Open Source Flash Server

Red5 - Reference Documentation

Version 0.7.1

Steven Gong, Paul Gregoire, Daniel Rossi

Preface vii
1. Introduction 1
2. What's new in Red5 0.7.1 2
2.1. Introduction 2
I. Getting Started 3
3. Frequently Asked Questions 4
3.1. Downloading 4
3.1.1. Where can I download Red5? 4
3.2. General 4
3.2.1. What is Red5? 4
3.2.2. Why does Red5 exist? 4
3.2.3. What does Red5 mean? 4
3.2.4. Why is it not called Red5 Media Server? 4
3.2.5. What other names does Red5 go by? 4
3.2.6. What does the Red5 logo look like? 4
3.2.7. What would be the best possible scenario that could come out of this project? 4
3.2.8. What license does Red5 use? 5
3.2.9. What is the estimated time of delivery for Red5 (eta) 5
3.3. Protocols 5
3.3.1. RTMP 5
3.3.2. AMF 5
3.4. Server Development 5
3.4.1. What implementations are currently being implemented? 5
3.4.2. How far along is the Java implementation? 5
3.4.3. Why did the Red5 team choose Mina? 5
3.4.4. Why did the Red5 team choose Spring? 6
3.5. Client Development 6
3.5.1. Which component frameworks are we currently reviewing? 6
3.5.2. What are the advantages of this framework? 6
3.5.3. Do I have to use this component framework? 6
3.5.4. Can I use Macromedia Flash Communication Server components? 6
3.6. Developer Resources 6
3.6.1. What IDE (integrated development environment) are we using?..... 6
3.6.2. Is there a place that we can meet and talk about the project status? 6
3.6.3. Is there a mailing list ? 6
3.6.4. Where's the best place to start? 6
3.7. Team Members 7
3.7.1. Flash Codecs. AMF, AMF3, RTMP, FLV 7
3.7.2. Streaming and Networking 7
3.7.3. App Server 7
3.7.4. Client Side. Flash, HTML, AJAX, Admin Interface 7
3.7.5. Docs and Tutorials 7
3.7.6. Website and Marketing 7
3.7.7. Project Management 7
3.8. Accomplishments 8
3.8.1. What are the milestones? 8

4. Configuration files used by Red5 9
4.1. Preface 9
4.2. Directory "conf" 9
4.2.1. jetty.xml 9
4.2.2. keystore 9
4.2.3. log4j.properties 9
4.2.4. realm.properties (Jetty) 9
4.2.5. tomcat-users.xml (Tomcat) 9
4.2.6. red5.globals 10
4.2.7. red5.properties 10
4.2.8. red5.xml 10
4.2.9. red5-common.xml 10
4.2.10. red5-core.xml 12
4.2.11. red5-rtmpt.xml 12
4.2.12. web.xml (Tomcat) 12
4.2.13. web-default.xml (Jetty) 12
4.3. Webapp config directory 13
4.3.1. red5-web.xml 13
5. Migration Guide 14
5.1. Preface 14
5.2. Application callbacks 14
5.2.1. Interface IScopeHandler 14
5.2.2. Class ApplicationAdapter 14
5.2.3. Accepting / rejecting clients 15
5.3. Current connection and client 16
5.4. Additional handlers 16
5.4.1. Handlers in configuration files 17
5.4.2. Handlers from application code 17
5.5. Calls to client methods 18
5.6. SharedObjects 19
5.6.1. Serverside change listeners 19
5.6.2. Changing from application code 20
5.6.3. SharedObject event handlers 21
5.7. Persistence 21
5.8. Periodic events 23
5.9. Remoting 24
5.9.1. Remoting server 24
5.9.2. Remoting client 25
5.10. Streams 26
6. Red5 Libraries 27
6.1. Spring scripting support 27
6.2. Groovy 27
6.3. Beanshell 27
6.4. Ruby 27
6.5. Jython / Python 27
6.6. Java 5 Libraries 27
6.7. Script related JSR's 28
6.8. Javascript / Rhino 28
7. Build Environment Setup 29

7.1. Introduction 29
7.2. Ant 29
7.3. Java 29
8. Building Red5 30
8.1. Introduction 30
8.2. Running the ant build 30
8.3. Red5 debian package 30
9. How to build with eclipse 31
9.1. Importing the Red5 Project 31
9.2. Debugging Red5 in Eclipse 32
10. Releasing Red5 34
II. Red5 Core Technologies 35
11. Create new applications in Red5 36
11.1. Preface 36
11.2. The application directory 36
11.3. Configuration 36
11.3.1. webAppRootKey 36
11.4. Handler configuration 36
11.4.1. Context 36
11.4.2. Scopes 37
11.4.3. Handlers 38
11.5. Sample handler 39
12. Create new applications in Red5 WAR 40
12.1. Preface 40
12.2. The application directory 40
12.3. Configuration 40
12.3.1. globalScope 40
12.3.2. contextConfigLocation 40
12.3.3. listener (start-up / shutdown) 41
12.3.4. parentContextKey 41
12.3.5. log4jConfigLocation 41
12.4. Handler configuration 41
12.4.1. Context 41
12.4.2. Scopes 42
12.4.3. Handlers 43
13. Build Environment Setup 44
13.1. Filename generator service 44
13.2. Custom generator 44
13.3. Activate custom generator 45
13.4. Change paths through configuration 45
14. Security 47
14.1. Streams 47
14.1.1. Stream playback security 47
14.1.2. Stream publishing security 47
14.2. Shared objects 49
15. Scripting Implementations 51
15.1. I. Select a scripting implementation 51
15.2. II. Configuring Spring 51
15.3. III. Creating an application script 53

15.4. 1. Application adapter 53
15.5. 2. Application services 54
15.6. IV. Creating your own interpreter 58
15.7. V. Links with scripting information 58
16. Clustering 61
16.1. Server Configuration 61
16.2. Configure Edge Server 61
16.3. Edge on a different Server from Origin 61
16.4. Edge on the same Server as Origin 61
16.5. Configure Origin Server 61
17. Management 62
A. RTMPT Specification 64
A.1. Overview 64
A.2. URLs 64
A.3. Request / Response 64
A.4. Polling interval 65
A.5. Initial connect (command "open") 65
A.6. Client updates (command "send") 65
A.7. Polling requests (command "idle") 65
A.8. Disconnect of a session (command "close") 65
B. Changelog 66
B.1. Red5 0.7.1 (unreleased) 66
B.2. Red5 0.7.0 (2008-02-23) 66
B.3. Red5 0.6.3 (2007-09-17) 67
B.4. Red5 0.6.2 (2007-06-17) 68
B.5. Red5 0.6.1 (2007-05-23) 69
B.6. Red5 0.6 (2007-04-23) 70
B.7. Red5 0.6rc3 (2007-04-11) 71
B.8. Red5 0.6rc2 (2007-02-12) 72
B.9. Red5 0.6rc1 (2006-10-30) 74
B.10. Red5 0.5 (2006-07-25) 75
B.11. Red5 0.5rc1 (2006-07-11) 75
B.12. Red5 0.4.1 (2006-05-01) 76
B.13. Red5 0.4 (2006-04-20) 76
B.14. Red5 0.3 (2006-02-21) 76
B.15. Red5 0.2 (2005-10-21) 77

Preface

preface

overview here

2.1. Introduction

New features.

Part I. Getting Started

Red5 intro here

- Chapter 3, *Frequently Asked Questions*
- Chapter 4, *Configuration files used by Red5*
- Chapter 5, *Migration Guide*
- Chapter 6, *Red5 Libraries*
- Chapter 7, *Build Environment Setup*
- Chapter 8, *Building Red5*
- Chapter 9, *How to build with eclipse*
- Chapter 10, *Releasing Red5*

Dominick Accattato

Published 2005-10-24

3.1. Downloading

3.1.1. Where can I download Red5?

Mirrors: http://osflash.org/red5#conference_links

3.2. General

3.2.1. What is Red5?

An open source project dedicated towards the interaction between the Flash Player and a Free Connection Oriented Server using rtmp (real time messaging protocol).

3.2.2. Why does Red5 exist?

Like most open source projects, the project exists because there was interest in the topic. Even before any code was written people had been dissecting the bytes that come down the pipe from the flash comm server and flash player interaction.

3.2.3. What does Red5 mean?

Please read: Why Red5 [http://osflash.org/red5#why_red5]

3.2.4. Why is it not called Red5 Media Server?

Red5 does much more than server up media. We feel that Red5 is a technology.

3.2.5. What other names does Red5 go by?

Red5 aka R5

3.2.6. What does the Red5 logo look like?

RTMP (real time messaging protocol) has been introduced to the Flash Platform as a proprietary closed source protocol. Can we legally create source code that may unveil the true workings behind this protocol?

Yes, examining packets is not illegal. The Red5 team has stated its intent to remain completely legal in this situation, and thanks to some outstanding developers in the Flash world, we've never had to even consider any other alternative.

3.2.7. What would be the best possible scenario that could come out of this project?

Macromedia would endorse us and provide the open source community with the RTMP protocol specifications. This would greatly help out the few who are coding ?Free Servers? out there including the Red5 open source project.

3.2.8. What license does Red5 use?

Red5 uses the LGPL license.

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

3.2.9. What is the estimated time of delivery for Red5 (eta)

Please Review: Project Roadmap
[http://review.codegent.net/opensource/RED5_draft_roadmap.pdf]

3.3. Protocols

3.3.1. RTMP

Real Time Messaging Protocol (RTMP) Is a proprietary protocol developed by Macromedia [http://en.wikipedia.org/wiki/Macromedia] that is primarily used with Macromedia [http://en.wikipedia.org/wiki/Macromedia]'s Media Server product to stream audio and video over the web.

The default connection port is 1935.

wikipedia [http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol]

Red5 page dedicated to RTMP: http://osflash.org/rtmp_os

3.3.2. AMF

Action Message Format (AMF) is a binary message format designed for the ActionScript object model.

3.4. Server Development

3.4.1. What implementations are currently being implemented?

Java and Ruby. Currently all focus has been on the Java implementation.

3.4.2. How far along is the Java implementation?

There is a codebase checked into subversion. The code currently uses two frameworks, Mina and Spring discussed below. The protocol handler has been created and output of byte information is viewable. Team members are currently testing the code and sending different rtmp calls from the flash player to figure out how to deal with the rest of the rmtp protocol.

3.4.3. Why did the Red5 team choose Mina?

It allowed us to focus on the protocol, and not implementing low level nio code. We also plan to implement other protocols / transports in the future so having a standard framework is good.

I looked at a number of frameworks for this. Mina, EmberIO, Mule, etc. Mina seemed to be the most focused and developed (essentially being v2 of Netty). EmberIO is quite

similar and something we should look into in the future, esp the threading stategies but not as mature or documented as a framework. Mule seems to be message exchange / network framework on speed. It does everything which I think is too much for our stage of development.

3.4.4. Why did the Red5 team choose Spring?

TODO

3.5. Client Development

3.5.1. Which component frameworks are we currently reviewing?

Actionstep, ASwing

3.5.2. What are the advantages of this framework?

TODO

What is the estimated time of delivery for these components (eta)?

TODO

3.5.3. Do I have to use this component framework?

TODO

3.5.4. Can I use Macromedia Flash Communication Server components?

TODO

3.6. Developer Resources

3.6.1. What IDE (integrated development environment) are we using?

I'm using eclipse. I know a few others are too. As well, you will need to install the subversion plugin so that you can check out the code base. Additionally, you should install one of the actionscript plugins if you plan to do any of the front end coding.

3.6.2. Is there a place that we can meet and talk about the project status?

Yes, you can join a few of us developers on IRC (internet relay chat). Connect to the irc.freenet.org server and then join room #red5

3.6.3. Is there a mailing list ?

Yes, red5@osflash.org [mailto:red5@osflash.org]. You can find more information on the site http://osflash.org/doku.php?id=red5

3.6.4. Where's the best place to start?

Read about the protocol. http://osflash.org/doku.php?id=red5#protocol. Navigate the Red5 site, learn the basics behind the frameworks. Then check out the codebase and

discuss through one of the many communication channels (irc, mailing list). If you are interested in becoming a Red5 team member please fill out the Team Signup Form [http://osflash.org/red5_signupform] and email it to the project managers.

3.7. Team Members

3.7.1. Flash Codecs. AMF, AMF3, RTMP, FLV

- Luke Hubbard [http://osflash.org/luke_hubbard]
- Paul Gregoire [http://osflash.org/paulgregoire]

3.7.2. Streaming and Networking

- Luke Hubbard [http://osflash.org/luke_hubbard]
- Joachim Bauch [http://osflash.org/joachim_bauch]
- Steven Gong [http://osflash.org/steveng]
- Paul Gregoire [http://osflash.org/paulgregoire]

3.7.3. App Server

- Joachim Bauch [http://osflash.org/joachim_bauch]
- Paul Gregoire [http://osflash.org/paulgregoire]
- Michael Klishin [http://osflash.org/michaelklishin]
- Dominick Accattato [http://osflash.org/dominick_accattato]

3.7.4. Client Side. Flash, HTML, AJAX, Admin Interface

- John Grden [http://osflash.org/john_grden]
- Thijs Triemstra [http://osflash.org/thijs_triemstra]

3.7.5. Docs and Tutorials

- Joachim Bauch [http://osflash.org/joachim_bauch]
- Michael Klishin [http://osflash.org/michaelklishin]
- Thijs Triemstra [http://osflash.org/thijs_triemstra]

3.7.6. Website and Marketing

- John Grden [http://osflash.org/john_grden]
- Chris Allen [http://osflash.org/chris_allen]

3.7.7. Project Management

- John Grden [http://osflash.org/john_grden]

- Chris Allen [http://osflash.org/chris_allen]

3.8. Accomplishments

3.8.1. What are the milestones?

TODO

Joachim Bauch

Paul Gregoire

0.6-dev

4.1. Preface

This document describes the configuration files used by Red5.

Please note that this document is still *work in progress*, so some things may (and surely will) change until the final release of Red5!

4.2. Directory "conf"

4.2.1. 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.

4.2.2. keystore

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

4.2.3. 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/].

4.2.4. 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

4.2.5. 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

4.2.6. 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".

4.2.7. red5.properties

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

4.2.8. 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="shu
   ... cut for brevity ...
</bean>
```

4.2.9. 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.

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

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

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

- Red5 homegrown simple example

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

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

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

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

```
<bean id="object.cache" class="org.red5.server.cache.WhirlyCacheImpl" init-method="init" autowire="b
<property name="maxEntries" value="5" />
<property name="cacheConfig">
<bean class="com.whirlycott.cache.CacheConfiguration">
<property name="name" value="flv.cache" />
<property name="maxSize" value="5" />
<!-- This policy removes cached items, biased towards least frequently used (LFU) Items -->
<property name="policy"><value>com.whirlycott.cache.policy.LFUMaintenancePolicy</value></property>
<!-- This policy removes cached items, biased towards least recently used (LRU) Items -->
<!-- property name="policy"><value>com.whirlycott.cache.policy.LRUMaintenancePolicy</value></propert
<!-- This policy removes cache items in the order in which they were added -->
```

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

4.2.10. 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="b
```

- Tomcat

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

4.2.11. 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.

4.2.12. 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

4.2.13. 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.

4.3. Webapp config directory

4.3.1. 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>
```

Joachim Bauch

0.6-dev

5.1. Preface

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.

5.2. 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.

5.2.1. 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.

5.2.2. 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

FCS / FMS	Red5
	roomDisconnect appLeave roomLeave

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

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

5.2.2.1. Execution order of connection methods

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

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

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

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

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

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

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

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

5.2.3. Accepting / rejecting clients

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

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

5.3. Current connection and client

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

```
import org.red5.server.api.IClient;
import org.red5.server.api.IConnection;
import org.red5.server.api.IScope;
import org.red5.server.api.Red5;

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

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

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

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

5.4. 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.

5.4.1. 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*.

5.4.2. 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.

5.5. 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.

5.6. 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].

5.6.1. 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
       implements ISharedObjectListener {

  public void onSharedObjectUpdate(ISharedObject so,
                                   String key, Object value) {
      // The attribute &lt;key&gt; of the shared object &lt;so&gt;
      // was changed to &lt;value&gt;.
  }

  public void onSharedObjectDelete(ISharedObject so, String key) {
      // The attribute &lt;key&gt; of the shared object &lt;so&gt; was deleted.
  }

  public void onSharedObjectSend(ISharedObject so,
                                 String method, List params) {
      // The handler &lt;method&gt; of the shared object &lt;so&gt; was called
      // with the parameters &lt;params&gt;.
  }

  // Other methods as described in the interface...
}
```

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

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

5.6.2. 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*.

5.6.3. SharedObject event handlers

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" />
```

5.7. 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.

5.8. Periodic events

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

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

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

```
import org.red5.server.api.IScope;
import org.red5.server.api.IScheduledJob;
import org.red5.server.api.ISchedulingService;
import org.red5.server.adapter.ApplicationAdapter;

class MyJob implements IScheduledJob {

  public void execute(ISchedulingService service) {
      // Do something
  }
}

public class SampleApplication extends ApplicationAdapter {

  public boolean roomStart(IScope room) {
      if (!super.roomStart(room))
          return false;

      // Schedule invokation of job every 10 seconds.
      String id = addScheduledJob(10000, new MyJob());
      room.setAttribute("MyJobId", id);
      return true;
  }
}
```

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

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

5.9. 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.

5.9.1. 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:

```
<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
```

5.9.2. 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);
```

5.10. Streams

TODO: How can streams be accessed from an application?

6.1. Spring scripting support

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. Introduction

This chapter covers the work environment setup for building red5 and the tools you will need.

7.2. 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

7.3. Java

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

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

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

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

JAVA_HOME=C:\development\j2sdk\1.5.0_07 JAVA_VERSION=1.5 You can check this on windows by typing set JAVA_HOME or on unix by typing echo $JAVA_HOME

Caution

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]

8.1. Introduction

This chapter covers the work environment setup for building red5 with ant.

8.2. 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
```

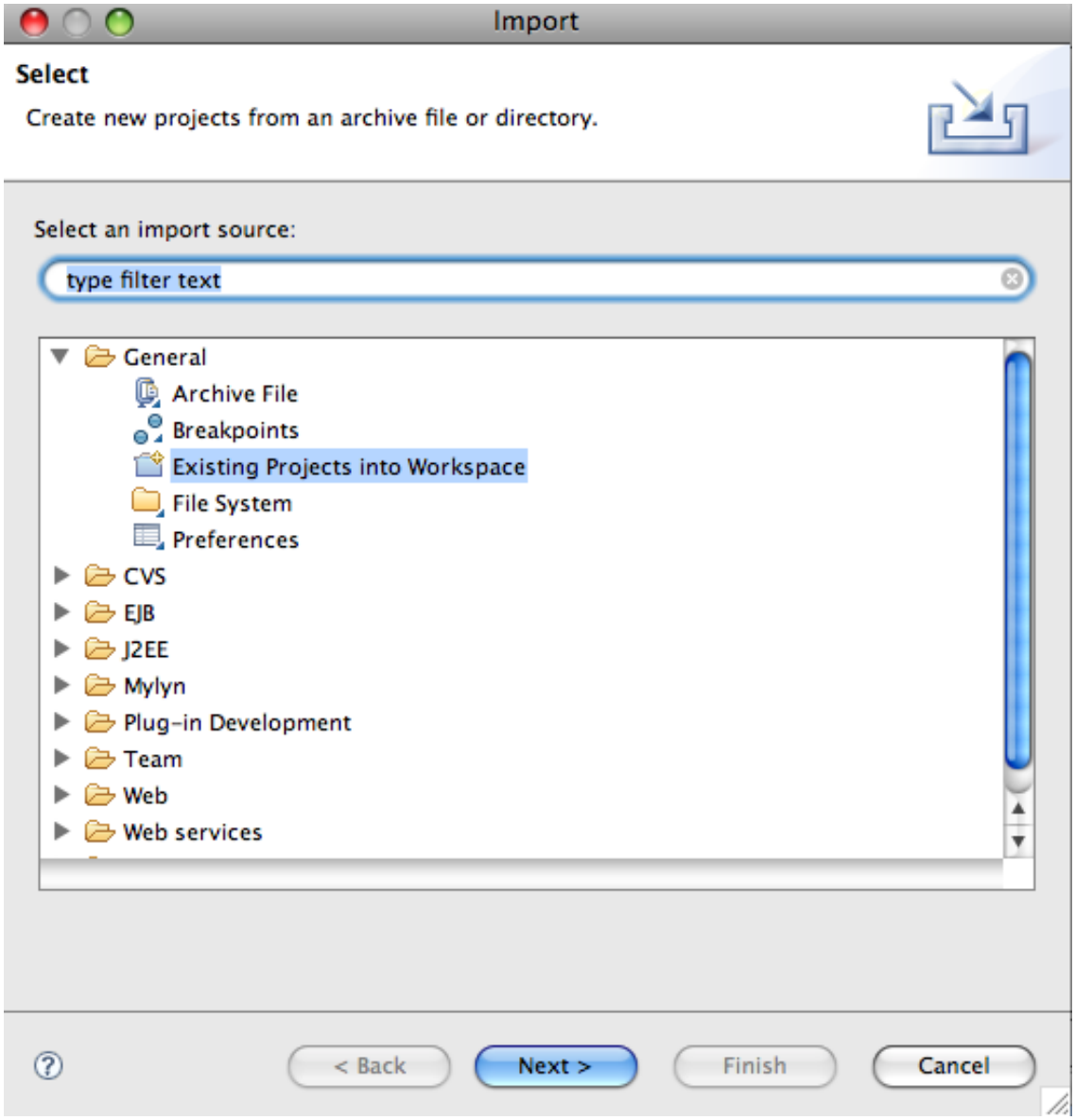
8.3. Red5 debian package

1. Install the debian packages "dpkg-dev", "debhelper", "dh-make", "devscripts" and "fakeroot".
2. Checkout the debian build scripts to a folder "debian" inside the Red5 root from http://svn1.cvsdude.com/osflash/red5/debian/trunk
3. Update "debian/changelog" and add an entry for the new version you are building. Note that the syntax must match the previous entries!
4. Update the filename in "debian/files" to match the version you are building.
5. Make sure you run from a clean Red5 checkout of the tag to build!!!
6. From the red5 root run "dpkg-buildpackage -uc -b -rfakeroot"
7. If all goes well, you should have a debian package one folder below the Red5 root.

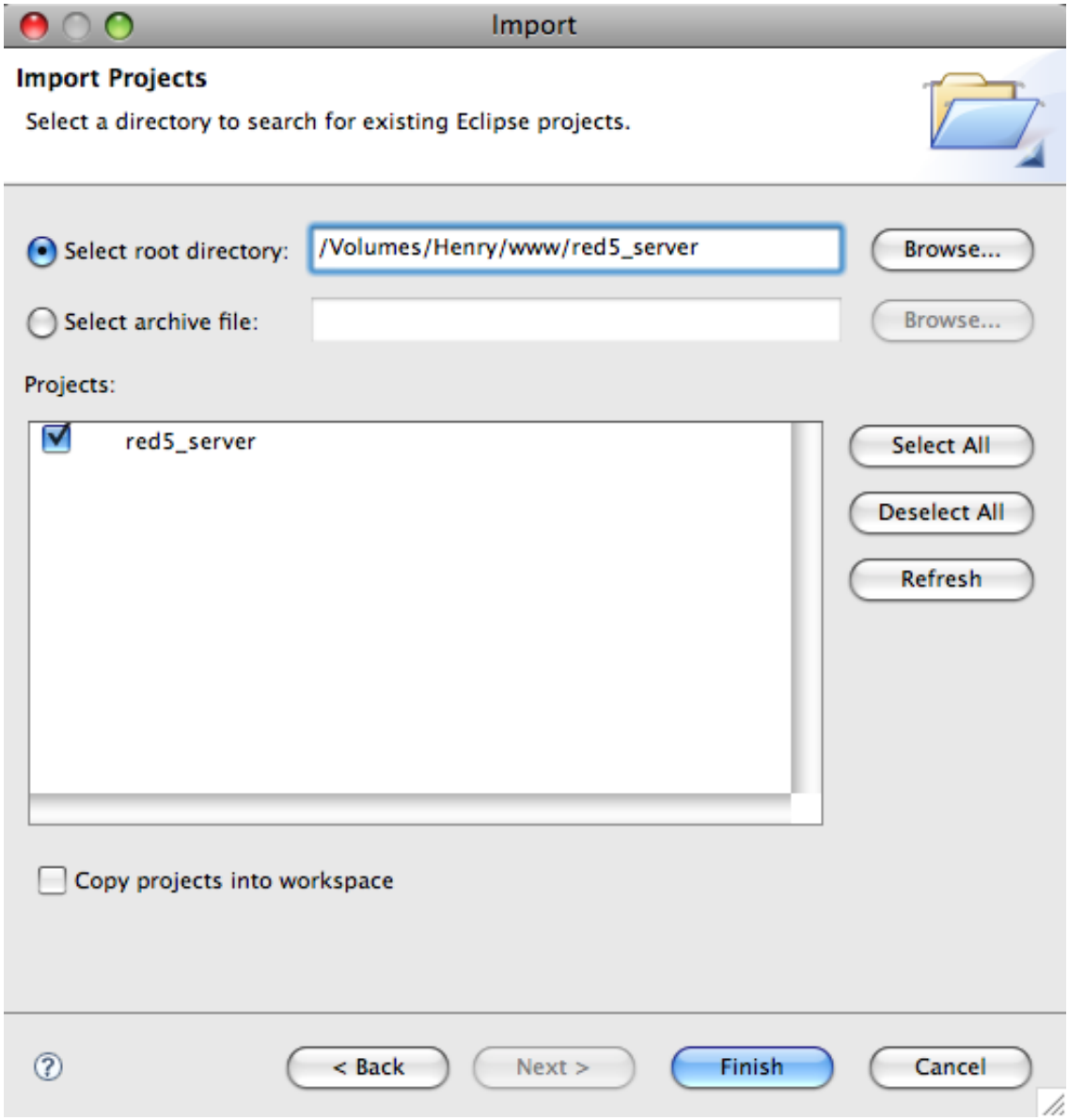
This guide assumes eclipse 3.1.0 and you have downloaded the entire red5 build from the subversion repository at http://svn1.cvsdude.com/osflash/red5

9.1. Importing the Red5 Project

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
6. Make sure red5 is selected in the projects area and hit "Finish"



7. Eclipse should automatically build the project, you can force a build from the "project" menu and selecting "build project"

9.2. Debugging Red5 in Eclipse

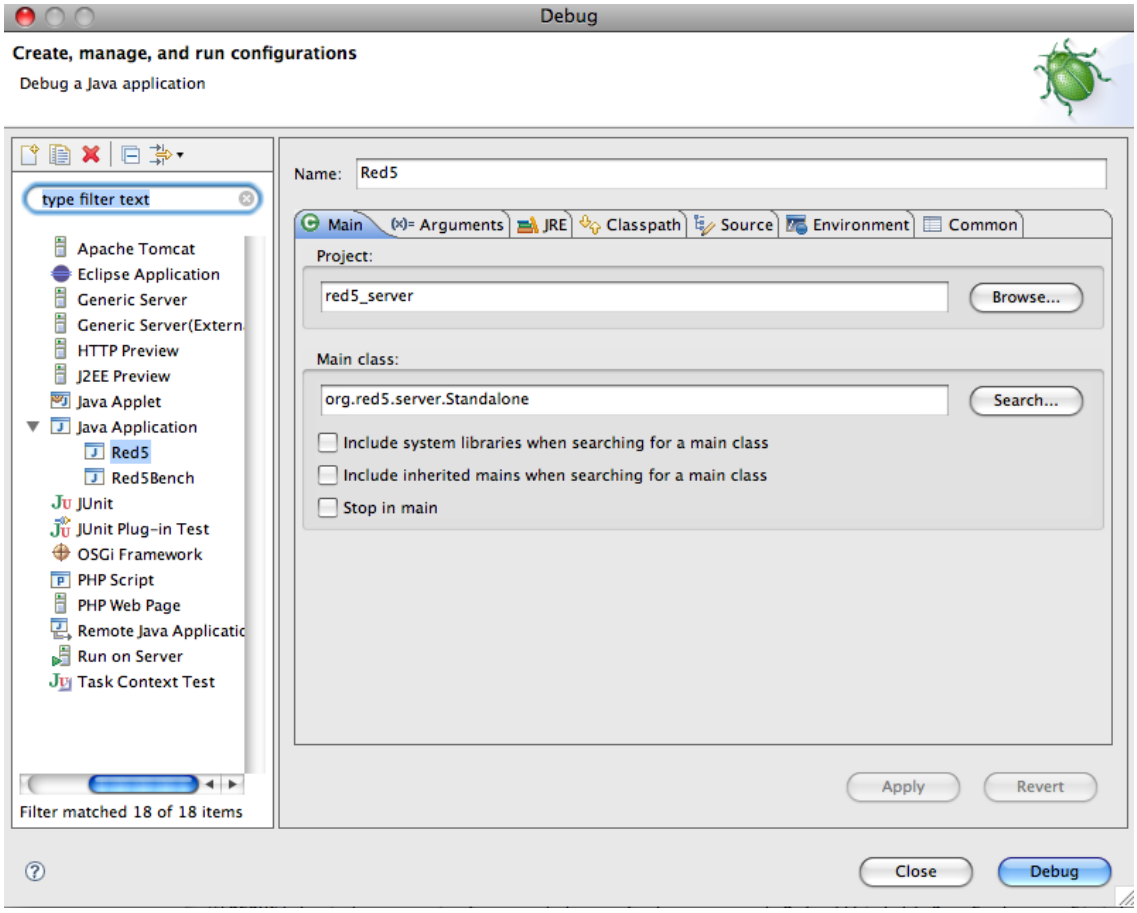
1. Click the arrow next to the Debug icon menu

   and then click "Open Debug Dialog".

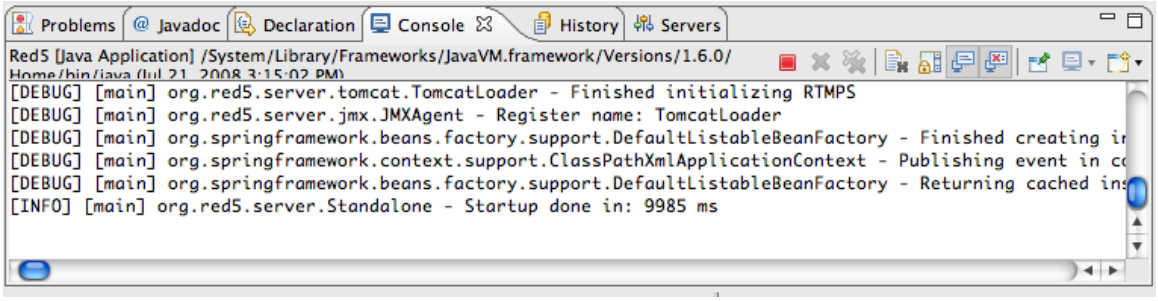
2. Click "Java Application" in the menu then right click and "New".

3. Type a name for the debug configuration (ie "Red") and type "org.red5.server.Standalone" an the main class. Add extra VM arguments in the arguments tab if required.



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

5. Console logging will appear in the console window.



If you get an error in the console like :

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

Then the socket red5 wants to run is in use, you can change the socket and I will write this up later today once I speak with Luke.

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://svn1.cvsdude.com/osflash/red5/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://svn1.cvsdude.com/osflash/red5/doc/tags/1_2_3
4. Tag the server code according to the naming scheme from above at http://svn1.cvsdude.com/osflash/red5/java/server/tags
5. Update the svn:externals in the newly created server code tag to point to the tagged modules from step 3.
6. You're done.

Part II. Red5 Core Technologies

Red5 core intro here

- Chapter 11, *Create new applications in Red5*
- Chapter 12, *Create new applications in Red5 WAR*
- Chapter 13, *Build Environment Setup*
- Chapter 14, *Security*
- Chapter 15, *Scripting Implementations*
- Chapter 16, *Clustering*
- Chapter 17, *Management*

11.1. Preface

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

11.2. 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.

11.3. Configuration

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

11.3.1. 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>
```

11.4. Handler configuration

Every handler configuration file must contain at least three beans:

11.4.1. 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.

11.4.2. 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">
```

```
    <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.

11.4.3. 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" />
```

The *id* attribute is referenced by the scope definition above.

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.MultiThreadedApplicationAdapter"
  singleton="true" />
```

11.5. Sample handler

A sample handler can be implemented in a few lines of code:

```
package the.path.to.my;

import org.red5.server.adapter.MultiThreadedApplicationAdapter;

public class Application extends MultiThreadedApplicationAdapter {

    public Double add(Double a, Double b){
        return a + b;
    }

}
```

Assuming the sample configuration above, you can call this method using the following ActionScript snippet:

```
nc = new NetConnection();
nc.connect("rtmp://localhost/myapp");
nc.onResult = function(obj) {
trace("The result is " + obj);
}
nc.call("add", nc, 1, 2);
```

This should give you the output:

```
The result is 3
```

WAR addendum

12.1. Preface

This document describes how applications can be configured in Red5 when using the WAR implementation. In this version of Red5 the J2EE container is not contained within Red5 and therefore is configured differently. This document assumes that the application WAR has already been expanded.

12.2. The application directory

An application war is normally expanded into a directory based upon the name of the war file, eg. red5.war expands into tomcat/webapps/red5 on a Tomcat server. In a standard Red5 installation, all the applications are stored within their own directory under the webapps directory; the difference here is that they are all located in the same directory.

12.3. Configuration

The WAR version stores all application definitions as Spring configuration files suffixed with the string "-context.xml"; If your application was called ofla then its configuration file would be named "ofla-context.xml". The context files are loaded automatically upon server startup.

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

12.3.1. globalScope

The name of the global scope, this should be left at the default:

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

12.3.2. contextConfigLocation

Specifies the name(s) of handler configuration files for this application. The handler configuration files reference the classes that are used to notify the application about joining / leaving clients and that provide the methods a client can call.

Additionally, the handler configuration files specify the scope hierarchy for these classes.

The path name given here can contain wildcards to load multiple files:

```
<context-param>
   <param-name>contextConfigLocation</param-name>
```

```
   <param-value>/WEB-INF/applicationContext.xml, /WEB-INF/red5-common.xml, /WEB-INF/red5-core.xml, /WE
</context-param>
```

12.3.3. listener (start-up / shutdown)

References the context listener servlet of the application, this technically takes the place of the Standalone.class in a standard Red5 server

```
<listener>
   <!-- Impersonates a org.springframework.web.context.ContextLoaderListener -->
   <listener-class>org.red5.server.MainServlet</listener-class>
</listener>
```

12.3.4. parentContextKey

Name of the parent context, this usually is "default.context":

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

12.3.5. log4jConfigLocation

Path to the configuration file for the logging subsystem:

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

12.4. Handler configuration

Every handler configuration file must contain at least three beans:

12.4.1. Context

The default 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 and they should follow this naming convention '<application name>.context' so that they will not conflict with one another. Application contexts can be shared across multiple scopes.

12.4.2. 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' and they should follow this naming convention '<application name>.scope' so that they will not conflict with one another.

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. In this version we do not control the host names, this is accomplished by the server.

A sample definition looks like this:

```
<bean id="ofla.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="ofla.context" />
   <property name="handler" ref="ofla.handler" />
   <property name="contextPath" value="/oflaDemo" />
   <property name="virtualHosts" value="localhost, 127.0.0.1" />
</bean>
```

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.

12.4.3. 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="ofla.handler" class="the.path.to.my.Application" singleton="true" />
```

The *id* attribute is referenced by the scope definition above.

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.MultiThreadedApplicationAdapter" singleton="true
```

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

13.1. Filename generator service

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

13.2. Custom generator

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

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

public class CustomFilenameGenerator implements IStreamFilenameGenerator {

/** Path that will store recorded videos. */
public String recordPath = "recordedStreams/";
/** Path that contains VOD streams. */
public String playbackPath = "videoStreams/";

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;
}
}
```

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

13.3. Activate custom generator

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

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

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

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

13.4. Change paths through configuration

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

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

Add two 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;
}
```

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/" />
</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}" />
```

```
</bean>
```

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

```
recordPath=recordedStreams/
playbackPath=videoStreams/
```

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

14.1. Streams

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

14.1.1. Stream playback security

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

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

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

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

           public class NamePlaybackSecurity implements IStreamPlaybackSecurity {

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

           }
```

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

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

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

14.1.2. Stream publishing security

In most applications that allow the user to publish and/or record streams, this access must be limited to prevent the server from being

misused. Therefore, Red5 provides the interface IStreamPublishSecurity [http://dl.fancycode.com/red5/api/org/red5/server/api/stream/IStreamPublishSecurity.html] to deny publishing of certain streams.

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

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

```
import org.red5.server.api.IConnection;
            import org.red5.server.api.IScope;
            import org.red5.server.api.Red5;
            import org.red5.server.api.stream.IStreamPublishSecurity;

            public class AuthNamePublishSecurity implements IStreamPublishSecurity {

            public isPublishAllowed(IScope scope, String name, String mode) {
            if (!"live".equals(mode)) {
            // Not a live stream
            return false;
            }

            IConnection conn = Red5.getConnectionLocal();
            if (!"authenticated".equals(conn.getAttribute("UserType"))) {
            // User was not authenticated
            return false;
            }

            if (!name.startswith("liveStream")) {
            return false;
            } else {
            return true;
            }
            };

            }
```

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

```
registerStreamPublishSecurity(new AuthNamePublishSecurity());
```

Of course, you will also have to add code in one of the **Connect* or **Join* methods that set the *UserType* attribute of a connection to *authenticated* for users that are allowed to publish streams.

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

14.2. Shared objects

Once applications get complex, you might want to control the data that is stored in a shared object, thus not allowing the clients to modify SOs directly but only through methods exposed by the application.

The interface ISharedObjectSecurity [http://dl.fancycode.com/red5/api/org/red5/server/api/so/ISharedObjectSecurity.html] can be used to write handlers that deny certain actions on a given shared object or prevent the client from creating arbitrary shared objects.

Below is an example handler that only allows the creation of the persistent shared object *Chat*. Any client may connect to it and only sending messages *saySomething* through the SO is allowed. All write access to properties is denied. You could however change properties through serverside code as these changes are never protected by the security handlers.

```
import java.util.List;
import org.red5.server.api.IScope;
import org.red5.server.api.so.ISharedObject;
import org.red5.server.api.so.ISharedObjectSecurity;

public class SampleSOSecurityHandler implements ISharedObjectSecurity {

public boolean isConnectionAllowed(ISharedObject so) {
// Note: we don't check for the name here as only one SO can be
//       created with this handler
.
return true;
}

public boolean isCreationAllowed(IScope scope, String name,
boolean persistent) {
if (!"Chat".equals(name) || !persistent) {
return false;
} else {
return true;
}
}

public boolean isDeleteAllowed(ISharedObject so, String key) {
return false;
}

public boolean isSendAllowed(ISharedObject so, String message,
List arguments) {
if (!"saySomething".equals(message)) {
return false;
} else {
return true;
}
}

public boolean isWriteAllowed(ISharedObject so, String key,
Object value) {
return false;
}

}
```

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

```
registerSharedObjectSecurity(new SampleSOSecurityHandler());
```

If you want to register a security handler only for a given shared object, use code like this:

```
ISharedObject so = getSharedObject(scope, "MySharedObject");
so.registerSharedObjectSecurity(new MySOSecurityHandler());
```

15.1. I. Select a scripting implementation

Level: Beginner

Red5 includes interpreters for the following scripting languages:

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

Future versions may include:

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

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

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

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

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

15.2. II. Configuring Spring

Level: Intermediate

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

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

- Java class implementation

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

- Javascript implementation

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

- Ruby implementation

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

- Groovy implementation

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

- Python implementation

```
<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>
```

```
  </list>
 </constructor-arg>
        <constructor-arg index="2">
  <list>
            <value>One</value>
            <value>2</value>
            <value>III</value>
        </list>
    </constructor-arg>
</bean>
```

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

Argument 1 - Location of the script source file

Argument 2 - Java interfaces implemented by the script

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

Argument 3 - Java classes extended by the script

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

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

15.3. III. Creating an application script

Level: Intermediate

15.4. 1. Application adapter

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

i. JRuby application adapter implementation

```
# JRuby
require 'java'
module RedFive
    include_package "org.red5.server.api"
    include_package "org.red5.server.api.stream"
    include_package "org.red5.server.api.stream.support"
    include_package "org.red5.server.adapter"
    include_package "org.red5.server.stream"
end

#
# application.rb - a translation into Ruby of the ofla demo application, a red5 example.
#
# @author Paul Gregoire
#
```

```
class Application &lt; RedFive::ApplicationAdapter

    attr_reader :appScope, :serverStream
        attr_writer :appScope, :serverStream

        def initialize
           #call super to init the superclass, in this case a Java class
           super
           puts "Initializing ruby application"
        end

        def appStart(app)
        puts "Ruby appStart"
                @appScope = app
                return true
        end

        def appConnect(conn, params)
                puts "Ruby appConnect"
                measureBandwidth(conn)
                puts "Ruby appConnect 2"
                if conn.instance_of?(RedFive::IStreamCapableConnection)
                    puts "Got stream capable connection"
                        sbc = RedFive::SimpleBandwidthConfigure.new
                        sbc.setMaxBurst(8388608)
                        sbc.setBurst(8388608)
                        sbc.setOverallBandwidth(8388608)
                        conn.setBandwidthConfigure(sbc)
                end
                return super
        end

        def appDisconnect(conn)
                puts "Ruby appDisconnect"
                if appScope == conn.getScope &amp;&amp; @serverStream != nil
                        @serverStream.close
                end
                super
        end

        def toString
                return "Ruby toString"
        end

    def setScriptContext(scriptContext)
           puts "Ruby application setScriptContext"
    end

    def method_missing(m, *args)
      super unless @value.respond_to?(m)
      return @value.send(m, *args)
    end

end
```

15.5. 2. Application services

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

i. Simple Java interface for implementation by scripts

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

import java.util.Map;

public interface IDemoService {

        /**
     * Getter for property 'listOfAvailableFLVs'.
     *
     * @return Value for property 'listOfAvailableFLVs'.
     */
    public Map getListOfAvailableFLVs();

    public Map getListOfAvailableFLVs(String string);

}
```

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

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

iii. JRuby script implementing the interface

```
# JRuby - style
require 'java'
module RedFive
    include_package "org.springframework.core.io"
    include_package "org.red5.server.webapp.oflaDemo"
end
include_class "org.red5.server.api.Red5"
include_class "java.util.HashMap"

#
# demoservice.rb - a translation into Ruby of the ofla demo application, a red5 example.
#
# @author Paul Gregoire
#
class DemoService &lt; RedFive::DemoServiceImpl

    attr_reader :filesMap
    attr_writer :filesMap

        def initialize
           puts "Initializing ruby demoservice"
           super
           @filesMap = HashMap.new
        end
```

```
        def getListOfAvailableFLVs
                puts "Getting the FLV files"
                begin
            dirname = File.expand_path('webapps/oflaDemo/streams').to_s
                        Dir.open(dirname).entries.grep(/\.flv$/) do |dir|
                            dir.each do |flvName|
                            fileInfo = HashMap.new
                            stats = File.stat(dirname+'/'+flvName)
                            fileInfo["name"] = flvName
                            fileInfo["lastModified"] = stats.mtime
                            fileInfo["size"] = stats.size || 0
                        @filesMap[flvName] = fileInfo
                        print 'FLV Name:', flvName
                        print 'Last modified date:', stats.mtime
                        print 'Size:', stats.size || 0
                        print '-------'
                    end
                end
                    rescue Exception =&gt; ex
                            puts "Error in getListOfAvailableFLVs #{errorType} \n"
                            puts "Exception: #{ex} \n"
                            puts caller.join("\n");
                    end
                    return filesMap
        end

        def formatDate(date)
                return date.strftime("%d/%m/%Y %I:%M:%S")
        end

    def method_missing(m, *args)
      super unless @value.respond_to?(m)
      return @value.send(m, *args)
    end

end
```

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

```


package org.red5.server.webapp.oflaDemo;

import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.red5.server.api.IScope;
import org.red5.server.api.Red5;
import org.springframework.core.io.Resource;

public class DemoService {

        protected static Log log = LogFactory.getLog(DemoService.class.getName());

        /**
     * Getter for property 'listOfAvailableFLVs'.
```

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

                        Resource[] mp3s = scope.getResources("streams/*.mp3");
                        if (mp3s != null) {
                                for (Resource mp3 : mp3s) {
                                        File file = mp3.getFile();
                                        Date lastModifiedDate = new Date(file.lastModified());
                                        String lastModified = formatDate(lastModifiedDate);
                                        String flvName = mp3.getFile().getName();
                                        String flvBytes = Long.toString(file.length());
                                        if (log.isDebugEnabled()) {
                                                log.debug("flvName: " + flvName);
                                                log.debug("lastModified date: " + lastModified);
                                                log.debug("flvBytes: " + flvBytes);
                                                log.debug("-------");
                                        }
                                        fileInfo = new HashMap&lt;String, Object&gt;();
                                        fileInfo.put("name", flvName);
                                        fileInfo.put("lastModified", lastModified);
                                        fileInfo.put("size", flvBytes);
                                        filesMap.put(flvName, fileInfo);
                                }
                        }
                } catch (IOException e) {
                        log.error(e);
                }
                return filesMap;
        }

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

}
```

v. Flex AS3 method calling the service

```
[Bindable]
public var videoList:ArrayCollection;

public function catchVideos():void{
        // call server-side method
        // create a responder and set it to getMediaList
        var nc_responder:Responder = new Responder(getMediaList, null);
        // call the server side method to get list of FLV's
        nc.call("demoService.getListOfAvailableFLVs", nc_responder);
}

public function getMediaList(list:Object):void{
        // this is the result of the server side getListOfAvailableFLVs
        var mediaList:Array = new Array();
        for(var items:String in list){
            mediaList.push({label:items, size:list[items].size, dateModified:list[items].lastModifi
        }
        // videoList is bindable and the datagrid is set to use this for it's dataprovider
        // wrap it in an ArrayCollection first
        videoList = new ArrayCollection(mediaList);
}
```

15.6. 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.

15.7. V. Links with scripting information

- Spring scripting

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

- Java scripting

```
http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/

http://blogs.sun.com/sundararajan/
```

https://scripting.dev.java.net/

http://today.java.net/pub/a/today/2006/04/11/scripting-for-java-platform.html

http://www.javaworld.com/javaworld/jw-03-2005/jw-0314-scripting_p.html

http://www.oreillynet.com/onjava/blog/2004/01/java_scripting_half_the_size_h.html

http://www.robert-tolksdorf.de/vmlanguages.html

- Javascript

  http://www.mozilla.org/rhino/

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

- Ruby

  http://jruby.codehaus.org/

- BeanShell

  http://www.beanshell.org/

- Python

  http://www.jython.org/Project/

  http://www.onjava.com/pub/a/onjava/2002/03/27/jython.html

  http://jepp.sourceforge.net/

  http://jpe.sourceforge.net/

  http://jpype.sourceforge.net/

- Python

  http://groovy.codehaus.org/

16.1. Server Configuration

There are several configuration files added to support Edge/Origin configuration, they are under conf/:

- red5-edge.xml, red5-edge-core.xml -- used for Edge Spring bean configuration.
- red5-origin.xml, red5-origin-core.xml -- used for Origin Spring bean configuration

16.2. Configure Edge Server

You don't need to deploy your application on Edges. We strongly recommend you to deploy Edge on a different server from Origin. But it should be OK to deploy the Edge on the same server as Origin.

16.3. Edge on a different Server from Origin

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

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

Now you can start the server with red5.sh or 'java -jar red5.jar'.

16.4. Edge on the same Server as Origin

You don't need to change red5.xml. Copy $(RED5_EDGE_ROOT)/conf/red5.xml to $(RED5_ORIGIN_ROOT)/red5-edge.xml and start the server with 'java -jar red5.jar red5-edge.xml' or update red5.sh and add the 'red5-edge.xml' parameter.

16.5. Configure Origin Server

Deploy your application to webapps/. Make sure your 9035 port is not blocked by a firewall. The port will be used by Edge servers to create a connection with the Origin server. Start the server with red5.sh or 'java -jar red5.jar'.

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

I. 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.

II. 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 -->
<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="9999"/>
 <!-- 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="false"/>
 <!-- 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"/>
 <!-- The HTML adapter allows connections to the MBeanServer via a web browser -->
 <property name="enableHtmlAdapter" value="false" />
 <property name="htmlAdapterPort" value="8082"/>
</bean>
```

The HTML adapter is disabled by default, but it allows easy management of MBeans from a web browser. The RMI adapter may only be used if an RMI registry is running. To start a registry simply execute this at the command prompt:

- windows

```
rmiregistry 9999
```

- unix

```
rmiregistry 9999 &
```

The "9999" is the port and should match your RMI configuration.

III. jConsole

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

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".

IV. 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

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 emtpy replies. The maximum delay is 0x21 which causes a delay of approximately 0.5 seconds between two requests.

Red5 currently increases the delay in the following steps: 0x01, 0x03, 0x05, 0x09, 0x11, 0x21.

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.getConnectParams"
- "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) - RTMPTHandler 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 im 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 connetions (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