

Tutorials:Getting Started With Red5 Server

By Milan Toth

Milan Toth is the Chief Flash Developer of Jasmin Media Group, he created one of the world's biggest flash media server system, check <http://livejasmin.com> and subbrands. He loves Eclipse and OS X, AS3 and JAVA, sci-fi and horror, metal and electronic.

[edit]

Part One - Setting up environment

Although we are on Actionsript.org, this article is mainly about java, ha-ha :D But do not fear my friend, you will need red5 sooner or later for your flash project. I created this tutorial on OS X, so there can be difference on other oses.

Download and install red5 v0.6 [from here](#). (Using this version is recommended, because there can be big differences between releases). You will need [an Eclipse](#) too. Install both with basic parameters. [XMLBuddy](#) also comes handy when editing the configuration xml's, just copy it under Eclipse/plugins. For flash compiling you should use Adobe Flex 2 as an Eclipse plugin, because in this case you can edit and compile everything in Eclipse under one workspace.

If your os doesn't have a default JRE or JDK, you have to install one, get one [from Sun](#). In this case you have to play with the java CLASSPATH also, check [Sun's pages](#) for os specific howtos.

Okay, let's examine the root folder of Red5, red5.jar contains the red5-related packages, webapps folder is the home for our applications. [Jetty servlet container](#) will look for applications inside this folder, searching for configuration xml's under WEB-INF folders.

We can create a red5 application many ways, we can use [apache ant](#) to compile our new application based on red5/build.xml, but it's a little bit complicated. We will use Eclipse to write and compile our application, then start red5.

Start Eclipse, File menuitem-> New Project, choose Java Project, press Next, type Red5FirstApp for Project Name, press Finish.

There is a brand new project in Package Explorer, with a standard JRE system library. Right click to the project, New -> Folder, type WEB-INF. Create a classes and a src folder under WEB-INF .

Right click on WEB-INF/src folder -> Build Path -> Use as Source Folder, and you set WEB-INF/src folder as the root folder of our java sources.

We have to set WEB-INF/classes folder as the output folder for our build. Project menu item -> Properties -> Java Build Path menu item -> Source Path tab > Default Output Folder at the bottom -> Choose, and choose WEB-INF/classes.

We are almost ready, only the configuration files are missing from a proper [Spring](#) injection.

Start your favourite file manager, change to red5/doc/templates/myapp/WEB-INF folder, and copy every file from there to EclipseWorkspace/Red5FirstApp/WEB-INF.

Switch back to Eclipse, right click on Red5FirstApp project, choose Refresh, and the four files appear under WEB-INF.

[edit]

Part Two - Writing a basic application

Everything is ready to create a brand new application. Right click on WEB-INF/src, choose New -> Class, type com.milgra for package, Application for name, then press Finish. Our java source appeared in the editor v

This will be our main class, so we need to extend it from the ApplicationAdapter class of red5. Add red5.jar to our build path first. Project menuitem -> Properties -> Java Build Path -> Libraries -> Add External JARs, choose red5.jar in the root folder of red5. Now Eclipse can compile and tip for us.

Create appStart and appStop functions first:

```
package com.milgra;
import org.red5.server.adapter.ApplicationAdapter;
public class Application extends ApplicationAdapter
{
    public Boolean appStart ( )
    {
    }
}
```

navigation

- [Main Page](#)
- [Tutorials](#)
- [How to...](#)
- [FAQ](#)
- [Code snippets](#)
- [Downloads](#)
- [Recent changes](#)
- [Random page](#)

search

toolbox

- [What links here](#)
- [Related changes](#)
- [Upload file](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

```
public void appStop ( )
{
}
}
```

Seeing client connections would be good, but we have to import a new class for this:

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

so:

```
public boolean appConnect( IConnection conn , Object[] params )
{
    return true;
}

public void appDisconnect( IConnection conn , Object[] params )
{
}
}
```

To debug our application we need logging, so create a logger what prints our logs to the standard output. Lets import a log and logfactory.

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
```

But something is wrong, because Eclipse marks this two rows with red underline, what is the problem? Eclipse doesn't know about org.apache.commons.logging, lets add its package quickly:

Project menuitem -> Properties -> Build path -> Libraries -> Add External JARs - pick commons-logging-1.1.jar from folder red5/lib, press OK.

Log a bit:

```
private static final Log log = LogFactory.getLog( Application.class );

public boolean appStart ( )
{
    log.info( "Red5First.appStart" );
    return true;
}

public void appStop ( )
{
    log.info( "Red5First.appStop" );
}

public boolean appConnect( IConnection conn , Object[] params )
{
    log.info( "Red5First.appConnect " + conn.getClient().getId() );
    return true;
}

public void appDisconnect( IConnection conn , Object[] params )
{
    log.info( "Red5First.appDisconnect " + conn.getClient().getId() );
}
}
```

Only a simple demo logic is needed now, lets say if we pass true to the server at connection, we let the client in, if we pass false, we reject it.

```
private static final Log log = LogFactory.getLog( Application.class );

public boolean appStart ( )
{
    log.info( "Red5First.appStart" );
    return true;
}

public void appStop ( )
{
    log.info( "Red5First.appStop" );
}
}
```

```

public boolean appConnect( IConnection conn , Object[] params )
{
    log.info( "Red5First.appConnect " + conn.getClient().getId() );

    boolean accept = (Boolean)params[0];

    if ( !accept ) rejectClient( "you passed false..." );

    return true;
}

public void appDisconnect( IConnection conn , Object[] params )
{
    log.info( "Red5First.appDisconnect " + conn.getClient().getId() );
}

```

Final code will look like this:

```

package com.milgra;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import org.red5.server.api.IConnection;
import org.red5.server.adapter.ApplicationAdapter;

public class Application extends ApplicationAdapter
{
    private static final Log log = LogFactory.getLog( Application.class );

    public boolean appStart ( )
    {
        log.info( "Red5First.appStart" );
        return true;
    }

    public void appStop ( )
    {
        log.info( "Red5First.appStop" );
    }

    public boolean appConnect( IConnection conn , Object[] params )
    {
        log.info( "Red5First.appConnect " + conn.getClient().getId() );

        boolean accept = (Boolean)params[0];

        if ( !accept ) rejectClient( "you passed false..." );

        return true;
    }

    public void appDisconnect( IConnection conn , Object[] params )
    {
        log.info( "Red5First.appDisconnect " + conn.getClient().getId() );
    }
}

```

We are ready with the java code, only the configuration of the xml's is needed. Check the four files under WEB-INF folder.

log4j.properties :

application-related logging parameters

red5-web.properties :

this file is included by red5-web.xml, constant parameters can be placed here. webapp.contextPath will be the path to our application at connection, (not the folder name under webapps!!!), set it to firstapp

```

webapp.contextPath=/firstapp
webapp.virtualHosts=localhost, 127.0.0.1

```

and save it.

red5-web.xml:

spring loads and configures our application based on this file. bean handlers can also be here.

Our web handler will be the Application created above, so set it as:

```

<bean id="web.handler"
      class="com.milgra.Application"
      singleton="true" />

```

jetty will read this first. Rename the application here to firstapp, and delete the two gateway-related nodes, why use gateway when direct database connection is available from java? :)

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

OK, lets Build the project. Project menuitem -> Build Automatically, so Eclipse will recompile after every modification, and that's great.

Set up our new application under red5: create a new folder under red5/webapps named firstapp (this is what we set in red5-web.properties), and copy here WEB-INF from EclipseWorkspace/Red5FirstApp. Then start red5 with your os-specific startup file in the root of red5, and our application is ready.

[\[edit\]](#)

Part Three - Creating flash client

Okay, server-side is ready, but we should test it somehow.

Create a new Actionscript project in Eclipse/Flex, and name it Red5FirstClient.

```
package
{
    import flash.net.NetConnection;
    import flash.net.ObjectEncoding;
    import flash.events.NetStatusEvent;
    import flash.display.Sprite;

    public class Red5FirstClient extends Sprite
    {
        private var nc:NetConnection;

        public function Red5FirstClient()
        {
            // new netconnection
            nc = new NetConnection( );

            // set encoding to old amf
            nc.objectEncoding = ObjectEncoding.AMF0;

            // netstatus event listening
            nc.addEventListener( NetStatusEvent.NET_STATUS , netStatus );

            // connect to red5, passing false as parameter
            nc.connect( "rtmp://localhost/firstapp" , false );
        }

        private function netStatus ( event:NetStatusEvent ):void
        {
            trace( event.info.code );

            if ( event.info.code == "NetConnection.Connect.Rejected" )
            {
                // trace reject message
                trace( event.info.application );
            }
        }
    }
}
```

Run menuitem -> Debug. Because we passed false as connection parameter, our red5 application will reject us, check the last line of red5/jvm log in terminal.

```
[INFO] 87028 pool-2-thread-8:( com.milgra.Application.appConnect ) Red5First.appConnect 1
```

so we were connected, let's check the eclipse/flex console:

```
NetConnection.Connect.Rejected  
you passed fals . . .  
NetConnection.Connect.Closed
```

The server application rejected us, and it passed the rejection message also, it worked as we planned.

Let's change the connection parameter to true, and wonder happens:

```
nc.connect( "rtmp://localhost/firstapp" , true );
```

the result is:

```
NetConnection.Connect.Success
```

Our application is working!!!



This page was last modified 17:59, 6 June 2007.
times.

Content is available under [Attribution-Noncommercial-Share Alike 2.5](#) .
[policy](#)

This page has been accessed 7,927
times.

[Privacy](#)

[About Red5Tutorials](#)

[Disclaimers](#)

