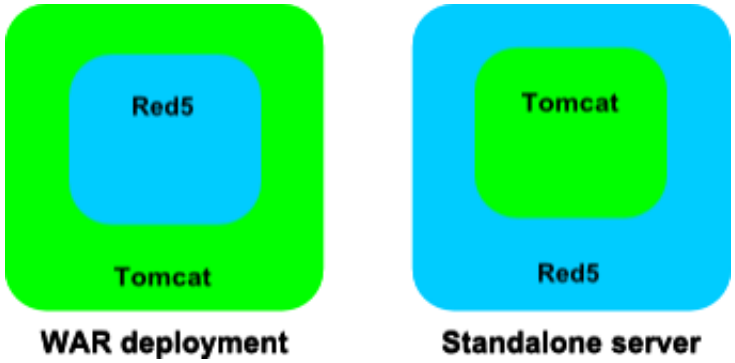
Deploying Red5 to Tomcat

Author: Paul Gregoire
Contact: mondain@gmail.com
Date: September 2007
Updated: June 2009 for Release 0.8.0

Table of Contents

Preface

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



We have a preference for Tomcat, but it is possible to use the WAR with another J2EE container.

Deployment

The Tomcat war deployer scans the *webapps* directory for wars periodically. When a war is found that has not yet been deployed, the deployer will expand the war file into a directory based on the filename of the war. A war named *myapp.war* would be expanded into a directory named *myapp*; depending upon your installation the full path would look similar to this *C:\Tomcat-6.0.18\webapps\myapp*.

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

The ROOT.war may be renamed to red5.war, but this has not been tested and would require changes to some of the configuration files. Remember that this name and its resolution within the J2EE container is primarily for HTTP requests and should not affect RTMP communications.

Context descriptors

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

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

Red5 Configuration

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

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

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

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

Spring contexts

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

Default context

The default context is synonymous with the global application context and is responsible for providing objects and resources at the top or global level. Spring beans in this level are configured via the *defaultContext.xml and beanRefContext.xml* which are located in the ROOT classes directory (ex. C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes). The bean ref file defines the *default.context* bean which as an instance of *org.springframework.context.support.ClassPathXmlApplicationContext.* Two other configuration files *red5-common.xml* and *red5-core.xml* are used to construct the default context; these files are derived from the standalone configuration files of the same names, the primary difference is that the server embedding sections have been removed.

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

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

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

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

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

Web context

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

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

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

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

- **Context** - Each context must have a unique name assigned since all the contexts exist within a single Spring application context. The root web context is named *web.context*, additional contexts suffix this base name with their web application name; for example *oflaDemo* would be named *web.context.oflaDemo*. A context is specified in the web context file as shown below.

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

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

  The root scope has the name *web.scope*, additional scope names should follow the naming convention specified for contexts. A scope for *oflaDemo* would be named *web.scope.oflaDemo* so that it will not conflict with other contexts.

  A scope bean has the following properties:
  1. server - This references the server *red5.server*
  2. parent - The parent for this scope is normally g*lobal.scope*
  3. context - Context for this scope, use the *web.context* for *root* and *web.context.oflaDemo* for *oflaDemo*
  4. handler - Handler for this scope, which is similar to a *main.asc* in FMS.
  5. contextPath - The path to use when connecting to this scope.
  6. virtualHosts - A comma separated list of host names or IP addresses this scope listens on. In the war version we do not control the host names, this is accomplished by Tomcat. A value of "*" which means "all" is expected here.

  The root scope definition looks like this:

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

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

- **Handler** - Every context needs a handler to provide the methods called by connecting clients. All handlers are required to implement *org.red5.server.api.IScopeHandler*, however you may implement

additional interfaces for controlling access to shared objects or streams. A sample implementation is provided with Red5 that may be used as your base class: *org.red5.server.adapter.ApplicationAdapter*. Please refer to the javadoc for this class for additional details.

As an example the scope handler for the *oflaDemo* is shown:

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

The *id* attribute is referenced by the *oflaDemo* scope definition:

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

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

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

External applications

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

```
<context-param>
        <param-name>tunnel.acceptor.url</param-name>
        <param-value>http://localhost:8080/gateway</param-value>
</context-param>

<context-param>
        <param-name>tunnel.timeout</param-name>
        <param-value>30000</param-value>
</context-param>

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

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

The tunnel servlet class must be on the classpath of the application under which it is executed. In addition to the tunnel servlet the

org.red5.server.net.servlet.ServletUtils class is required along with the following library jars:

```
commons-codec-1.3.jar
commons-httpclient-3.0.1.jar
mina-core-2.0.0-M6.jar
slf4j-api-1.5.6.jar
jcl-over-slf4j-1.5.6.jar
jul-to-slf4j-1.5.6.jar
log4j-over-slf4j-1.5.6.jar
logback-core-0.9.14.jar
logback-classic-0.9.14.jar
red5-remoting-0.8.0.jar (contains ServletUtils class)
```

These jars should be placed in the *WEB-INF/lib* directory of your application. ex. *C:\Tomcat-6.0.18\webapps\myapp\WEB-INF\lib*

Creating and deploying your application

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

Remote application

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

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

Steps

1. Create a web application named RemoteApp in your IDE.
2. Obtain a red5 jar, which may be downloaded from *http://red5.googlecode.com/svn/repository/red5/red5-0.8.0.jar* or built from source with the command “ant jar”. This library is needed if you extend the ApplicationAdapter for your scope handler.
3. Obtain the remoting jar, this may be accomplished by building yourself from the command line with “ant remotejar” or by downloading it from *http://red5.googlecode.com/svn/repository/red5/red5-remoting-0.8.0.jar*. This library provides the AMF tunnel servlet.
4. Place the library jars in your project library directory and add them to your build classpath.
5. Compile the Java and Flex source.
6. Create a directory named RemoteApp in the Tomcat *webapps* directory. ex. *C:\Tomcat-6.0.18\webapps\RemoteApp*
7. Copy the contents of the *web* directory to the RemoteApp directory.
8. From the *bin* directory copy the RemoteApp.swf to the *webapps\RemoteApp* directory.
9. Copy the *lib* directory and its contents to the WEB-INF, excluding the red5.jar file.
10. Copy the whole *example* directory and the *RemoteApp-web.xml* file from the *bin* directory to the *classes* directory under ROOT. ex. *C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes*
11. Restart tomcat
12. Open your browser and go to: http://localhost:8080/RemoteApp/RemoteApp.html

13. Click on the RTMP or HTTP connect buttons. For a successful test you should see a server response of “Hello World”.

Local application

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

Steps

1. Create a web application named LocalApp in your IDE.
2. Obtain a *red5.jar*, which may be downloaded from *http://red5.googlecode.com/svn/repository/red5/red5-0.8.0.jar* or built from source with the command “ant jar”. This library is needed if you extend the ApplicationAdapter for your scope handler.
3. Place the library jar in your project library directory and add it to your build classpath.
4. Compile the Java and Flex source.
5. Copy the LocalApp.html and LocalApp.swf from the *bin* directory to the ROOT directory. ex. *C:\Tomcat-6.0.18\webapps\ROOT*
6. Copy the whole *example* directory and the *LocalApp-web.xml* file from the *bin* directory to the *classes* directory under ROOT. ex. *C:\Tomcat-6.0.18\webapps\ROOT\WEB-INF\classes*
7. Restart tomcat
8. Open your browser and go to: http://localhost:8080/LocalApp.html
9. Click on the connect button. For a successful test you should see a server response of “Hello World”.

Example Source

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

Additional web configuration

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

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

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

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

```
<servlet>
        <servlet-name>rtmpt</servlet-name>
        <servlet-class>org.red5.server.net.rtmpt.RTMPTServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>rtmpt</servlet-name>
        <url-pattern>/fcs/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
        <servlet-name>rtmpt</servlet-name>
        <url-pattern>/open/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
        <servlet-name>rtmpt</servlet-name>
        <url-pattern>/idle/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
        <servlet-name>rtmpt</servlet-name>
        <url-pattern>/send/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
        <servlet-name>rtmpt</servlet-name>
        <url-pattern>/close/*</url-pattern>
</servlet-mapping>
```

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

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

<security-constraint>
        <web-resource-collection>
```

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

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

Troubleshooting

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

1. Stop the Tomcat server

2. Locate your Tomcat installation directory
   C:\Program Files\Apache\Tomcat

3. Delete the "work" directory
   C:\Program Files\Apache\Tomcat\work

4. Delete the "Catalina" directory from the "conf" directory
   C:\Program Files\Apache\Tomcat\conf\Catalina

5. Delete the expanded war directories, if they exist
   C:\Program Files\Apache\Tomcat\webapps\ROOT
   C:\Program Files\Apache\Tomcat\webapps\echo
   C:\Program Files\Apache\Tomcat\webapps\SOSample

6. Ensure your WAR files are in the webapps directory
   C:\Program Files\Apache\Tomcat\webapps\ROOT.war
   C:\Program Files\Apache\Tomcat\webapps\echo.war
   C:\Program Files\Apache\Tomcat\webapps\SOSample.war

7. Restart Tomcat

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

1. Java version

2. Tomcat version

3. Operating system

4. Red5 version (0.6.2, Trunk, Revision 2283, etc...)

Definitions

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

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

AS3 – A scripting language based on ECMAScript, used primarily for the development of websites and software using the Adobe Flash Player platform.

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

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

RTMPT – RTMP using HTTP tunneling.

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

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

Bibliography

Red5 – http://osflash.org/red5

Apache Tomcat – http://tomcat.apache.org

Wikipedia – http://en.wikipedia.org