.

11.4.1. 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.

11.4.2. 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>
```

11.4.3. 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 used 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" />
```

11.4.4. 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
```

11.5. 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.

11.5.1. 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".

11.5.2. 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".

11.5.3. Example Source

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

11.6. 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>

```

11.7. 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 Jira 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 (0.6.2, Trunk, Revision 2283, etc...)

11.8. 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.

11.9. Bibliography

- Red5 - <http://osflash.org/red5>
- 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.

12.1. 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.

12.2. 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.

12.3. 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.

12.4. 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.

13.1. Stream Security

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

13.1.1. 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>).

13.1.2. 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:

14.1. 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 preferred over any specified elsewhere.

14.2. 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

14.3. III. Creating an application script

14.3.1. 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 < 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 && @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
end

```

14.3.2. 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 < 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 => 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 %l:%M:%S")
  end
  def method_missing(m, *args)
    super unless @value.respond_to?(m)
    return @value.send(m, *args)
  end
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<String, Map> filesMap = new HashMap<String, Map>();
        Map<String, Object> 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<String, Object>();
                    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<String, Object>();
                    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);
}

```

14.4. IV. 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.

14.5. V. Links with scripting information

- Spring scripting

<http://static.springframework.org/spring/docs/2.0.x/reference/dynamic-language.html> <http://rhinoin.spring.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/>

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.

15.1. 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.

15.2. Server Configuration

15.2.1. 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/.

15.3. 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.

15.3.1. 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
```

15.3.2. 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
```

15.4. 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
```

15.5. Use Your Appliation

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

16.1. 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.

16.2. 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.

16.3. 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.

16.4. 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.

16.5. 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
```

16.5.1. 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"`.

16.5.2. Remote Management

For remote connections with jconsole / JMX clients the command is

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

16.5.3. SSL Remote Management

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

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
```

16.6. 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>

This document gives a list of available custom-bean names

17.1. how to use the custom settings

see: Customize Stream Paths [Documentation/UsersReferenceManual/
Red5CoreTechnologies/Chapter3]

17.2. Bean Definitions

Class	Bean	Description
org.red5.server.api.stream.IStreamableFileConnectionGenerator	streamableFileConnectionGenerator	Chapter13.CustomizeStreamPath
org.red5.io.IStreamableFileService	streamableFileService	?

This document explains how to add and maintain Red5 demo applications which are 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.

18.2. 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.

18.3. 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

18.4. 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

ex: 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.

18.5. 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
```

18.6. 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.

18.7. 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.

18.7.1. 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>

18.7.2. 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);
}
```

18.7.3. ServerClientDetection

The `ServerClientDetection` class detects server to client bandwidth. 3 set of payload data arrays are initialized, 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 occurred.
- 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[]{k

    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});
    }
}

```

18.7.3.1. 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();
}
```

18.7.4. 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.getWrittenBytes(), time});

    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/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):



Note

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

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 RTMPCClient, 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 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 Jira [<http://jira.red5.org/browse/DT>] 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 RTMPCClient.

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 Page for 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/IEchoService.java]
```

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 snapshot 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 the AsUnit 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 new class 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/~aclerke/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) vldeshow.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 Jira [<http://jira.red5.org/browse/DT>] 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 Page for 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:

[Redacted content]

```
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.

Appendix A. RTMPT Specification

A.1. Overview

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.

A.2. 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

A.3. 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.

A.4. Polling interval

The server always starts with a value of 0x01 after data was returned and increases it after 10 empty 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.

A.5. 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.

A.6. 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.

A.7. 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.

A.8. 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 B. Changelog

B.1. Red5 0.7.1 (unreleased)

New Features: - Added socket policy file server to support new security model, starting

Unexpected indentation.

in flash player 9,0,124,0

Block quote ends without a blank line; unexpected unindent.

- Added virtual hosting capabilities (Tomcat only)
- Added W3C log appender for logback modeled after FMS log events and categories
- Added the ability to unload a context using the ContextLoader
- Added RTMPS support (Jira SN-69)
- Set default J2EE servlet container / HTTP server to Tomcat

Bugfixes: - RTMPProtocolDecoder fixed to support RSO sendMessage (Jira CODECS-9)
- Fixed Tomcat logging problem - Fixed memory leak in ServiceUtils - Fixed connection timeout (Jira SN-95 / APPSERVER-274) - Resolved exception with WarLoaderServlet (Jira APPSERVER-224) - Resolved log directory issue (Jira APPSERVER-246) - Resolved ServerStream issue with w3c logging (Jira APPSERVER-263) - Added patch to support ability to implement IBroadcastStream for custom streaming protocols (Jira SN-87)

B.2. Red5 0.7.0 (2008-02-23)

New Features: - Initial Edge/Origin clustering support for multiple Edges with a single

Unexpected indentation.

Origin (Jira APPSERVER-66)

Block quote ends without a blank line; unexpected unindent.

- Added stream listeners that can get notified about received packets
- Support for server-side Javascript (Jira APPSERVER-169)
- Added new base class org.red5.server.adapter.MultiThreadedApplicationAdapter that allows multiple clients to connect simultaneously to the same application
- Added new Flash Player 9 statuses NetStream.Play.FileStructureInvalid and NetStream.Play.NoSupportedTrackFound
- New Flex admin tool (Jira APPSERVER-242)

Bugfixes: - Pause near end of buffered streams works as expected (Jira APPSERVER-199)
- Fixed potential memory leak with RTMPT connections that are not properly

Unexpected indentation.

closed (Jira APPSERVER-193)

Block quote ends without a blank line; unexpected unindent.

- "onMetaData" is only written to newly recorded FLV files and contains valid properties now
- Don't try to decode objects for closed RTMPT connections (Jira APPSERVER-208)
- New multi-threaded connection code fixes various timeout issues (Jira APPSERVER-122, Jira APPSERVER-166 and Jira APPSERVER-167)
- Always use correct classloader inside applications (Jira APPSERVER-200)
- Tomcat cannot undeploy red5 application (Jira APPSERVER-204)
- "ByteArray" objects used old data after calling "compress" or "uncompress" (Jira APPSERVER-211)
- "@DontSerialize" checks for properties also in inherited classes (Jira APPSERVER-225)
- Enabled bidirectional class serialization (Jira APPSERVER-219)
- Array typed parameters in remoting service methods converted properly (Jira APPSERVER-161)

B.3. Red5 0.6.3 (2007-09-17)

New Features: - Remoting requests from "mx:RemoteObject" supported (Jira APPSERVER-144) - RTMPT working with Tomcat - Added thread that writes modified persistent objects periodically.

Unexpected indentation.

This reduces server load if multiple attributes of one object, or the same object is modified frequently.

Block quote ends without a blank line; unexpected unindent.

- Location of "webapps" folder can be configured in bean "jetty6.server" inside "conf/red5.xml" (Jira APPSERVER-152)
- "IStreamFilenameGenerator" can specify if it returns absolute or relative paths
- Applications can be unloaded and loaded without restarting Red5
- "mx.collections.ArrayCollection" objects supported by AMF3 codec
- Object attributes are converted if necessary in AMF0/AMF3 codecs
- "mx.utils.ObjectProxy" objects supported by AMF3 codec (Jira APPSERVER-173)
- "IConnection" objects for Remoting properly store attributes accross multiple requests by using sessions

- Remoting headers are accessible through "IConnection.getConnectionParams"
- "ByteArray" objects supported (Jira APPSERVER-189)
- "NetStream.send" messages are properly passed through from Flex clients (Jira APPSERVER-185)
- Class fields that should not be serialized when sending objects to clients can be annotated with "@DontSerialize" (in "org.red5.annotations")
- Public methods can be protected from being called through RTMP, RTMPT or Remoting by using "@DeclarePrivate" and "@DeclareProtected".
- Support for XML objects added to AMF3 codec (Jira APPSERVER-196)

Bugfixes: - Validate RTMP handshake received from client (Jira APPSERVER-159) - Array typed parameters are converted correctly (Jira APPSERVER-161) - RTMPHandler is wired through Spring (Jira APPSERVER-150) - fixed concurrency issue in RTMP encoder that could result in wrong

Unexpected indentation.

packet header types (Jira APPSERVER-177)

Block quote ends without a blank line; unexpected unindent.

- IStreamAwareScopeHandler methods are also called for server side streams
- "NetConnection.Connect.AppShutdown" is returned when trying to connect to application that currently is unloaded (Jira APPSERVER-13)
- State is properly reset if exceptions occur in package decoding (Jira APPSERVER-137)
- Numbers outside integer range are correctly serialized in AMF3 codec
- return proper error object that triggers "onStatus" for "NetConnection.call" in case of errors (Jira APPSERVER-192)
- Fixed endless loop in playlist controller with only one item in it (Jira APPSERVER-191)
- Fixed renaming across filesystems (Jira SN-59)
- Updated Jetty to 6.1.5 (Jira APPSERVER-123)
- Fixed deserialization of AMF3 encoded SO events (Jira APPSERVER-188)

B.4. Red5 0.6.2 (2007-06-17)

Bugfixes: - "pause" no longer breaks live streams (Jira APPSERVER-136) - Configured subscopes don't get released when a client disconnects - AMF requests could not be decoded when run in the context root

Unexpected indentation.

(Jira APPSERVER-146)

Block quote ends without a blank line; unexpected unindent.

- Fixed bug for Remoting requests without parameters (Jira APPSERVER-147)
- Fixed issue with stop/start of war in Tomcat (Jira APPSERVER-155)
- Fixed handshake reply for Flash Player 9 Update 3
- IMetaData supports fractional framerates (Jira APPSERVER-157)
- Correctly reject empty stream names (Jira APPSERVER-156)
- Fixed problem with loading some JAR files from the applications classpath (Jira APPSERVER-141)
- Fixed decoding of Remoting requests with multiple parameters (Jira APPSERVER-151)

B.5. Red5 0.6.1 (2007-05-23)

New Features: - Switched to use mina 1.1, more config options in red5.properties - Newly recorded files start with an "onMetaData" tag containing the

Unexpected indentation.

duration and the codecs used

Block quote ends without a blank line; unexpected unindent.

- Added a JMX subsystem with RMI and HTTP connectors
- Simplified MBean unregistration and added a registration check prior to the unregister attempt (Jira APPSERVER-118)
- "IServerStream" now also supports "pause" and "seek"
- Enabled RMI + SSL for JMX
- Added JMX authentication
- Added Shutdown class for cleanly shutting down a Red5 instance
- Added support for AMF3 in remoting server
- "receiveAudio" and "receiveVideo" work for VOD streams (Jira SN-22)

Bugfixes: - "NetStream.Record.Failed" is sent for IO errors that occurred during

Unexpected indentation.

recording (Jira APPSERVER-64)

Block quote ends without a blank line; unexpected unindent.

- Fixed possible deadlock if methods are invoked by a connecting client on a client that is currently disconnecting (Jira APPSERVER-108)
- Fixed NPE when connecting without application given (Jira APPSERVER-116)

- Fixed various problems with deserialization of AMF3 objects that implement IExternalizable (Jira CODECS-2)
- Fixed warning about deprecated Jetty configuration (Jira APPSERVER-115)
- Fixed possible deadlock involving PersistableAttributeStore and Scope (Jira APPSERVER-122)
- Display better message if RMI connection to "rmiregistry" could not be established (Jira APPSERVER-125)
- Python scripts can import classes available only in the classpath of a webapp (Jira APPSERVER-92)
- Fixed Ruby application issue by updating to Spring 2.0.5 and JRuby 0.9.8 (Jira APPSERVER-93)
- Fixed async calling of remoting methods (Jira APPSERVER-131)
- Accessing root of RTMPT server no longer results in 404 but redirects to HTTP port (Jira APPSERVER-130)
- Disconnect clients that don't send a valid handshake (Jira APPSERVER-128)
- Reduced max. idle time to prevent too many open sockets when using RTMPT with HTTP/1.0 (Jira APPSERVER-87)
- Fixed potential NPEs in PlaylistSubscriberStream (Jira SN-40)
- Fixed various problems with deserializing AMF0 references in remoting
- Fixed frozen video if audio is disabled in live streams (Jira SN-22)

B.6. Red5 0.6 (2007-04-23)

New features: - Recording/playback of files to/from subscopes implemented

Unexpected indentation.

(Jira APPSERVER-103)

Bugfixes: - Ghost connection detection code rewritten to better detect dead clients

Unexpected indentation.

(Jira APPSERVER-38, SN-37)

Block quote ends without a blank line; unexpected unindent.

- Deserialization of objects defined in webapp classpath fixed (Jira APPSERVER-80, APPSERVER-100)
- Fixed AMF3 deserializer for references from attributes to parent classes (Jira APPSERVER-101)
- Jython example adjusted for new bandwidth API (Jira APPSERVER-92)

- Workaround added to deal with broken MP3 files (Jira APPSERVER-62)
- "start" and "length" are properly evaluated when playing back VOD streams
- Fixed seeking not working for MP3 or audio-only FLV files
- Don't log contents of wrong objects (Jira APPSERVER-109)
- Fixed potential NPEs in PlaylistSubscriberStream
- A client buffer of 0 on live streams no longer breaks playback (Jira CS-3)
- Fixed shutdown error in Tomcat with WAR version by updating to SLF4J 1.3.1 (Jira APPSERVER-107)
- "NetStream.Play.InsufficientBW" is sent if client is too slow receiving video streams (Jira APPSERVER-51)
- Improved frame dropping code for slow connections

B.7. Red5 0.6rc3 (2007-04-11)

New features: - Keyframe informations are cached so files don't need to be reparsed

Unexpected indentation.

before playback

Block quote ends without a blank line; unexpected unindent.

- Connections from Flash Media Encoder and On2 Flix Live supported
- Access to shared objects can be limited (Jira APPSERVER-25)
- Connections can provide a list of remote addresses. This is usefull for proxied RTMPT connections.

Bugfixes: - Bandwidth control code has been rewritten to fix stability issues and

Unexpected indentation.

memory leaking in high concurrency connection count situations

Block quote ends without a blank line; unexpected unindent.

- Serialization of Maps with non-number keys fixed (Jira APPSERVER-60)
- Multiple IO processor threads are used by default
- Memory leak when closing RTMPT connections fixed (Jira APPSERVER-61)
- Merged WAR build script with primary script, also moved WAR specific startup servlet into trunk
- Deserializing of remoting results fixed (Jira APPSERVER-63)
- Fixed "error in object encoding" when rejecting AMF3 clients (Jira APPSERVER-73)

- Concurrency problems when closing a connection fixed (Jira APPSERVER-59)
- Unnecessary NetStream.Play.* events are no longer sent when playback stopped (Jira APPSERVER-70)
- SimplePlaylistController setRepeat and setRandom fixed (Jira SN-27)
- NPE in SimpleBWControlService fixed (Jira APPSERVER-75)
- Reference bugs in AMF3 encoder fixed (Jira APPSERVER-81)
- "NetStream.Play.Failed" is sent correctly now (Jira APPSERVER-52)
- Concurrency issue fixed in SimpleBWControlService (Jira SN-32)
- Fixed problem when decoding MP3 files with signed values in the ID3v2 tag size (Jira APPSERVER-86)
- "NetStream.Seek.Failed" is sent when trying to seek in live streams (Jira APPSERVER-84)
- "NetStream.Failed" is sent for exceptions during streaming methods (Jira APPSERVER-85)
- Random server freezing resolved (Jira APPSERVER-41)
- Send correct timestamps if seeking beyond end of file (Jira APPSERVER-54)
- Fixed NoSuchElementException when iterating connections during disconnect (Jira APPSERVER-94)
- Reference bugs in AMF3 decoder fixed (Jira APPSERVER-95)
- "NetStream.Play.Complete" is sent (APPSERVER-50)
- "NetStream.Play.Switch" is sent (APPSERVER-82)
- Streams are always played to the end (SN-8)
- Seeking in stopped streams fixed (APPSERVER-89)
- Fixed deadlock in shared objects under high load (APPSERVER-98)

B.8. Red5 0.6rc2 (2007-02-12)

New features: - Stream classes can be configured through red5-common.xml (Trac #223)
- RTMP network library supports client mode (Trac #94) - Source of VOD streams can be customized through IStreamFilenameGenerator

Unexpected indentation.

(Trac #120)

Block quote ends without a blank line; unexpected unindent.

- API: IStreamFilenameGenerator differs between playback and recording

- Results of method calls can be deferred until they are available to free io threads
- Transient fields will not be serialized any longer (Jira APPSERVER-27)
- Red5 compiles with Java6 now
- Support for AMF3 incl. IExternalizable objects added (Jira APPSERVER-31)
- Access to streams can be limited (Jira APPSERVER-25)
- (non-persistent) shared objects can be acquired by serverside code to prevent them from being released when the last client disconnects (Jira APPSERVER-48)

Bugfixes: - Serialize RecordSet objects (Trac #201) - "NetConnection.Connect.Rejected" is sent for non-existing scopes to

Unexpected indentation.

match result code of FCS/FMS

Block quote ends without a blank line; unexpected unindent.

- RTMPT through Jetty working again (Trac #213)
- Size of last frame is correctly written to .flv files
- Errors during "connect" are reported back to client through RTMPT
- Fixed NPE in FlowControlService thread (Trac #175)
- Deserializing of mixed arrays now works in all cases (Trac #109, #195)
- "NetStream.Record.Start" and "NetStream.Record.Stop" are sent (Trac #127)
- "NetStream.Publish.BadName" is sent if two clients try to publish/record a stream with the same name
- Streams stopped if bandwidth limit was set too high (Trac #165)
- Fixed potential concurrency issue in FlowControlService (Trac #224)
- Stream notification callbacks are invoked on reused connections (Trac #133)
- The playlist is flushed by default (Jira APPSERVER-6)
- Fixed ClassCastException in "pendingVideoMessages" (Jira APPSERVER-14)
- calling "pause" with null argument works again (Jira APPSERVER-12)
- "NetStream.Publish.BadName" is only sent if another client is already publishing a stream
- Playing a stream while being recorded now works (Jira SN-4, SN-13)
- "IPendingServiceCall.isSuccess()" returns true when a result has been received (Jira APPSERVER-35)

- The "http.host" setting from "red5.properties" is evaluated (Jira APPSERVER-36)
- "IBroadcastStream" knows about the filename it is being recorded to (Jira APPSERVER-30)
- BufferOverflowException for empty RTMP packets fixed (Jira APPSERVER-37)
- FLV files are no longer locked after playback (Jira APPSERVER-17)
- SharedObjects support "getAttributes" (Jira APPSERVER-45)
- MP3 files containing images can be played back (Jira APPSERVER-47)
- Fixed parsing of long strings (Jira APPSERVER-44)
- Fixed pausing and seeking audio-only flv files (Jira SN-17)
- Number of streams is no longer limited (Jira SN-14)
- "NetStream.Play.Failed" is returned if a VOD stream can not be played due to IO errors (Jira APPSERVER-52)
- "NetStream.InvalidArg" is returned for invalid arguments (Jira APPSERVER-55)
- "NetConnection.Connect.InvalidApp" is returned for non-existing application scopes on the server
- "NetStream.Record.NoAccess" is returned if file could not be created or written to (Jira APPSERVER-53)
- Error when setting SO attributes fixed (Jira APPSERVER-57)

B.9. Red5 0.6rc1 (2006-10-30)

New features: - Created WAR (Web Application Archive) version of Red5

Unexpected indentation.

(Separate repository java/war)

Block quote ends without a blank line; unexpected unindent.

- Enabled Tomcat or Jetty as J2EE container implementations
- FLV cache implementations (2 are included) (Trac #99)
- Scripting support (javascript, ruby, python, groovy, and bsh) based on Spring 2 and JSR223

Bugfixes: - Last frames aren't lost when reading .flv files (Trac #90) - FileConsumer acted on all consumer pipe events (Trac #92) - Improved timestamps of live streams to be more in sync with FMS (Trac #93) - FileConsumer modified position of incoming messages (Trac #91) - Events should support reference counting (Trac #103) - ServerStream playback jerky (Trac #77) - "NetStream.send" events are properly recorded - Reusing streams works (Trac #123) - Fixed NPE if no bandwidth settings are available (Trac #129) - "close" can be

called on RTMPT connections multiple times (Trac #166) - Fixed synchronizing problem with clients publishing repeatedly (Trac #124) - RTMPT connections can be closed from the serverside (Trac #179)

B.10. Red5 0.5 (2006-07-25)

New features: - Frame dropping for live streams depending on available bandwidth - Added "receiveAudio", "receiveVideo" and "send" for streams - Destination of recorded streams can be customized (Trac #73) - VOD stream flow control adapts bandwidth based on buffer time (Trac #63) - Up-/downstream bandwidth can be specified

Bugfixes: - Only the same instances are serialized as references (Trac #58) - Re-added JSP support in manifest file of red5.jar (Trac #59) - "tagPosition" is updated in FLVReader when seeking (Trac #55) - Automatic subscopes of the host scope are disabled so only connections

Unexpected indentation.

to existing applications are possible

Block quote ends without a blank line; unexpected unindent.

- Running "ant" after setup keeps wrapper configuration (Trac #76)
- MP3 files with unsupported sample rates are detected (Trac #66)
- Timestamps of recorded .flv files were wrong sometimes (Trac #78)
- Stream types could be reused leading to a ClassCastException (Trac #84)
- "ns.pause" working if no flag given (Trac #67)
- A keyframe is sent for paused streams when seeking

B.11. Red5 0.5rc1 (2006-07-11)

New features: - Refactored streaming code - Refactored scope services - Refactored rtmp message de-/encoding - Enabled subscopes - Bandwidth control for on-demand streams - Experimental support for serverside streams - Added dynamic "onMetaData" for mp3 streams - Added persistence for scopes and shared objects - Added support for simple "directory-only" applications - Added remoting client support (sync / async) - Added deserializer for RecordSet remoting results - Arbitrary objects can be registered as service handlers - IClientRegistry can be customized for each scope - WEB-INF directories are added to the classpath (Trac #27) - Clients can be rejected with a custom error message - Basic "onMetaData" is generated dynamically for .flv files without any

Unexpected indentation.

meta data (Trac #23)

Bugfixes: - MP3 files that have their protection bit set - MP3 files encoded MPEG 2, Layer III (Trac #15) - MP3 files with incomplete last frame - Shared objects bugfixes (Trac #11, #22, #25) - Application handlers were not called on disconnect - IConnection.close() now

closes connection (Trac #19) - Connecting to non-existent applications returns correct error now - Jetty correctly runs on all virtual hosts (Trac #26) - Map objects are serialized correctly - Methods could be invoked with converted parameters before invoking them

Unexpected indentation.

with the original parameters

Block quote ends without a blank line; unexpected unindent.

- Support invoking methods with "null" as parameter (Trac #29)
- Directories for recorded files are created if they don't exist (Trac #20)
- "pause(java.lang.Object, int)" was reversed for streams (Trac #16)
- Serialization of arbitrary objects uses reflect api to access fields, fixes various problems with inner classes and internal objects like IConnection / IClient
- Invalid stream ids are handled in "deleteStream" (Trac #21)
- Stream name prefixes and names without extensions supported (Trac #28)

B.12. Red5 0.4.1 (2006-05-01)

- MP3 audio streams
- "seek" and "pause" for on-demand streams (Trac #4)
- "Address already in use" fixed after restart (Trac #5)
- Bugfixes for shared objects (Trac #6)
- Bugfixes for videoconference sample (Trac #7)
- Connection strings without hostname supported (Trac #8)
- Flash 7 version of the videoconference sample added

B.13. Red5 0.4 (2006-04-20)

- Public server-side api
- AMF remoting
- RTMPT
- Metadata API
- Basic samples and documentation

B.14. Red5 0.3 (2006-02-21)

- Live streams

- Shared objects

B.15. Red5 0.2 (2005-10-21)

- First public release
- Video streams
- Echo service