

Red5 Flash Server

Reference Documentation

Dan Rossi

Red5 Flash Server: Reference Documentation

by Dan Rossi

1.0.0

Table of Contents

Preface	vii
I. Red5 Architecture	8
1. Introduction	9
What is Red5?	9
Where can I download Red5?	9
Why does Red5 exist?	9
What does Red5 mean?	9
Why is it not called Red5 Media Server?	9
What other names does Red5 go by?	9
What does the Red5 logo look like?	9
Is Red5 Legal ?	9
What is the outcome of this project ?	10
What license does Red5 use?	10
What is the estimated time of delivery for Red5 ?	10
RTMP	10
AMF	10
What implementations are there ?	10
How mature is the Java implementation?	10
Why did the Red5 team choose Mina?	10
Why did the Red5 team choose Spring?	11
Which component frameworks are we currently reviewing?	11
What are the advantages of this framework?	11
What is the estimated time of delivery for these components (eta)?	11
Do I have to use this component framework?	11
Can I use Macromedia Flash Communication Server components?	11
What IDE (integrated development environment) are we using?	11
Is there a place that we can meet and talk about the project status?	11
Is there a mailing list ?	12
Where's the best place to start?	12
What are the milestones ?	12
Who are the team members?	12
2. Migration guide from FCS/FMS to Red5	14
Application callbacks	14
Interface IScopeHandler	14
Class ApplicationAdapter	14
Execution order of connection methods	14
Accepting / rejecting clients	15
Current connection and client	15
Additional handlers	16
Handlers in configuration files	16
Handlers from application code	17
Calls to client methods	17
SharedObjects	18
Serverside change listeners	19
Changing from application code	19
SharedObject event handlers	20
Persistence	20
Periodic events	21
Remoting	22
Remoting server	22
Remoting client	23
Streams	24
II. Configuration files used by Red5	25
3. Directory "conf"	26
jetty.xml	26
keystore	26
log4j.properties	26

realm.properties (Jetty)	26
tomcat-users.xml (Tomcat)	26
red5.globals	27
red5.properties	27
red5.xml	27
red5-common.xml	27
red5-core.xml	28
red5-rtmpt.xml	29
web.xml (Tomcat)	29
web-default.xml (Jetty)	29
4. Webapp config directory	30
red5-web.xml	30
III. Building Red5	31
5. Building with Ant	32
6. Building a Red5 Debian package	33
7. Release	34
IV. Applications	35
8. Create new applications in Red5	36
The application directory	36
Configuration	36
globalScope	36
contextConfigLocation	36
locatorFactorySelector	37
parentContextKey	37
log4jConfigLocation	37
webAppRootKey	37
Handler configuration	38
Context	38
Scopes	38
Handlers	39
Sample handler	40
9. Setup applications in Red5 WAR	41
The application directory	41
Configuration	41
globalScope	41
contextConfigLocation	41
listener (start-up / shutdown)	42
parentContextKey	42
log4jConfigLocation	42
Handler configuration	42
Context	42
Scopes	43
Handlers	44
V. Management	45
10. Java Management Extensions (JMX) and Red5	46
JMX classes	46
Spring configuration	46
jConsole	46
Links	46
VI. Scripting	48
11. Select a scripting implementation	49
12. Configuring Spring	50
13. Creating an application script	53
Application adapter	53
JRuby application adapter implementation	53
Application services	54
Simple Java interface for implementation by scripts	54
Spring bean definition for a script implementation of the interface	55
JRuby script implementing the interface	55
Java application implementing the interface	56
Flex AS3 method calling the service	57
Creating your own interpreter	58
Links with scripting information	58

List of Tables

2.1.	14
-----------	----

Preface

Overall Preface here

Part I. Red5 Architecture

Part intro

Chapter 1. Introduction

Dominick Accattato <(daccattato at gmail dot com)>

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).

Where can I download Red5?

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

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.

What does Red5 mean?

Please read: Why "RED5"? [http://osflash.org/red5#why_red5]

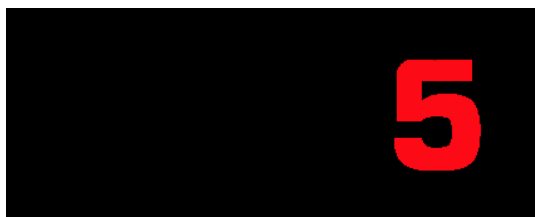
Why is it not called Red5 Media Server?

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

What other names does Red5 go by?

Red5 aka R5

What does the Red5 logo look like?



Is Red5 Legal ?

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.

What is the outcome of this project ?

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.

What license does Red5 use?

Red5 uses the LGPL license. LGPL license [<http://www.opensource.org/licenses/lgpl-license.php>]

What is the estimated time of delivery for Red5 ?

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

RTMP

Real Time Messaging Protocol [http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol] (RTMP) Is a proprietary protocol developed by Macromedia that is primarily used with Macromedia's Media Server product to stream audio and video over the web. The default connection port is 1935.

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

AMF

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

What implementations are there ?

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

How mature 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 rtmp protocol.

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 strategies 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.

Why did the Red5 team choose Spring?

TODO

Which component frameworks are we currently reviewing?

Actionstep, ASwing.

What are the advantages of this framework?

TODO

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

TODO

Do I have to use this component framework?

TODO

Can I use Macromedia Flash Communication Server components?

TODO

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.

Recommend IDE / Software

- Eclipse IDE [<http://www.eclipse.org/downloads>]
- Spring IDE Eclipse Plugin [<http://springide.org/project>]
- ASDT Eclipse Actionscript Plugin [<http://aseclipseplugin.sourceforge.net/wordpress>]

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

Is there a mailing list ?

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

Where's the best place to start?

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.

What are the milestones ?

TODO

Who are the team members?

Project Management

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

Server Side Development

- Mick Merres [http://osflash.org/mick_merres]
- Luke Hubbard [http://osflash.org/luke_hubbard]
- Grant Davies [http://osflash.org/grant_davies]
- Fernando Augusto [http://osflash.org/fernando_augusto]
- Lucas Ferreira [http://osflash.org/lucas_ferreira]
- Chris Allen [http://osflash.org/chris_allen]
- Ashvin Savani [http://osflash.org/ashvin_savani]
- Yannick Connan [http://osflash.org/yannick_connan]
- Dominick Accattato [http://osflash.org/dominick_accattato]

Release Manager

- Grant Davies [http://osflash.org/grant_davies]

Codecs/Media integration

- Dominick Accattato [http://osflash.org/dominick_accattato]

Client Side/API Testing

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

- Marcos Augusto [http://osflash.org/marcos_augusto]
- Gabriel Laet [http://osflash.org/gabriel_laet]
- Marlos Carmo [http://osflash.org/marlos_carmo]
- Tim Beynart [http://osflash.org/tim_beynart]
- Lucas Ferreira [http://osflash.org/lucas_ferreira]
- Chris Allen [http://osflash.org/chris_allen]
- Ashvin Savani [http://osflash.org/ashvin_savani]

Branding/Logo/Website

- Tim Beynart [http://osflash.org/tim_beynart]
- Aldo Bucchi [http://osflash.org/aldo_bucchi]

Documentation

- Patrick Mineault (AMF documentation) [http://osflash.org/patrick_mineault]
- John Grden [http://osflash.org/john_grden]
- Lee McColl-Sylvester [http://osflash.org/lee_mccoll-sylvester]

Chapter 2. Migration guide from FCS/FMS to Red5

Joachim Bauch <jojo@struktur.de>

This document describes API differences between the Macromedia Flash Communication Server / Flash Media Server 2 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.

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.

Interface IScopeHandler

Red5 specifies these actions in the interface `IScopeHandler_`. See the API documentation for further details.

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_` 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_` in Red5:

Table 2.1.

FCS / FMS	Red5
<code>onAppStart</code>	<code>appStart roomStart</code>
<code>onAppStop</code>	<code>appStop roomStop</code>
<code>onConnect</code>	<code>appConnect roomConnect appJoin roomJoin</code>
<code>onDisconnect</code>	<code>appDisconnect roomDisconnect appLeave roomLeave</code>

The `app*`` methods are called for the main application, the `room*`` methods are called for rooms (i.e. instances) of the application. You can also use the `ApplicationAdapter_` to check for streams, shared objects, or subscribe them. See the API documentation for further details.

Execution order of connection methods

Assuming you connect to `rtmp://server/app/room1/room2`` At first, the connection is established, so the user "connects" to all scopes that are traversed up to `room2``:

1. `app`` (-> `appConnect`)
2. `room1`` (-> `roomConnect`)

3. ``room2` (-> roomConnect)`

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

1. ``app` (-> appJoin)`

2. ``room1` (-> roomJoin)`

3. ``room2` (-> roomJoin)`

If the same client establishes a second connection to the same scope, only the ``connect`` methods will be called. If you connect to partially the same scopes, only a few ``join`` methods might be called, e.g. ``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.

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_` class. Any parameter passed to ``rejectClient`` is available as the ``application`` property of the status object that is returned to the caller.

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_` 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.getConnection();
    IClient client = conn.getClient();
    IScope scope = conn.getScope();
    // ...
}
```

The second possibility requires the method to be defined with an argument of type `IConnection_` 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();
    // ...
}
```

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/");
nc.call("handler.method", nc,
```

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.

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

Note

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

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_` which is implemented by `ApplicationAdapter_`.

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

```
public boolean appStart(IScope scope) {
    if (!super.appStart(scope))
        return false;

    Object handler = new MyHandler();
    app.registerServiceHandler(handler, "sample", "service");
    return true;
}
```

Note

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

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.IService;
...
IConnection conn = Red5.getConnectionLocal();
```

If the connection implements the `IServiceCapableConnection_` interface, it supports calling methods on the other end:

```
if (conn instanceof IServiceCapableConnection) {
    ((IServiceCapableConnection) conn).invoke("the_method", args);
}
```

If you need the result of the method call, you must provide a class that implements the `IPendingServiceCallback_` interface:

```
import org.red5.server.api.se
import org.red5.server.api.se

class MyCallback implements I
    public void resultRec
        // Do somethi
    }
}
```

The method call looks now like this:

```
if (conn instanceof IServiceC
    IServiceCapableConnec
    sc.invoke("the_method
}
```

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

SharedObjects

The methods to access shared objects from an application are specified in the interface `ISharedObjectService_`. 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.adapte
import org.red5.server.api.IS
import org.red5.server.api.so

public class SampleApplication
    public boolean roomSt
        if (!super.ro
            return false;

        createSharedO
        ISharedObject

        // Now you co

        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_`.

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_`:

```
import org.red5.server.api.so.  
import org.red5.server.api.so.  
  
public class SampleSharedObject  
    implements ISharedObjectListener_  
  
    public void onSharedObjectChanged(  
        String key, Object value)  
        // The attribute  
        // was changed  
    }  
  
    public void onSharedObjectCreated(  
        Object value)  
        // The attribute  
        // was created  
    }  
  
    public void onSharedObjectDeleted(  
        String method)  
        // The handle  
        // with the p  
    }  
  
    // Other methods as d  
}
```

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

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

Changing from application code

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

```
ISharedObject so = getSharedObject("so");  
so.setAttribute("fullname", "John Doe");
```

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("so");  
so.beginUpdate();  
so.setAttribute("fullname", "John Doe");  
so.endUpdate();
```

```
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``.

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_` interface similar to the application handlers:

```
package com.fancycode.red5;

class MySharedObjectHandler
{
    public void myMethod(
        // Now do something
    )
}

...
ISharedObject so = getSharedObject();
so.registerServiceHandler(new MySharedObjectHandler());
```

Handlers with a given name can be registered as well:

```
ISharedObject so = getSharedObject();
so.registerServiceHandler("one.two.myMethod", 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="sampleS0.one.two.soservice"
      class="com.fancycode.red5.MySharedObjectHandler"
      singleton="true" />
```

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_`. Basically these objects have a ``type``, a ``path``, a ``name`` (all strings) and know how to serialize and deserialize them-

selves. Here is a sample of serialization and deserialization:

```
import java.io.IOException;
import org.red5.io.object.InputObject;
import org.red5.io.object.OutputObject;
import org.red5.server.api.IPersistentObject;

class MyPersistentObject implements IPersistentObject {

    // Attribute that will store the data
    private String data = null;

    void serialize(OutputObject output) {
        // Save the object to the output
        output.writeS...
    }

    void deserialize(InputObject input) {
        // Load the object from the input
        data = input...
    }

    // Other methods as defined by the interface
}
```

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

```
import org.red5.server.adapter.IAdapter;
import org.red5.server.api.IScope;
import org.red5.server.api.IAttributeStore;
import org.red5.server.api.IPersistentObject;

class MyApplication extends IAdapter {

    private void saveObject(IScope scope, IPersistentObject obj) {
        // Get current attribute store
        IAttributeStore store = scope.getAttributeStore();
        // Save object to the store
        store.put(obj.getName(), obj);
    }

    private void loadObject(IScope scope, String name) {
        // Get current attribute store
        IAttributeStore store = scope.getAttributeStore();
        // Load object from the store
        IPersistentObject obj = store.get(name);
    }

}
```

If no custom objects are required for an application, but data must be stored for future reuse, it can be added to the `IScope` through the interface `IAttributeStore`. 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.

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`) that is implemented by `ApplicationAdapter` like most other services. The service can register an object (which needs to implement the `IScheduledJob` 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.ISchedulingService;
import org.red5.server.api.IScheduledJob;
import org.red5.server.api.IScope;
import org.red5.server.adapter.ApplicationAdapter;

class MyJob implements IScheduledJob {
    public void execute(IScope scope) {
        // Do something
    }
}

public class SampleApplicationAdapter extends ApplicationAdapter {
    public boolean roomStop(IScope scope, String id) {
        if (!super.roomStop(scope, id)) {
            return false;
        }

        // Schedule job
        String jobId = "roomStop-" + id;
        room.setAttri(jobId, new MyJob());
        return true;
    }
}
```

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

```
public void roomStop(IScope scope, String id) {
    String jobId = (String) scope.getAttri(id);
    removeScheduledJob(jobId);
    super.roomStop(scope, id);
}
```

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.

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-class>
        org.red5.server
    </servlet-class>
</servlet>

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

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/g
```

Remoting client

The class `RemotingClient_` 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.re

String url = "http://server/p
RemotingClient client = new R
Object[] args = new Object[]{
Object result = client.invoke
// Now do something with the
```

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_` must be passed as additional parameter:

```
import org.red5.server.net.re
import org.red5.server.net.re

public class CallbackHandler {

    void errorReceived(RemotingClient client, Object[] parameters) {
        // An error occurred
    }

    void resultReceived(RemotingClient client, Object[] parameters, Object result) {
        // The result
    }
}
```

```
String url = "http://server/p
RemotingClient client = new R
Object[] args = new Object[] {
    IRemotingCallback callback =
client.invokeMethod("service.
```

Streams

TODO: How can streams be accessed from an application?

Part II. Configuration files used by Red5

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!

Chapter 3. Directory "conf"

Joachim Bauch <jojo@struktur.de>

Paul Gregoire <mondain@gmail.com>

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`__ for further information about the syntax of this file. <http://jetty.mortbay.org/jetty6/>

keystore

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

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/>

realm.properties (Jetty)

This file defines users passwords and roles that can be used for protected areas. The format is:

`<username>: <password>`

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

tomcat-users.xml (Tomcat)

This file defines users passwords and roles that can be used for protected areas. The format is:

`<user name="<username>`

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>

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".

red5.properties

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

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.serv
```

- Tomcat

```
<bean id="tomcat.serv  
... cut for b  
</bean>
```

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.cach
```

The other bean configurations are as follows (Only one may be used at a time): - Red5 homegrown simple example

```
<bean id="object.cac
```

```
<property name=  
</bean>
```

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

```
<bean id="object.cach  
    <property nam  
    <property nam  
    <property nam  
    <property nam  
    <list>
```

```
    </lis  
    </property>  
</bean>
```

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

```
<bean id="object.cach  
    <property nam  
    <property nam  
    <bean
```

```
    </bea  
    </property>  
</bean>
```

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.serve
```

- Tomcat

```
<bean id="rtmpt.serve  
... cut for b  
</bean>
```

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` for further information about the syntax of this file. <http://jetty.mortbay.org/jetty6/>

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>

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.

Chapter 4. Webapp config directory

Joachim Bauch <jojo@struktur.de>

Paul Gregoire <mondain@gmail.com>

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.WebAppHandler">
```

- Javascript / Rhino application

```
<bean id="web.handler" class="org.red5.server.webapp.WebAppHandler">
    <constructor-arg>
        <!-- Implementer of the WebAppHandler interface -->
        <constructor-arg>
            <list>
                <value>org.red5.server.webapp.WebAppHandler$JavaScriptHandler
            </list>
        </constructor-arg>
        <!-- Extended constructor arguments -->
        <constructor-arg>
            <value>
                <!-- Extended constructor arguments -->
            </value>
        </constructor-arg>
    </bean>
```

- Ruby application

```
<bean id="web.handler" class="org.red5.server.webapp.WebAppHandler">
    <constructor-arg>
        <!-- Implementer of the WebAppHandler interface -->
        <constructor-arg>
            <list>
                <value>org.red5.server.webapp.WebAppHandler$RubyHandler
            </list>
        </constructor-arg>
    </bean>
```

Part III. Building Red5

Part intro

Chapter 5. Building with Ant

Dominick Accattato <(daccattato at gmail dot com)>

TODO

Chapter 6. Building a Red5 Debian package

Dominick Accattato <(daccattato at gmail dot com)>

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.

Chapter 7. Release

Dominick Accattato <(daccattato at gmail dot com)>

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 IV. Applications

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

Chapter 8. Create new applications in Red5

Paul Gregoire <>

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

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.

Configuration

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

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>
```

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/red5-*.xml</param-value>
```

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

locatorFactorySelector

References the configuration file of the root application context which usually is "red5.xml":

```
<context-param>
  <param-name>locatorFactorySelector</param-name>
  <param-value>red5.xml</param-value>
</context-param>
```

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>
```

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>
```

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>
```

Handler configuration

Every handler configuration file must contain at least three beans:

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.

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.

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

Sample handler

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

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

import org.red5.server.adapter.ApplicationAdapter;

public class Application extends ApplicationAdapter {

    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

Chapter 9. Setup applications in Red5 WAR

Paul Gregoire <mondain@gmail.com>

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.

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.

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:

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>
```

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-com
```

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

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.ContextLoad
  <listener-class>org.red5.server.MainServlet</listener-class>
</listener>
```

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>
```

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>
```

Handler configuration

Every handler configuration file must contain at least three beans:

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 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.

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" />
    />
```

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.

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.ApplicationAdapter" si
```

Part V. Management

Part intro

Chapter 10. Java Management Extensions (JMX) and Red5

Paul Gregoire <>

TODO

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 Object-Name or MBean instances.

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.jmx" />
</bean>
<bean id="jmxAgent" class="org.red5.server.jmx.JMXAgent"
    <!-- The RMI adapter is disabled by default
    <property name="rmiPort" value="9999" />
    <property name="htmlPort" value="9999" />
    <!-- The HTML adapter is disabled by default
    <property name="htmlPort" value="9999" />
    <property name="domain" value="org.red5.server.jmx" />
</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.

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".

Links

<http://www.onjava.com/pub/a/onjava/2004/09/29/tigerjmx.html?page=1>
<tp://java.sun.com/developer/JDCTechTips/2005/tt0315.html#2>
[tp://www.onjava.com/pub/a/onjava/2004/09/29/tigerjmx.html?page=1\]](tp://www.onjava.com/pub/a/onjava/2004/09/29/tigerjmx.html?page=1)

ht-
[ht-

Part VI. Scripting

TODO

Chapter 11. Select a scripting implementation

Paul Gregoire <>

Revision History
Revision 2

2 Feb 2007

Note

Level: Beginner

Red5 includes interpreters for the following scripting languages:

- Javascript - version 1.6 (Mozilla Rhino version 1.6.2)
- JRuby - version 0.9.2 (Ruby version 1.8.5)
- Jython - version 2.1 (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 (Not SSAS)

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

Java5 jsr-223-1.0-pr.jar /META-INF/services/javax.script.ScriptEngineFactory

Java6 resources.jar /META-INF/services/javax.script.ScriptEngineFactory

Red5 red5.jar /META-INF/services/javax.script.ScriptEngineFactory

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

Chapter 12. Configuring Spring

Paul Gregoire <>

Revision History
Revision 2

2 Feb 2007

Note

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

```
<bean id="web.handler" class="org.red5.server
```

Javascript

```
<bean id="web.handler" class="org.red5.server.script.rhino.RhinoScript"
  <constructor-arg index="0" value="classpath:applications/main" />
  <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

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

```
</bean>
```

Groovy

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

Python

```
<bean id="web.handler" class="org.red5.server.script.jython.JythonScript"
    <constructor-arg index="0" value="classpath:applications/main/web.xml" />
    <constructor-arg index="1">
        <list>
            <value>org.red5.server.api.IScopeHandler</value>
            <value>org.red5.server.adapter.IApplicationAdapter</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 | <p>Java interfaces implemented by the script</p> <p>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.</p> |
| Argument 3 | <p>Java classes extended by the script</p> <p>The extended class is not always necessary, it depends upon the scripting engine implementation.</p> |

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

Chapter 13. Creating an application script

Paul Gregoire <>

Revision History
Revision 2

2 Feb 2007

Note

Level: Intermediate

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.

JRuby application adapter implementation

```
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
#
# @author Paul Gregoire
#
class Application < RedFive::ApplicationAdapter
  attr_reader :appScope, :serverStream
  attr_writer :appScope, :serverStream

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

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

  def appConnect(conn, params)
    puts "Ruby appConnect"
    measureBandwidth(conn)
    puts "Ruby appConnect 2"
    if conn.instance_of?(RedFive::IStreamCapableConnection)
```

```

        puts "Got stream capable connection"
        sbc = RedFive::SimpleBandwidthConfigure.new
        sbc.setMaxBurst(8388608)
        sbc.setBurst(8388608)
        sbc.setOverallBandwidth(8388608)
        conn.setBandwidthConfigure(sbc)
    end
    return super
end

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

def toString
    return "Ruby toString"
end

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

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

```

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.

Simple Java interface for implementation by scripts

```

package org.red5.server

import java.util.Map;

public interface IDemoService {

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

    public Map getListOfAvailableFLVs(String string);

}

```

Spring bean definition for a script implementation of the interface

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

JRuby script implementing the interface

```
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
#
# @author Paul Gregoire
#
class DemoService < RedFive::DemoServiceImpl

  attr_reader :filesMap
  attr_writer :filesMap

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

  def getListOfAvailableFLVs
    puts "Getting the FLV files"
    begin
      dirname = File.expand_path('webapps/oflaDemo/streams').to_s
      Dir.open(dirname).entries.grep(/\.flv$/) do |dir|
        dir.each do |flvName|
          fileInfo = HashMap.new
          stats = File.stat(dirname+'/'+flvName)
          fileInfo["name"] = flvName
          fileInfo["lastModified"] = stats.mtime
          fileInfo["size"] = stats.size || 0
          @filesMap[flvName] = fileInfo
          print 'FLV Name:', flvName
        end
      end
    end
  end
end
```

```

        print 'Last modified date:', stats.mtime
        print 'Size:', stats.size || 0
        print '-----'
      end
    end
  rescue Exception => ex
    puts "Error in getListOfAvailableFLVs #{errorType} \n"
    puts "Exception: #{ex} \n"
    puts caller.join("\n");
  end
  return filesMap
end

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

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

```

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;

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<String, Map> filesMap = new HashMap<String, Map>();
        Map<String, Object> fileInfo;
        try {
            log.debug("getting the FLV files");
            Resource[] flvs = scope.getResources("streams/*.flv");
            if (flvs != null) {
                for (Resource flv : flvs) {

```

```

        File file = flv.getFile();
        Date lastModifiedDate = new Date(file.lastModified());
        String lastModified = formatDate(lastModifiedDate);
        String flvName = flv.getFile().getName();
        String flvBytes = Long.toString(file.length());
        if (log.isDebugEnabled()) {
            log.debug("flvName: " + flvName);
            log.debug("lastModified date: " + lastModified);
            log.debug("flvBytes: " + flvBytes);
            log.debug("-----");
        }
        fileInfo = new HashMap<String, Object>();
        fileInfo.put("name", flvName);
        fileInfo.put("lastModified", lastModified);
        fileInfo.put("size", flvBytes);
        filesMap.put(flvName, fileInfo);
    }
}

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

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

```

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, data:items});
    }
    // videoList is bindable and the datagrid is set to use this for data
    // wrap it in an ArrayCollection first
    videoList = new ArrayCollection(mediaList);
}

```

Creating your own interpreter

Note

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.

Links with scripting information

Spring scripting	http://static.springframework.org/spring/docs/2.0.x/reference/dynamic-language.html http://rhinoin.spring.sourceforge.net/
Java scripting	http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/ http://blogs.sun.com/sundararajan/ https://scripting.dev.java.net/ https://scripting.dev.java.net/ http://today.java.net/pub/a/today/2006/04/11/scripting-for-java-platform.html http://www.javaworld.com/javaworld/jw-03-2005/jw-0314-scripting_p.html http://www.oreillynet.com/onjava/blog/2004/01/java_scripting_half_the_size_h.html

	http://www.robert-tolksdorf.de/vmlanguages.html
Javascript	http://www.mozilla.org/rhino/ http://www.mozilla.org/rhino/ScriptingJava.html
Ruby	http://jruby.codehaus.org/
Beanshell	http://www.beanshell.org/
Python	http://www.jython.org/Project/ http://www.onjava.com/pub/a/onjava/2002/03/27/jython.html http://jepp.sourceforge.net/ http://jpe.sourceforge.net/ http://jpype.sourceforge.net/
Groovy	http://groovy.codehaus.org/