Resin Administration

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Starting Resin

1.1 Preconditions

Resin requires JDK 6.0 or later. An HTML 5 browser for some /resin-admin features.

Oracle's JDK for Windows, Solaris, and Linux can be found at http://www.oracle.com/technetwork/java/javase/downloads/-index.html . Oracle also has links to some other ports of the JDK. Resin will not be fully functional while using a JRE.

1.2 Directory Overview

When Resin is installed on a Unix system: resinctl is the starting-resin-command-line.xtp . config-resin-properties.xtp are in /etc/resin/resin.properties deploy.xtp are put in /var/resin/webapps Logs are in /var/log/resin Licenses belong in /etc/resin/licenses OpenSSL keys belong in /etc/resin/keys

Unix Directories

```
/etc/resin/resin.properties  # configuration properties
/etc/resin/resin.xml         # configuration file
/etc/resin/licenses/         # license location
/etc/resin/keys/             # openssl keys

/var/resin/                  # resin.root (content)
/var/resin/webapps/          # default deployment directory

/var/log/resin/              # resin logs

/usr/local/share/resin/      # resin.home (binaries and libraries)
```

When Resin is installed on a Windows system: resin is the starting-resin-command-line.xtp . config-resin-properties.xtp are in conf/resin.properties deploy.xtp are in webapps Logs are in log Licenses belong in conf/licenses OpenSSL keys belong in conf/keys

Windows/unzip Directories

```
resin-4.0.x/                 # installation directory: resin.home, resin.root
  conf/resin.properties      # configuration properties
  conf/resin.xml             # configuration file
  conf/licenses/             # license location
  conf/keys/                 # openssl keys

  webapps/                   # default deployment directory

  logs/                      # resin logs
```

1.3 Installing Resin

Installation steps for the major operating systems are outlined below:

1.3.1 Debian Unix .deb and apt-get

Debian users can download a .deb packaged version of Resin or use apt-get to install Resin. The Debian package performs all of the installation steps above for you and creates all the recommended server and content directories. Simply download from the http://caucho.com/download and install using dpkg.

Alternatively, you can add Caucho's Debian repository to your system's repositories to use automated update tools like Synaptic and apt-get. To do this, you can add the debian repository as follows:

```
unix# add-apt-repository http://caucho.com/download/debian
```

After adding this line, update your local repository cache by running:

```
unix# apt-get update
```

Install Resin Professional with this command:

```
unix# apt-get install resin-pro
```

Or install Resin Open Source with this command:

```
unix# apt-get install resin
```

1.3.2 RedHat and CentOS .rpm files

RPM files are available at http://caucho.com/download . The RPM public key is at http://caucho.com/download/rpm/RPM-GPG-KEY-caucho .

RPM/yum install

```
unix# rpm --import http://caucho.com/download/rpm/RPM-GPG-KEY-caucho
unix# yum install http://caucho.com/download/rpm/4.0.30/x86_64/resin-pro-4.0.30-1.x86_64. ↩
    rpm
```

1.3.3 Other Unix, Linux, Solaris, and Mac OS X

Install JDK 6 or later and link /usr/java to the Java home or define the environment variable JAVA_HOME . `tar -vzxf resin-4.0.x.tar.gz`

`cd resin-4.0.x`

`./configure` some details on the starting-resin-command-line.xtp . `make`

`sudo make install` Execute `sudo resinctl start` or run `java -jar lib/resin.jar start` Browse to http://localhost:8080

1.3.4 Windows

Install JDK 6 or later. Check that the environment variable JAVA_HOME is set to the JDK location, e.g. `c:\java\jdk1.6.0_14` Unzip resin-4.0.x.zip Define the environment variable RESIN_HOME to the location of Resin, for example `c:\resin-4.0.x` Execute `resin.exe` or run `java -jar lib/resin.jar start` Browse to http://localhost:8080

1.4 Installing a license file for Resin Pro

After installing Resin Professional, you need to install the license file.

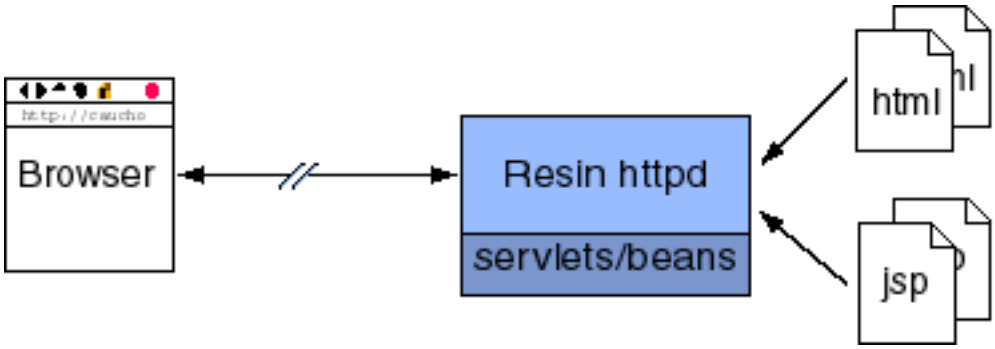
If you have a license file for Resin, save it in /etc/resin/licenses . You can also install the license from the command line with:

```
unix> resinctl license-add mylicense.license
```

1.5 Resin as a Web Server

Resin includes a high-performance HTTP server that outperforms NginX and Apache httpd. The easiest and fastest Resin configuration uses Resin as the web server as well as the application server. We highly recommend you start with this configuration although you are free to use other web servers like Apache or IIS with Resin.

Keep in mind, Resin can be used for development or evaluation in addition to using it in production. If desired, you can easily apply for a development license to enable Resin Pro features. You can also simply use Resin Open Source to start development.



The built-in HTTP server listens on port 8080 in the default configuration and can be changed to listen on the HTTP port 80 instead during deployment.

Example: Starting Resin

```
unix> resinctl start

c:\windows> resin start
```

For troubleshooting your installation, you can also run Resin in "console" mode, which will let you see the logging messages in your console.

Example: Starting Resin in console mode

```
unix> resinctl console
```

For more details, see the http-server.xtp configuration page.

1.6 Browser-Based Administration (/resin-admin)

To enable the /resin-admin browser-based administration, you'll need to create an admin user and password. You can either create the user in resin-admin itself, or if you are using the standard resin.xml and resin.properties you can generate the key from the command line resinctl

1.6.1 Installation with resin.properties and generate-password

You can generate the user and password properties from the command-line. Resin's password must be hashed for security.

Example: generate-password

```
unix> resinctl generate-password my-user my-password
admin_user : my-user
admin_password : {SSHA}HTfP60Ceq0K0IAvN31wQtBxtql9D+8Wo
```

Add the admin_user and admin_password lines to the end of the /etc/resin/resin.properties file. You may also use those values to change your admin-users.xml file.

You can update the resin.properties in one step on unix by using a pipe:

Example: generate-password for resin.properties

```
unix# resinctl generate-password my-user my-password >> /etc/resin/resin.properties
```

1.6.2 Installation with /resin-admin

Create an admin user following the prompt at /resin-admin. Rename admin-users.xml.generated to admin-users.xml . Change the resin_admin_external to true in the resin.xml if you need access from a non-local IP address. Browse /resin-admin with an HTML 5 browser.

On Linux, admin-users.xml.generated is typically generated in /etc/resin/ .

The steps are for security reasons. Copying the admin-users.xml verifies that you have access to the server. And the default resin_admin_external=false makes sure you're not exposing the /resin-admin to the internet.

For more information, see the resin-admin-console.xtp .

1.7 Deploying Applications

Once you've made sure Resin is working, you can start to run applications and add content.

1.7.1 Local network command-line deployment

Command-line deployment on a local network deploys a .war file to a running Resin server, using the .war file's name as the context-path. It looks like the following example.

Example: command-line deployment

```
unix> resinctl deploy hello.war

unix> resinctl undeploy hello.war
```

The URL for the application would be: http://localhost:8080/hello .

To deploy to the root context-path, use a file name ROOT.war or use the --name attribute.

Example: root command-line deployment

```
unix> resinctl deploy --name ROOT mywar.war

unix> resinctl undeploy --name ROOT
```

1.7.2 Remote network command-line deployment

Deploying to a remote network requires some more configuration for security reasons: enable remote administration (disabled by default) configure an admin user and password

After the changes, resin.properties will look something like:

Example: resin.properties for remote deployment

```
...

admin_user          : my-user
admin_password      : {SSHA}HTfP60Ceq0K0IAvN31wQtBxtql0D+8Wo
admin_remote_enable : true
```

When you deploy, you will need to give the user and password:

Example: remote deploy

```
unix> resinctl deploy --user my-user --password my-password hello.war
```

1.7.3 webapps directory deployment

You can deploy .war files by copying them to the webapps directory like this: `resin-4.0.x/webapps/hello.war` . The URL for the application would be: http://localhost:8080/hello .

You can also deploy .wars in exploded form like: `resin-4.0.x/webapps/hello/index.php` . The URL for the application would be: http://localhost:8080/hello .

You can use a web.xml file to configure the *hello* web application:

`resin-4.0.x/webapps/hello/WEB-INF/web.xml` .

For more information on deployment, see the deploy.xtp .

1.8 Running Resin as a Daemon

Most production environments will run Resin as a background daemon. When running as a daemon, Resin detaches from the console and continues running until it is stopped.

The following are the basic steps to running Resin as a daemon: Start resin with `resinctl start` Stop resin with `resinctl stop` Restart resin with `resinctl restart`

The .rpm and .deb files install Resin in /etc/init.d/resin, which will start Resin when the system boots.

1.9 Command-Line PDF Reports

PDF reports about the Resin server can be generated with the command-line, the /resin-admin browser-based GUI, or as automatic health system tasks.

From the command-line you can generate a pdf-report with the following:

Example: generating a PDF snapshot report

```
unix> resinctl pdf-report

generated /var/resin/log/default-Watchdog-20111010T1426.pdf
```

You can also generate a report for the most recent restart event saved by the watchdog. The watchdog report will give information about why Resin was last stopped.

Example: generating a PDF watchdog report

```
unix> resinctl pdf-report -watchdog

generated /var/resin/log/default-Watchdog-20111010T1426.pdf
```

1.10 Resin IDE Support

Resin includes excellent support for Eclipse. In fact, using the Eclipse support may be the easiest way get started with Resin.

Resin plugin support is included in Indigo (Eclipse 3.7) and above (you can install the Resin plugin manually for earlier Eclipse versions). The Resin plugin allows you to create new server instances, configure servers, start servers, stop servers, restart instances, deploy/undeploy applications, debug on the server and so on. You can even automatically download and install the latest version of Resin using the plugin or apply for a development license for Resin Pro.

Further details on the Resin Eclipse plugin is available resin-eclipse-support.xtp .

1.11 More Installation Options

More installation and configuration are available at install.xtp .

1.12 Next Steps

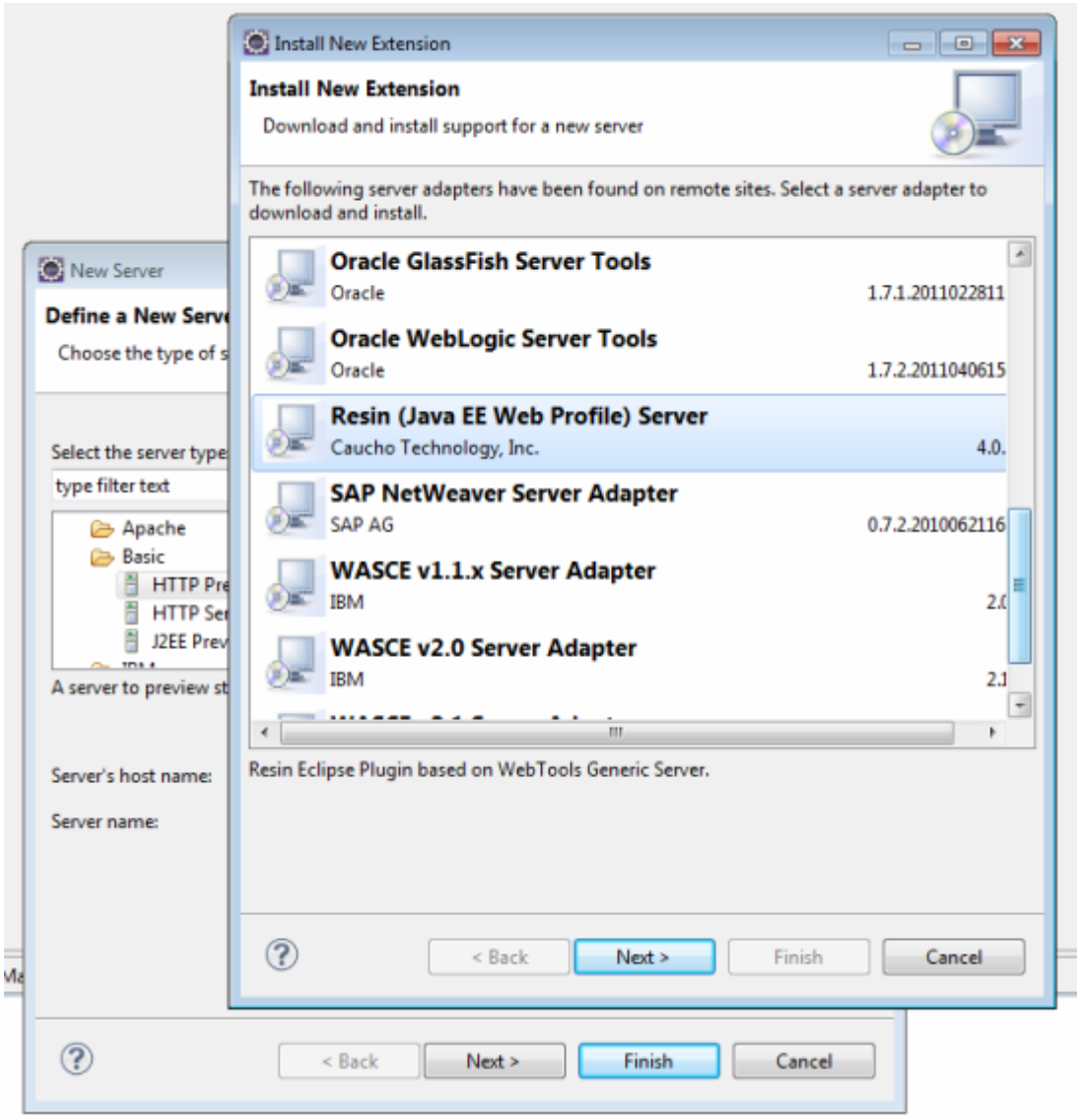
config-resin-properties.xtp configuration shows how to customize basic configuration. config-resin-xml.xtp configuration shows more advanced and specialized configuration. resin-admin-command-line.xtp describes using the resinctl command line interface. deploy.xtp deploying applications to a running server. clustering.xtp describes clustering, cloud, and dynamic servers. resin-admin.xtp describes the /resin-admin browser-based interface. health.xtp describes the Resin health and monitoring system. resin-admin-rest.xtp describes the REST interface for remote third-party admin integration. http.xtp describes fast, scalable HTTP web server. http-rewrite.xtp describes Resin's URL rewriting (like mod_rewrite). security.xtp describes authentication, authorization and SSL.

Chapter 2

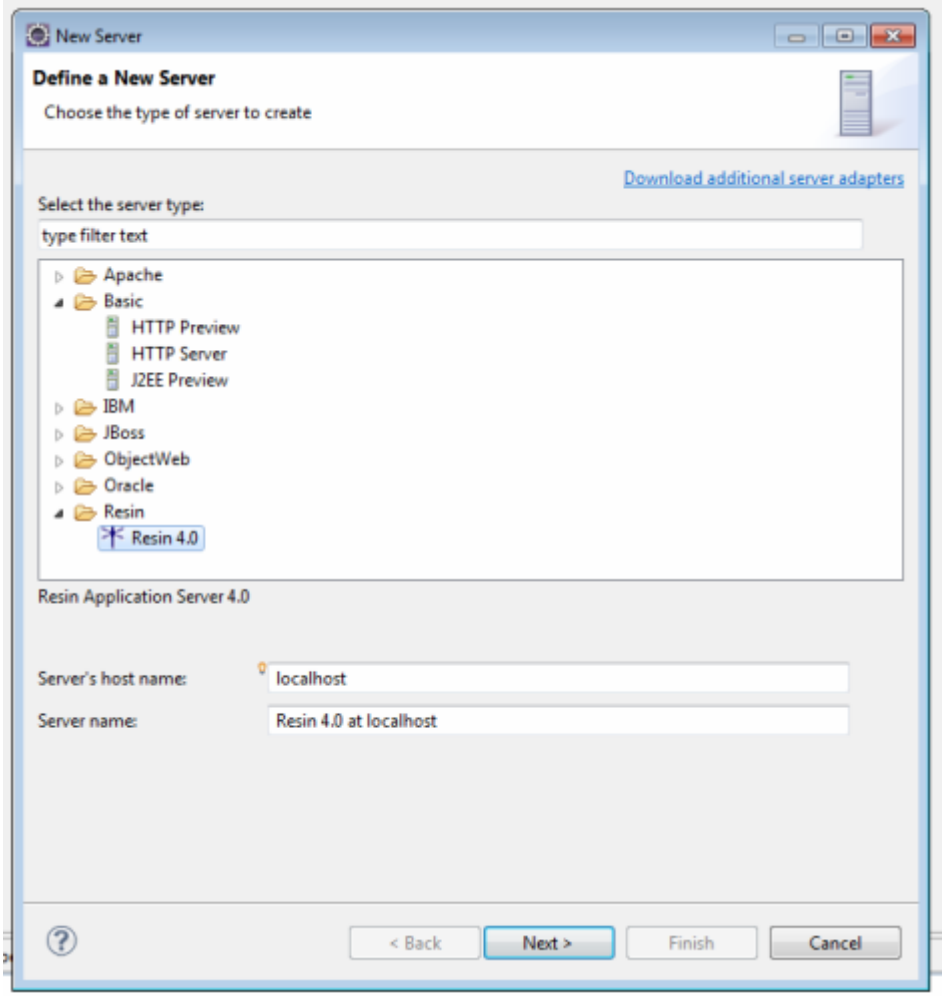
Eclipse

2.1 Installing the Resin Plugin

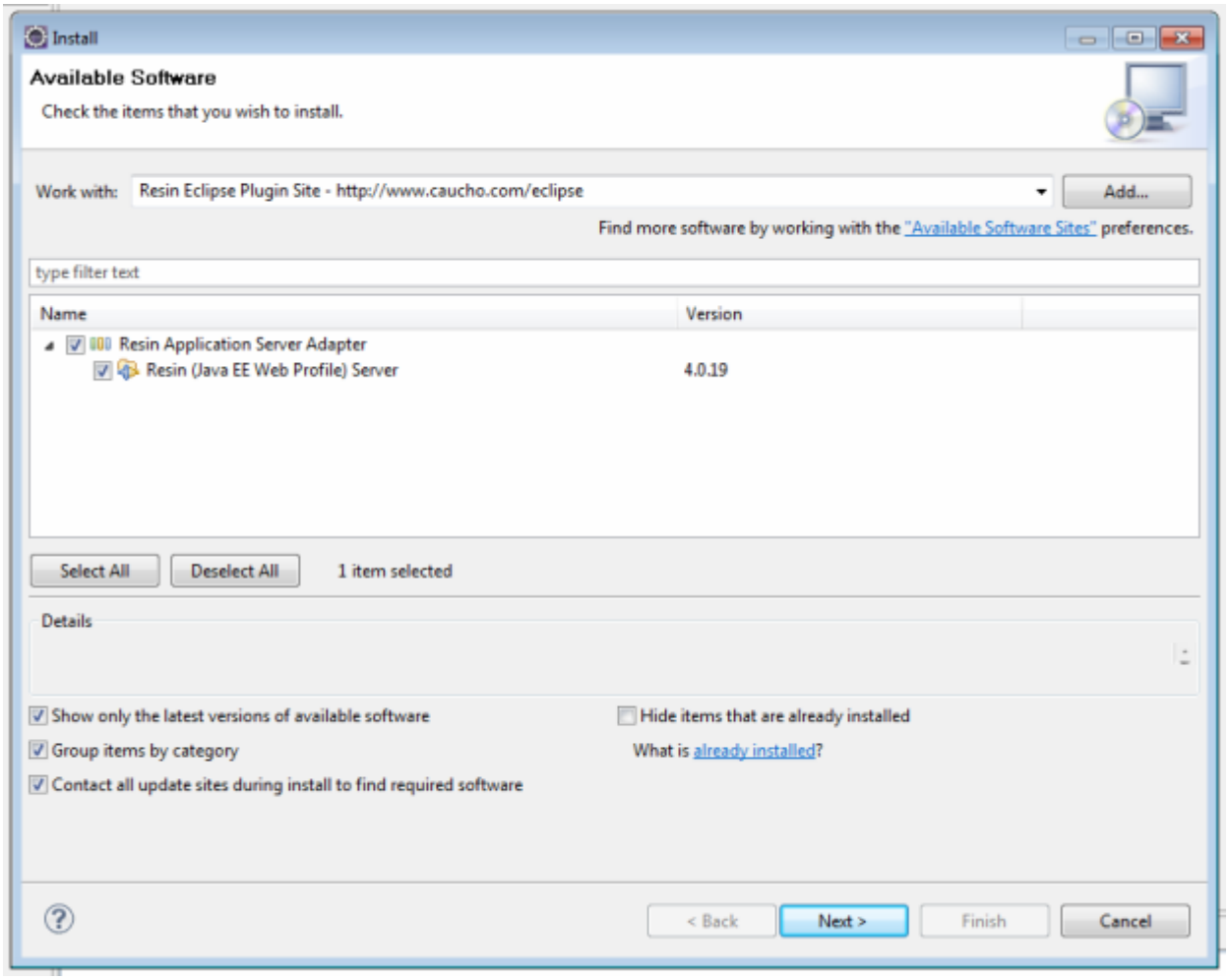
The Resin Eclipse plugin supports Helios (Eclipse 3.6) and above. Indigo (Eclipse 3.7) and above already comes with the Resin plugin registered. To download the Resin plugin, go to the *New Server* dialog and click on *Download additional server adapters*. In the *Install New Extension* dialog, you'll see Resin as an option.



Once the plugin is installed, you'll see Resin 4.0 as an option in the *New Server* dialog. When you select Resin, the wizard will take you through the steps to create a server instance in Eclipse.

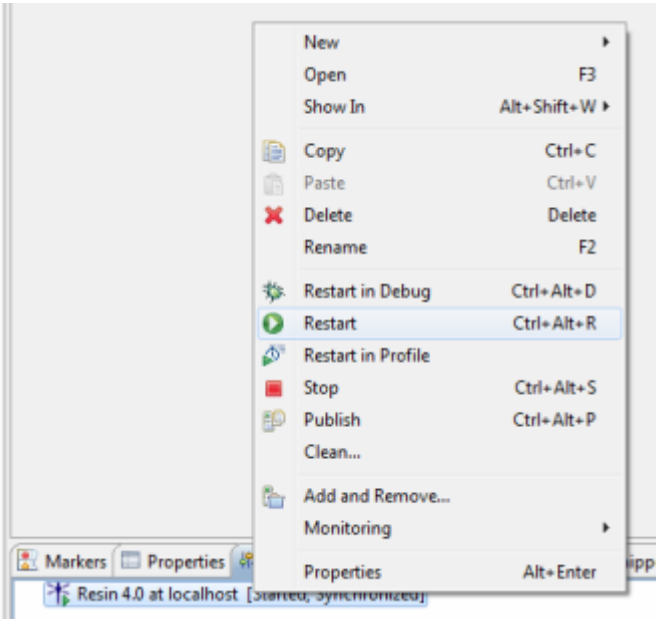


If you're using Helios, you will need to install the Resin plugin yourself directly from the Resin Eclipse plugin website (http://www.caucho.com/eclipse/). You can do this by registering the plugin website as an Available Software Site in Eclipse. Once you've done that, you'll be able to install the plugin manually.



2.2 Using the Eclipse Plugin

The Resin plugin supports all of the common server operations supported by WTP. You can start a server, stop a server, restart a server, debug on the server (including debugging remote servers), profile a server, add/remove projects to the server, clean the server, deploy/redeploy applications and so on. These operations are accessed either by right-clicking the server instance or right-clicking a project associated with a Resin server. If you are an experienced Eclipse user you should feel right at home.



For greater details on the Resin Eclipse plugin, check out the http://wiki.caucho.com/Eclipse .

Chapter 3

Migrating to Resin 4.0

3.1 Migrating from Resin 3.0 to Resin 4.0

The Caucho Wiki features a full http://wiki.caucho.com/Migrating_from_Resin_3.0_to_Resin_4.0 for those upgrading from Resin 3.0.

3.2 Migrating from WebLogic

The Caucho Wiki features http://wiki.caucho.com/Migrating_from_WebLogic_to_Resin for those transitioning from WebLogic to Resin.

3.3 Deprecated configuration

```
 The following is a list of deprecated tags in Resin 4.0 that should
no longer be used:
<accept-buffer-size> (see accept-listen-backlog ) <authenticator> (see resin:PropertiesAut
```

Chapter 4

Administration web application

4.1 /resin-admin web-app

The /resin-admin web-app provides an administration overview of a Resin server. Resin-Pro users can obtain information across the entire cluster, profile a running Resin instance, and obtain thread dumps and heap dumps.

4.1.1 Configuring /resin-admin

Since /resin-admin is just a web-app implemented with Quercus/PHP, enabling it is just adding an appropriate <web-app> tag.

To enable the /resin-admin, you'll need to create an admin user and password. Create an admin user following the prompt at /resin-admin. Copy the conf/admin-users.xml.generated to conf/admin-users.xml . Change the resin_admin_external to true if you need access from a non-local IP address. Browse /resin-admin with an HTML 5 browser.

The steps are for security reasons. Copying the admin-users.xml verifies that you have access to the server. And the default resin_admin_external=false makes sure you're not exposing the /resin-admin to the internet.

4.1.1.1 Adding /resin-admin users

The admin-users.xml contains the admin users and passwords who are allowed access to /resin-admin. The format is an XML file where each user has password scheme used by LDAP. Since the /resin-admin login page includes an passwords.

Example: sample admin-users.xml

```
<resin:AdminAuthenticator xmlns="http://caucho.com/ns/resin"
                          xmlns:resin="urn:java:com.caucho.resin">
  <user name="admin" password="{SSHA}h5QdSulQyqIgYo7BIJ3YfnRSY56kD847"/>
  <user name="resin" password="{SSHA}J0y0dlG2uRKT83g3OfzEjFNTIaTHVsA9"/>
</resin:AdminAuthenticator>
```

4.1.1.2 Allowing external internal access

Example: exposing /resin-admin to the internet

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  ...

  <web-app id="/resin-admin" root-directory="${resin.root}/doc/admin">
     <prologue>
```

```
        <!-- allow access from any IP address -->
        <resin:set var="resin_admin_external" value="true"/>
        <resin:set var="resin_admin_insecure" value="true"/>
     </prologue>
  </web-app>

</host>
</cluster>
</resin>
```

4.1.2 /resin-admin summary tab

The summary tab provides a basic overview of the Resin instance. Resin-Pro users can see the summary page of each server in a cluster.

The overview section shows basic configuration information like the server-id, Resin version, Resin home, memory information, and how long the instance has been up. It's useful as a basic check to verify the configuration and see if the server is having any problems.

4.1.2.1 Thread pool

The thread pool give the current state of Resin's threads.

4.1.2.2 TCP ports

The TCP ports gives information about the HTTP, HTTPS, and cluster ports, showing how many threads are active and also how many connections are in various keepalive states.

4.1.2.3 Server Connectors - Load Balancing

The Server Connectors section is the main section for load balancing. It will give an overview of any failures in connecting to the backend servers, showing the latency and load.

4.1.2.4 Connection pools - Database pooling

The connection pool section shows the state and history of the database pools.

4.1.2.5 WebApps

The WebApps shows the current state of the active web-apps for each virtual host. In particular, it will show the time and number of any 500 errors, letting you track down errors in the log files.

4.1.3 /resin-admin config tab

The config tag summarizes Resin's internal configuration. This tab can be useful to double-check that the values in the resin.xml and web.xml match the expected values.

4.1.4 /resin-admin threads tab

The threads tab is a critical debugging tab. It shows the state and stack trace of every thread in the JVM, grouped by functionality. If the server ever freezes or moves slowly, use the thread tab as your first resource for figuring out what's going on in the system.

All Resin users should familiarize themselves with the thread dump for their application. It's very important to understand the normal, baseline status, so if something does go wrong, you'll know what looks different.

In particular, any freeze or slowness of Resin should immediately suggest looking at the thread page.

4.1.5 /resin-admin cpu profile tab

The cpu profile tab lets you gather basic profiling information on a running Resin server. Because the overhead is low, it's even possible to run a profile on a deployment server, which will give much better information about the performance of your system than any artificial benchmarks.

With Resin's integrated profiling, there's no excuse to skip the profiling step for your application.

4.1.6 /resin-admin heap dump tab

The heap dump tab lets you gather a heap memory information at any time, giving you critical information at times when you may not have a dedicated profiler available.

4.2 /resin-admin Custom Configuration

For advanced users, you can change the standard AdminAuthenticator to be any of the Resin authenticators with a little extra configuration. The default authenticator uses an XML file called admin-users.xml to define the admin users.

You can create an alternative authenticator by configuring it and setting its CDI name to be "resinAuth". For example, the following will configure an XmlAuthenticator.

resin.xml custom /resin-admin configuration

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:ee="urn:java:ee">
       xmlns:resin="urn:java:com.caucho.resin">

  <resin:XmlAuthenticator ee:Named="resinAuth">
    <user name="admin" password="{SSHA}h5QdSulQyqIgYo7BIJ3YfnRSY56kD847"/>
  </resin:XmlAuthenticator>

  ...

<cluster id="">
<host id="">

  <web-app id="/resin-admin" root-directory="${resin.root}/doc/admin">
     ...
  </web-app>
```

4.3 Admin topics

4.3.1 Interpreting the proxy cache hit ratio

The proxy cache is Resin's internal proxy cache (in Resin Pro). The hit ratio marks what percentage of requests are served out of the cache, i.e. quickly, and which percentage are taking the full time.

The proxy cache hit ratio is useful for seeing if you can improve your application's performance with better caching. For example, if you had a news site like www.cnn.com, you should have a high hit rate to make sure you're not overtaxing the database.

If you have a low value, you might want to look at your heavily used pages to see if you can cache more.

4.4 Resin's JMX Interfaces

See com.caucho.management.server . BlockManagerMXBean - performance data for the proxy cache, clustered sessions, and JMS queues. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ConnectionPoolMXBean - JDBC database pools. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/HostMXBean - virtual host management. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/JmsQueueMXBean - jms queue management. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/JmsTopicMXBean - jms topic management. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/LoggerMXBean - java.util.logging management. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/PortMXBean - http, https, and custom TCP ports. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ProxyCacheMXBean - Resin's integrated proxy cache. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ResinMXBean - Parent MXBean for Resin corresponding to the <resin> configuration tag. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ServerConnectorMXBean - client view of a peer server in a cluster. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ServerMXBean - information about the JVM http://wiki.caucho.com/Server instance. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/SessionManagerMXBean - information about a web-app's session manager. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/ThreadPoolMXBean - information about Resin's internal thread pool. http://caucho.com/resin-4.0-javadoc/com/caucho/management/server/WebAppMXBean - information about a Resin web-app.

4.5 JMX Instrumenting Beans

Resin's IoC container can register the application's services with JMX automatically. The registered beans will then be available in a JMX application like jconsole or through PHP or Java. For a class `MyFoo` , create an interface `MyFooXMBean` with the management interface. Class `MyFoo` needs to implement the `MyFooMBean` interface. When Resin handles the <bean> tag, it will register `MyFoo` with the JMX server.

4.5.1 Instrumenting a bean

Resin will automatically register any servlet which implements an MBean interface. By default, the JMX name will be:

```
resin:name= name ,type= type ,Host= name ,WebApp= name
```

Table 4.1: ObjectName attributes

ATTRIBUTE	VALUE
type	The FooMBean name minus the MBean, e.g. "Foo"
name	the bean's name value
Host	the virtual host name
WebApp	the web-app's context path

The domain is web-app , the type property is calculated from the MBean class name and the name property is the value of <name>.

JMX clients will use the name to manage the bean. For example, a client might use the pattern web-app:type=Foo,* to retrieve the bean.

MyServiceMBean.java

```
package test;

public interface MyServiceMBean {
  public int getCount();
}
```

MyServlet.java

```
package test;

import java.io.*;
import javax.servlet.*;

public class MyService implements MyServiceMBean
{
  private int \_count;

  public int getCount()
  {
    return \_count;
  }

  public void doStuff()
  {
    \_count++;
  }
}
```

4.5.2 PHP: Displaying and Managing Resources

The easiest way to display and manage JMX is with PHP. The /resin-admin web-app provides several examples of getting JMX data.

PHP: Getting the Count attribute

```
<php?

$mbean_server = new MBeanServer();

$service = $mbean_server->query("resin:*,type=MyService");

echo "Service.count: " . $service[0]->Count . "\n";

?>
```

4.6 JMX Console

JDK 5.0 includes a JMX implementation that is used to provide local and remote administration of a Resin server. The JVM will expose JMX if it's started with appropriate -D system properties. For example, -Dcom.sun.management.jmxremote will expose JMX to the local machine.

To configure the JVM arguments for Resin, you'll add a <jvm-arg> to the resin.xml. When Resin's http://wiki.caucho.com/-Watchdog process starts Resin, it will pass along the configured arguments, enabling JMX administration.

Start Resin and allow local JMX administration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <jvm-arg>-Dcom.sun.management.jmxremote</jvm-arg>
  </server-default>

  ...
```

```
</cluster>
</resin>
```

Start jconsole

```
win> jconsole.exe
unix> jconsole

Choose Resin's JVM from the "Local" list.
```

Start Resin and allow remote JMX administration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <jvm-arg>-Dcom.sun.management.jmxremote.port=9999</jvm-arg>
  </server-default>

  ...

</cluster>
</resin>
```

Without some configuration effort, the previous command will not work. Password configuration and SSL configuration is required by the JDK implementation of remote JMX. Detailed instructions are included in the JDK documentation.

The following is useful for testing, but should be done with caution as the port is not protected by password or by SSL, and if not protected by a firewall is accessible by anyone who can guess the port number.

Start Resin and remote JMX - disable password checking and SSL

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <jvm-arg>-Dcom.sun.management.jmxremote.port=9999</jvm-arg>
    <jvm-arg>-Dcom.sun.management.jmxremote.ssl=false</jvm-arg>
    <jvm-arg>-Dcom.sun.management.jmxremote.authenticate=false</jvm-arg>
  </server-default>

  ...

</cluster>
</resin>
```

Start jconsole

```
win> jconsole.exe
unix> jconsole

Enter the host name and port number (9999) on the "Remote" tab
```

Setting a password for remote JMX access

```
$ cd $JAVA_HOME/jre/lib/management
$ cp jmxremote.password.template jmxremote.password
$ chmod u=rw jmxremote.password
$ vi jmxremote.password

Set a password for "monitorRole" and "controlRole":

monitorRole 12monitor
controlRole 55control
```

Start Resin and remote JMX - disable SSL

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <jvm-arg>-Dcom.sun.management.jmxremote.port=9999</jvm-arg>
    <jvm-arg>-Dcom.sun.management.jmxremote.ssl=false</jvm-arg>
  </server-default>

  ...

</cluster>
</resin>
```

Start jconsole

```
win> jconsole.exe
unix> jconsole
```

Enter the host name and port number (9999) on the "Remote" tab Enter the username and password on the "Remote" tab

4.7 SNMP

Since 3.1.5, Resin has built-in support for SNMP (Simple Network Management Protocol). This allows Resin to be managed just like any network device (e.g. routers) from an SNMP manager application.

4.7.1 Enabling SNMP support in Resin

To enable Resin's SNMP service, you'll need to add an SNMP protocol tag to your resin.xml under the <server> tag:

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="">
    <server-default>

      <protocol class="com.caucho.server.snmp.SnmpProtocol" port="161"/>

    <server-default/>

    ...

  <cluster/>
<resin/>
```

This opens up a port that listens for SNMP requests and responds to them with an SNMP response. Currently, Resin can only respond to TCP SNMP get-pdu requests.

By default, the SNMP community string is "public". It can be changed with:

```
<protocol class="com.caucho.server.snmp.SnmpProtocol" port="161">
  <init community=" insert_password_here "/>
<protocol/>
```

4.7.2 MIB Variables

Internally, Resin stores a mapping from SNMP MIB variables to MBean attributes. Requests for a specific MIB variable are simply retrievals for the corresponding MBean attribute. The available MIB mappings are hard-coded in Resin and they are:

Table 4.2: SNMP to MBean mappings

SNMP OBJECT ID	SNMP TYPE	MBEAN	MBEAN ATTRIBUTE
1.3.6.1.2.1.1.1	Octet String	resin:type=Resin	Version
1.3.6.1.2.1.1.3	Time Ticks	java.lang:type=Runtime	UpTime
1.3.6.1.2.1.1.5	Octet String	resin:type=Host,name=default	URL
1.3.6.1.4.1.30350.1.1	Gauge	resin:type=Server	KeepaliveCountTotal
1.3.6.1.4.1.30350.1.2	Gauge	resin:type=Server	RequestCountTotal
1.3.6.1.4.1.30350.1.3	Gauge	resin:type=Server	RuntimeMemory
1.3.6.1.4.1.30350.1.4	Gauge	resin:type=Server	RuntimeMemoryFree
1.3.6.1.4.1.30350.1.5	Gauge	resin:type=Server	ThreadActiveCount
1.3.6.1.4.1.30350.1.6	Gauge	resin:type=Server	ThreadKeepaliveCount
1.3.6.1.4.1.30350.2.1	Gauge	resin:type=ThreadPool	ThreadActiveCount
1.3.6.1.4.1.30350.2.2	Gauge	resin:type=ThreadPool	ThreadCount
1.3.6.1.4.1.30350.2.3	Gauge	resin:type=ThreadPool	ThreadIdleCount
1.3.6.1.4.1.30350.2.4	Gauge	resin:type=ThreadPool	ThreadIdleMax
1.3.6.1.4.1.30350.2.5	Gauge	resin:type=ThreadPool	ThreadIdleMin
1.3.6.1.4.1.30350.2.6	Gauge	resin:type=ThreadPool	ThreadMax
1.3.6.1.4.1.30350.3.1	Gauge	resin:type=ProxyCache	HitCountTotal
1.3.6.1.4.1.30350.3.2	Gauge	resin:type=ProxyCache	MissCountTotal

4.7.3 Defining your own SNMP to MBean mappings

To define your own MIB variables, you'll need to extend the com.caucho.server.snmp.SnmpProtocol class and then use that class name in the <protocol> tag:

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="">
    <server-default>

      <protocol class="example.MySnmpProtocol" port="161"/>

    <server-default/>

    ...

  <cluster/>
<resin/>
```

example.MySnmpProtocol

```
package example;

import com.caucho.server.snmp.*;
import com.caucho.server.snmp.types.*;

public class MySnmpProtocol extends SnmpProtocol
{
  public MySnmpProtocol()
  {
    super();

    addOid(new Oid("1.2.3.4.5",
                   " my_mbean_object_name ",
                   " my_mbean_attribute ",
```

```
                    SnmpValue.OCTET_STRING));
  }
}
```

"1.2.3.4.5" is the SNMP ID you choose to give to your mapping. It should be in dot notation. SnmpValue.OCTET_STRING is the type Resin should return for that attribute. An abbreviated list of the available types are:

SNMP TYPES	DESCRIPTION
SnmpValue.OCTET_STRING	8-bit String
SnmpValue.INTEGER	signed 32-bit integer
SnmpValue.COUNTER	unsigned 32-bit integer that only increases, wraps around when overflows
SnmpValue.GAUGE	unsigned 32-bit integer that may increase and decrease
SnmpValue.TIME_TICKS	unsigned 32-bit integer representing time measured in hundredths of a second
SnmpValue.IP_ADDRESS	IP address

For a more complete list, see http://caucho.com/resin-4.0-javadoc/com/caucho/server/snmp/package-summary.html .

Chapter 5

Administration: REST

5.1 Overview

REST interface is provided for convenience of integration with administration and monitoring tools that are capable of using REST as integration protocol. Resin's set of REST actions covers functionality available via CLI and adds an extra action that allows retrieving Resin statistics data.

By default Resin REST is disabled. Default configuration also requires secure protocol.

REST actions, available via the interface, are required to be invoked with either GET or POST methods. Depending on the effect on the server the proper HTTP method is required: actions change state of the Resin's internal objects require POST method. Other actions must use GET.

5.2 enabling the interface

By default the interface is disabled. Un-commenting line

```
rest_admin_enable :  true
```

in resin properties enables the REST service.

By default, the connection is required to be made over a secure protocol(HTTPS). Relaxing the secure constraint can be done with setting

```
rest_admin_secure
```

to `true` .

Example: enabling Resin Secure REST interface

```
        # Enable Resin REST Admin
        rest_admin_enable : true

        # Require SSL for REST Admin
        rest_admin_secure : true
```

Resin REST Servlet is registered in

```
cluster-default.xml
```

file (see excerpt below for a reference).

Reference: registered REST Servlet

```
<resin:if test="${rest_admin_enable}">
  <web-app id="/resin-rest"
              root-directory="${resin.root}/doc/resin-rest">

    <resin:BasicLogin realm-name="resin"/>

    <resin:Allow url-pattern="/*">
      <resin:IfUserInRole role="resin-admin"/>
    </resin:Allow>

    <servlet-mapping url-pattern="/*"
                        servlet-class="com.caucho.admin.servlet.AdminRestServlet">
      <init>
        <require-secure>${rest_admin_secure}</require-secure>
      </init>
    </servlet-mapping>
  </web-app>
</resin:if>
```

The registration can be augmented further with

`resin:IfNetwork`

constraint to limit access to specific IPs only. Adding the constraint must be considered when enabling access over HTTP.

5.3 invoking REST action

Invoking REST action requires constructing an HTTP request with proper url and body. REST action is encoded in the URL. Base of the URL is the address of the Resin server itself combined with the context name of the Resin REST web-app. The defaut URL is

`http://localhost:8080/resin-rest` . Concatenating the URL with action name creates a complete Resin REST service URL. e.g.

`http://localhost:8080/resin-rest/jmx-list`

Example: invoking Resin REST jmx-list action

```
~$curl --user admin:secret http://localhost:8080/resin-rest/jmx-list
```

5.4 interpreting Resin REST Service results

Resin REST replies with JSON Objects encoded as a String. Since integration requires interchangeable data format JSON, with the multitudes of available parsers in various languages, is a proper choice. The output below has been post-formatted and shortened for easy reading.

Example: invoking Resin REST jmx-list action

```
~$curl --user admin:secret
'http://localhost:8080/resin-rest/jmx-list?pattern=resin:type=Resin&print-values= ↩
    true'

[
  {
    "attributes":[
      ...
      {
        "name":"ConfigFile",
```

```
            "value":"/Volumes/projects/caucho/trunk/resin/conf/resin.xml"
          },
          {
            "name":"LogDirectory",
            "value":"/Volumes/projects/caucho/trunk/resin/log"
          },
          {
            "name":"ResinHome",
            "value":"/Volumes/projects/caucho/trunk/resin/"
          },
          {
            "name":"RootDirectory",
            "value":"/Volumes/projects/caucho/trunk/resin/"
          }
        ],
       "name":"resin:type=Resin"
      }
      ...
  ]
```

5.5 available REST actions

Table 5.1: REST actions

ACTION	HTTP METHOD	DESCRIPTION
config-cat	GET	prints configuration file deployed with `config-deploy` action or command
config-deploy	POST	deploys configuration
config-ls	lists files deployed via `config-deploy` command or action	config-undeploy
POST	Un-deploys configuration deployed with config-deploy action	deploy-copy
POST	copies an application from one context to another	deploy-list
GET	lists all applications deployed on a server	jmx-call
POST	calls MBean's method	jmx-dump
GET	dump all MBean attributes and values	jmx-list
GET	lists MBeans, attributes and operations	jmx-set
POST	sets value of a jmx attribute	license-add
POST	adds a license file	list-restarts
GET	lists Resin server restarts	log-level
POST	configures log level for one or more loggers	pdf-report
GET	produces and returns pdf-report	stats
GET	prints stats collected by Resin	thread-dump
GET	produces a thread dump	user-add
POST	adds an user with Resin administrator privileges	user-list
GET	lists users with Resin administrator privileges	user-remove
POST	removes user added with `user-add` command or action	web-app-deploy

Table 5.1: (continued)

ACTION	HTTP METHOD	DESCRIPTION
POST	deploys an application to Resin	web-app-restart
POST	restarts deployed application	web-app-start
POST	starts deployed application	web-app-stop
POST	stops deployed application	web-app-undeploy

5.6 config-deploy

Command config-deploy deploys a configuration archive from an input stream to Resin cluster. Config-deploy action works with Resin's import tag, which imports from a `cloud:/resin-inf/.xml with default configuration (resin.xml)` `&resin:import fileset="cloud:/resin-inf/.xml" recover="true"/>`

Action expects the configuration to be streamed in the POST request body.

```
curl -H "Content-Type: " --user admin:secret --data-binary @/tmp/my-conf.jar 'http ↩
    ://localhost:8080/resin-rest/config-deploy'
```

A value other than "application/x-www-form-urlencoded" must be specified as a value for Content-Type header.

Table 5.2: config-deploy options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to deploy to	
stage	stage of the server's deployment	production
version	version e.g.(1.0.0)	
message	commit message	

5.7 config-cat

Similar to a unix *cat* command config-cat prints content of a configuration file deployed to Resin using config-deploy action.

Table 5.3: config-cat options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	Defaults to current server
name	full name of the configuration file inside the deployed configuration archive	required
stage	stage of the server's deployement	Defaults to *production*

5.8 config-ls

Action config-ls lists files deployed with config-deploy action.

Table 5.4: config-ls options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	Defaults to the current server
name	name of the file or directory	
stage	stage of the server's deployment	production
version	version e.g.(1.0.0)	

5.9 config-undeploy

Action config-undeploy un-deploys configuration deployed with config-deploy action.

Table 5.5: config-undeploy options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to query	
stage	stage of the server's deployment	production
version	version e.g.(1.0.0)	
message	commit message	

5.10 jmx-list

Action jmx-list lists MBeans with (optionally) attributes and values.

Table 5.6: jmx-list options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	Defaults to the current server
pattern	pattern used to query the MBeanServer e.g. java.lang:*	resin:*
print-attributes	prints attributes	false
print-values	prints values (overrides print-attributes)	false
print-operations	prints operations	false
print-all-beans	matches all MBeans	false
print-platform-beans	matches only java.lang:* MBeans	false

5.11 jmx-call

Action jmx-call invokes an action on an MBean.

Table 5.7: jmx-call options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	Defaults to the current server
pattern	JMX ObjectName pattern used to uniquely identify an MBean	required
operation	name of operation (method) to invoke	required
operation-index	when multiple methods are named with the same name, help to choose the method to invoke	0
values	space separated list of arguments	required

5.12 jmx-dump

Action jmx-dump produces a complete dump of JMX tree with attributes and values.

Table 5.8: jmx-dump options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server

5.13 jmx-set

Action jmx-set sets a value on an JMX JBean's attribute. Attribute name must be Capitalized (e.g. Foo not foo)

Table 5.9: jmx-set options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
pattern	JMX ObjectName pattern that uniquely identifies an MBean	required
attribute	name of the attribute	required
value	new value for the attribute	null

5.14 license-add

Action license-add writes a license file into a remote Resin's license directory. The license file should be passed in the request body.

Table 5.10: license-add options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	Defaults to the current server
overwrite	overwrites the file specified in *to* if the file exists	false
to	name of the file to write to	required
restart	optionally restarts server after license is deployed	false

```
curl -H "Content-Type: " --user admin:secret --data-binary @/tmp/my-conf.jar 'http ↩
    ://localhost:8080/resin-rest/config-deploy'
```

A value other than "application/x-www-form-urlencoded" must be specified as a value for Content-Type header.

5.15 list-restarts

Action list-restarts prints restarts since a start of specified period.

Table 5.11: list-restarts options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
period	lookback (from now) period. e.g. *7D*	7 days (specifed as 7D)

list server restarts

```
curl --user admin:secret
'http://localhost:8080/resin-rest/list-restarts'
```

5.16 log-level

Action log-level configures logging level for specified list of loggers.

Table 5.12: log-level options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
loggers	space separated list of loggers	root logger (*') and 'com.caucho* logger
level	new logging level(all, finest, finer, fine, config, info, warning, severe, off)	required
active-time	specifies period of time the new value will be active	until next restart

5.17 pdf-report

Action pdf-report creates a PDF report and optionally returns the PDF file with the Response.

Table 5.13: pdf-report options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	required
report	base name for the report	Snapshot
period	reporting period	1 Day (1D)
log-directory	directory to write the report to on the server side	Resin's log directory
profile-time	turns execution profiler on for a specified time e.g. 20s (20 seconds)	
sample-period	specifies sampling rate e.g. 150ms	100ms
snapshot	includes JMX, thread dump and memory analysis in the report	false
watchdog	specifies period as the starting of the server	false
load-pdf	Streams PDF Report file with the response	false

creating and loading pdf report

```
curl --user admin:secret
'http://localhost:8080/resin-rest/pdf-report?snapshot=true&load-pdf=true' > my- ↩
    report.pdf
```

5.18 stats

Action stats returns statistics that can be used for building graphs. The JSON formatting is compatible with jquery-flot graph solution()

Table 5.14: stats options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	server to query	required
meters	comma separated list of meters e.g.(JVM	Memory
Heap Memory Used,)		period

querying Memory statistics

```
++
curl --user admin:secret  'http://localhost:8080/resin-rest/stats?meters=JVM|Memory|Heap%20 ↩
    Memory%20Used'
```

```
[
{
"label":"00|JVM|Memory|Heap Memory Used",
"data": [
[1330983197106,0.0],
[1330983257158,8.5000192E7],
[1330983299997,8.5000192E7],
[1330983359987,8.5000192E7],
[1330983419984,8.5000192E7],
[1330983479994,8.5000192E7],
[1330983539986,8.5000192E7],
[1330983599990,8.5000192E7],
[1330983659998,8.5000192E7],
[1330983719999,8.5000192E7],
[1330983779994,8.5000192E7],
[1330983839995,8.5000192E7],
[1330983900012,8.5000192E7],
[1330983959983,8.5000192E7],
[1330984019988,8.5000192E7],
[1330984079986,8.5000192E7],
[1330984139984,8.5000192E7],
[1330984199999,8.5000192E7],
[1330984259992,8.5000192E7],
[1330984319991,8.5000192E7],
[1330984379991,8.5000192E7],
[1330984439987,8.5000192E7],
[1330984499994,8.5000192E7],
[1330984559984,8.5000192E7],
[1330984619996,8.5000192E7],
[1330984679996,8.5000192E7],
[1330984739988,8.5000192E7],
[1330984799980,8.5000192E7],
[1330984859999,8.5000192E7],
[1330984919990,8.5000192E7],
[1330984979992,1.07880448E8]
]
}
]
++
```


A complete list of Meters can be looked up in Resin-Admin's graphs section.

5.19 thread-dump

Action thread-dump produces and returns thread dump.

Table 5.15: thread-dump options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server

5.20 user-add

Action user-add adds administrative user.

Table 5.16: user-add options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to query	
user	specifies user name	required
password	specifies password	required

5.21 user-list

Action user-list lists administrative user.

Table 5.17: user-list options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to query	

5.22 user-remove

Action user-remove removes administrative user added with user-add action.

Table 5.18: user-remove options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to query	
user	specifies user name	required

5.23 web-app-deploy

Action web-app-deploy deploys a web application.

Table 5.19: web-app-deploy options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server		triad server to deploy to
context	web application context	required
host	host to deploy to	*default*
stage	state to deploy to	production
version	version e.g. 1.0.0	
message	commit message	

deploying an application via REST

```
curl -H "Content-Type: " --data-binary @/tmp/my-app.war --user admin:secret 'http ↩
    ://localhost:8080/resin-rest/web-app-deploy?context=foo'
```

5.24 deploy-copy

Action deploy-copy is used to copy an application from one context to another. Application copied from source context to a target context. The target context is started upon completion of the operation.

Table 5.20: deploy-copy options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server	defaults to the current server
source-context	context to copy an application from e.g. /foo	required
source-host	host to copy an application from	
source-stage	stage of the source deployment	production
source-version	version of the source deployment	HEAD
target-context	context to copy an application to e.g. /bar	required
target-host	host to copy an application to	
target-stage	stage of the target deployment	
target-version	version of the target deployment	
message	commit message	

5.25 deploy-list

Action deploy-list lists deployed applications.

Table 5.21: deploy-list options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server

5.26 web-app-restart

Action web-app-restart restarts an application.

Table 5.22: web-app-restart options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
tag	tag to restart e.g. (production/webapp/default/foo)	required

Table 5.22: (continued)

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
context	context to restart	required
stage	deployment stage of the context	production
host	deployment host	*default*
version	deployment version	

5.27 web-app-start

Action web-app-start starts web application.

Table 5.23: web-app-start options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
tag	tag to start e.g.(production/webapp/default/foo)	required
context	context to start	required
stage	deployment stage of the context	required

5.28 web-app-stop

Action web-app-stop stops web application.

Table 5.24: web-app-stop options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	cluster server to query	defaults to the current server
tag	tag to start e.g.(production/webapp/default/foo)	required
context	context to stop	required
stage	deployment stage of the context	required

5.29 web-app-undeploy

Action web-app-undeploy un-deploys deployed web application.

Table 5.25: web-app-undeploy options

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
server	triad server to query	

Table 5.25: (continued)

PARAMETER	DESCRIPTION	DEFAULT/REQUIRED
context	context to undeploy	required
host	context deployment host	*default*
stage	context deployment stage	production
version	version	
message	commit message	

Chapter 6

Clustering

6.1 Overview

In Resin, clustering is always enabled. Even if you only have a single server, that server belongs to its own cluster. As you add more servers, the servers are added to the cluster, and automatically gain benefits like clustered health monitoring, heartbeats, distributed management, triple redundancy, and distributed deployment.

Resin clustering gives you: HTTP Load balancing and failover. Dynamic servers: adding and removing servers dynamically. Distributed deployment and versioning of applications. A triple-redundant triad as the reliable cluster hub. Heartbeat monitoring of all servers in the cluster. Health sensors, metering and statistics across the cluster. Clustered JMS queues and topics. Distributed JMX management. Clustered caches and distributed sessions.

To create a cluster, you add servers to the standard configuration files. Because its is built into the Resin's foundations, the servers benefit from clustering automatically.

Example: sample resin.properties for a small cluster

```
...
app_servers : 192.168.1.10:6800 \
              192.168.1.11:6801 \
              192.168.1.12:6802 \
              192.168.1.13:6803
...
```

While most web applications start their life-cycle being deployed to a single server, it eventually becomes necessary to add servers either for performance or for increased reliability. The single-server deployment model is very simple and certainly the one developers are most familiar with since single servers are usually used for development and testing (especially on a developer's local machine). As the usage of a web application grows beyond moderate numbers, the hardware limitations of a single machine simply is not enough (the problem is even more acute when you have more than one high-use application deployed to a single machine). The physical hardware limits of a server usually manifest themselves as chronic high CPU and memory usage despite reasonable performance tuning and hardware upgrades.

Server load-balancing solves the scaling problem by letting you deploy a single web application to more than one physical machine. These machines can then be used to share the web application traffic, reducing the total workload on a single machine and providing better performance from the perspective of the user. We'll discuss Resin's load-balancing capabilities in greater detail shortly, but load-balancing is usually achieved by transparently redirecting network traffic across multiple machines at the systems level via a hardware or software load-balancer (both of which Resin supports). Load-balancing also increases the reliability/up-time of a system because even if one or more servers go down or are brought down for maintenance, other servers can still continue to handle traffic. With a single server application, any down-time is directly visible to the user, drastically decreasing reliability.

If your application runs on multiple servers, Resin's clustered deployment helps you updating new versions of your application to all servers, and verify that all servers in the cluster are running the same version. When you deploy a new version, Resin ensures that all servers have an identical copy, validates the application, and then deploys it consistently. The clustered deployment is

reliable across failures because its built on a transactional version control system (Git). After a new server starts or a failed server restarts, it checks with the redundant triad for the latest application version, copying it as necessary and deploying it.

If web applications were entirely stateless, load-balancing alone would be sufficient in meeting all application server scalability needs. In reality, most web applications are relatively heavily stateful. Even very simple web applications use the HTTP session to keep track of current login, shopping-cart-like functionality and so on. Component oriented web frameworks like JSF and Wicket in particular tend to heavily use the HTTP session to achieve greater development abstraction. Maintaining application state is also very natural for the CDI (Resin CanDI) and Seam programming models with stateful, conversational components. When web applications are load-balanced, application state must somehow be shared across application servers. While in most cases you will likely be using a sticky session (discussed in detail shortly) to pin a session to a single server, sharing state is still important for fail-over . Fail-over is the process of seamlessly transferring over an existing session from one server to another when a load-balanced server fails or is taken down (probably for upgrade). Without fail-over, the user would lose a session when a load-balanced server experiences down-time. In order for fail-over to work, application state must be synchronized or replicated in real time across a set of load-balanced servers. This state synchronization or replication process is generally known as clustering . In a similar vein, the set of synchronized, load-balanced servers are collectively called a cluster . Resin supports robust clustering including persistent sessions, distributed sessions and dynamically adding/removing servers from a cluster (elastic/cloud computing).

Based on the infrastructure required to support load-balancing, fail-over and clustering, Resin also has support for a distributed caching API, clustered application deployment as well as tools for administering the cluster.

6.2 Dynamic Servers

Resin lets you add and remove servers dynamically to the system, and automatically adjusts the cluster to manage those servers. When you add a new server, Resin: Pushes the latest deployed applications to the new server. Replicates distributed caching data if the new server is a triad. Updates the load balancer to send requests to the new server. Creates clients for the JMS queues. Updates the health system to monitor and administer the new server.

6.2.1 Enabling an dynamic servers in the resin.properties

To enable dynamic servers, you can modify the resin.properties with three values: the triad hub servers for the cluster, enabling the elastic flag, and the cluster to join. Configure at least one static server in app_servers . Configuring three static servers is better. Enable dynamic servers with elastic_cloud_enable . Select the cluster to join with home_cluster .

resin.properties to enable dynamic servers

```
...
app_servers          : 192.168.1.10:6800

elastic_cloud_enable : true

home_cluster         : app
...
```

A similar pattern works for other clusters like the *web* cluster. If you're enabling dynamic web servers, configure web_servers for the hub and home_cluster with *web*.

If you use a custom resin.xml configuration, you can add additional clusters following the same pattern. The standard resin.xml shows how the app_servers are defined. If you follow the same pattern, your custom *mytier* cluster can use properties like mytier_servers and home_cluster *mytier*.

6.2.2 Starting an dynamic server

The new server requires a copy of the site's resin.xml and resin.properties. In a cloud environment, you can clone the virtual machine with the properties preconfigured. At least one server must be configured in the hub, because the new server needs the IP address of the hub servers to join the cluster.

The main operation is a "--elastic-server" which forces Resin to create a dynamic server, and "--cluster app-tier" which tells the new server which cluster to join.

Example: minimal command-line for dynamic server

```
unix> resinctl start --elastic-server --cluster app-tier
```

As usual, you can name the server with "--server my-name". By default, a name is generated automatically.

You can omit the --elastic-server and --cluster if you define the home_cluster and if the server is not one of the static servers. (Resin searches for local IP address in the addresses of the static servers if you don't specify a --server. If it finds a match, Resin will start it as a static server.)

After the server shuts down, the cluster will remember it for 15 minutes, giving the server time to restart before removing it from the cluster.

6.3 Deploying Applications to a Cluster

Because each organization has different deployment strategies, Resin provides several deployment techniques to match your internal deployment strategy: Use command-line with "deploy" command. Upload and copy .war files individually. Use /resin-admin deploy form. Use resin-admin-rest.xtp interface.

Resin clustered deployment synchronizes deployment across the cluster These features work much like session clustering in that applications are automatically replicated across the cluster. Because Resin cluster deployment is based on Git, it also allows for application versioning and staging across the cluster.

The upload and copy method of deploying applications to the webapps file is always available. Although the upload and copy of application archives is common for single servers, it becomes more difficult to manage as your site scales to multiple servers. Along with the mechanics of copying the same file many times, you also need to consider the synchronization of applications across all the servers, making sure that each server has a complete, consistent copy. The benefits of using Resin's clustered deployment include: Verify complete upload by secure digest before deployment begins. Verify all servers receive the same copy of the application. Versioned deployment, including transparent user upgrades.

As an example, the command-line deployment of a .war file to a running Resin instance looks like the following:

```
unix> resinctl deploy test.war

Deployed production/webapp/default/test from /home/caucho/ws/test.war
  to http://192.168.1.10:8080/hmtp
```

6.3.1 Remote Deployment

In order to use Resin clustered deployment from a remote system, you must first enable remote deployment on the server, which is disable by default for security. You do this using the following Resin configuration:

resin.properties: Enabling Resin Clustered Deployment

```
remote_cli_enable : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
```

In the example above, both the remote admin service and the deployment service is enabled. Note, the admin authenticator must be enabled for any remote administration and deployment for obvious security reasons. To keep things simple, we used a clear-text password above, but you should likely use a password hash instead.

When deploying remotely, you will need to include the user and password.

```
unix> resinctl deploy --user my-admin --password my-password test.war

Deployed production/webapp/default/test from /home/caucho/ws/test.war
  to http://192.168.1.10:8080/hmtp
```

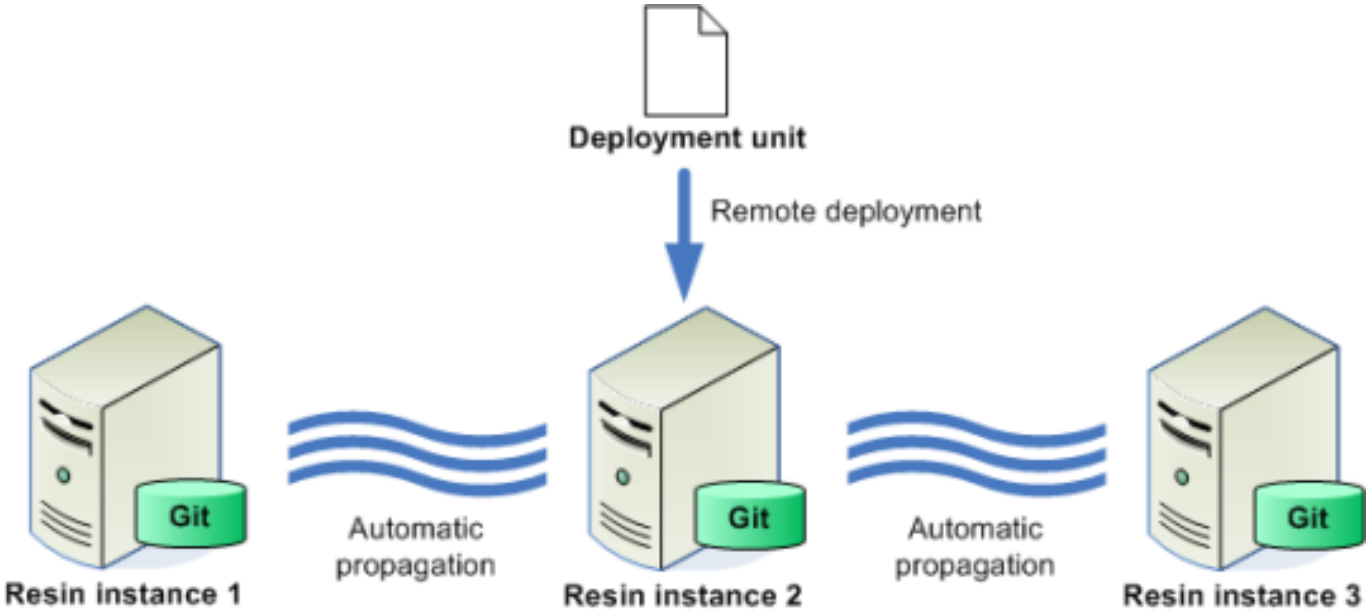
6.3.2 Deployment details and architecture

The output exposes a few important details about the underlying remote deployment implementation for Resin that you should understand. Remote deployment for Resin uses Git under the hood. In case you are not familiar with it, Git is a newish version control system similar to Subversion. A great feature of Git is that it is really clever about avoiding redundancy and synchronizing data across a network. Under the hood, Resin stores deployed files as nodes in Git with tags representing the type of file, development stage, virtual host, web application context root name and version. The format used is this:

```
stage/type/host/webapp[-version]
```

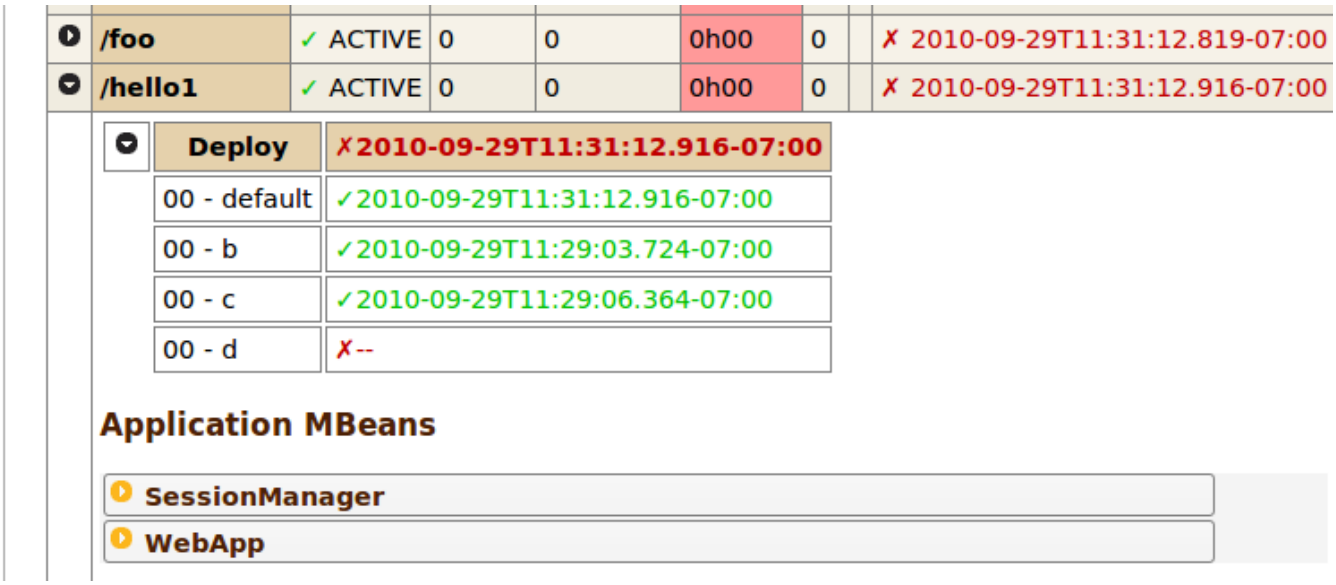
In the example, all web applications are stored under webapp , we didn't specify a stage or virtual host in the Ant task so default is used, the web application root is foo and no version is used since one was not specified. This format is key to the versioning and staging featured we will discuss shortly.

As soon as your web application is uploaded to the Resin deployment repository, it is propagated to all the servers in the cluster - including any dynamic servers that are added to the cluster at a later point in time after initial propagation happens.



When you deploy an application, it's always a good idea to check that all servers in the cluster have received the correct web-app. Because Resin's deployment is based on a transactional version-control system, you can compare the exact version across all servers in the cluster using the /resin-admin web-app page.

The following screenshot shows the /hello1 webapp deployed across the cluster using a .war deployment in the webapps/ directory. Each version (date) is in green to indicate that the hash validation matches for each server. If one server had a different content, the date would be marked with a red *x*.

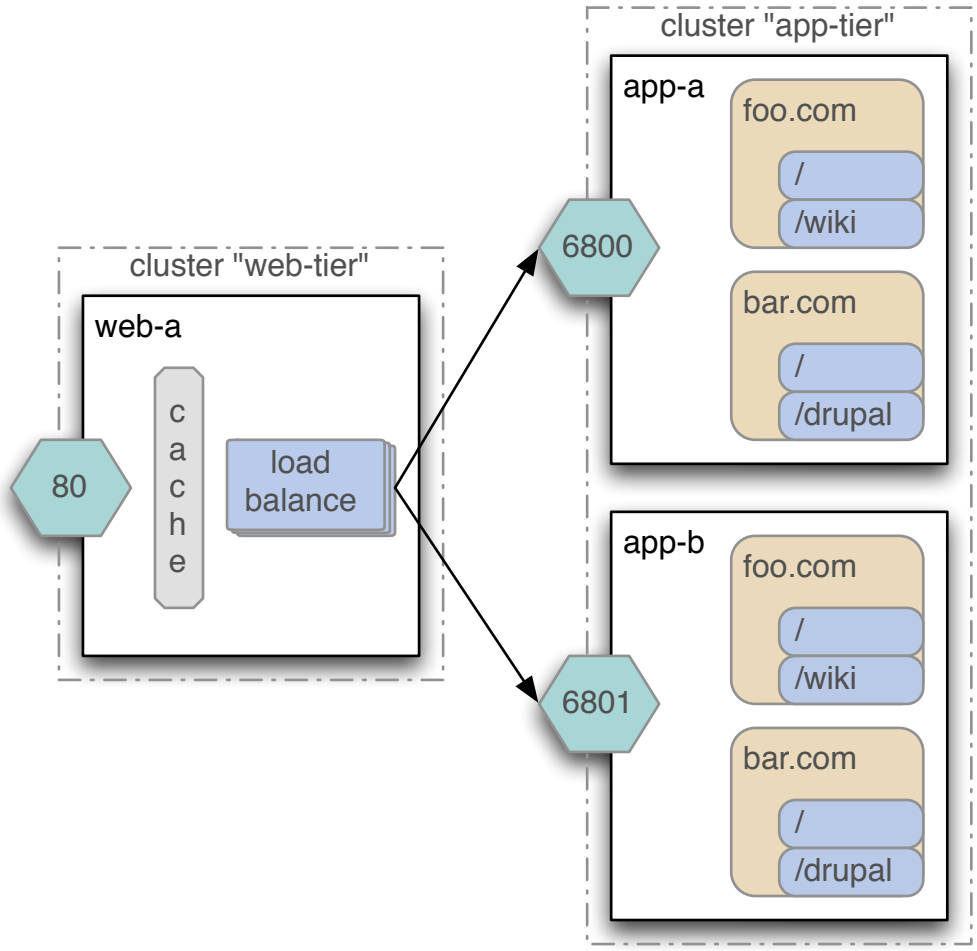


The details for each Ant and Maven based clustered deployment API is outlined clustering-deployment-ref.xtp .

6.4 HTTP Load Balancing and Failover

Once your traffic increases beyond the capacity of a single application server, your site needs to add a HTTP load balancer as well as the second application server. Your system is now a two tier system: an app-tier with your program and a web-tier for HTTP load-balancing and HTTP proxy caching. Since the load-balancing web-tier is focusing on HTTP and proxy caching, you can usually handle many app-tier servers with a single web-tier load balancer. Although the largest sites will pay for an expensive hardware load balancer, medium and small sites can use Resin's built-in load-balancer as a cost-effective and efficient way to scale a moderate number of servers, while gaining the performance advantage of Resin's HTTP proxy caching.

A load balanced system has three main requirements: define the servers to use as the application-tier, define the servers in the web-tier, and select which requests are forwarded (proxied) to the backend app-tier servers. Since Resin's clustering configuration already defines the servers in a cluster, the only additional configuration is to use the <resin:LoadBalance> tag to forward request to the backend.



6.4.1 resin.properties load balancing configuration

For load balancing with the standard resin.xml and resin.properties, the web_servers defines the static servers for the web (load-balancing) tier, and the app_servers defines the static servers for the application (backend) tier. The standard resin.xml configuration will proxy requests from the web tier to the app tier.

resin.properties: load balancing

```
web_servers : 192.168.1.20:6820
web.http : 80

app_servers : 192.168.1.10:6820 \
              192.168.1.11:6820 \
              192.168.1.22:6820
app.http : 8080
```

6.4.2 resin.xml load balancing configuration

Dispatching requests with load balancing in Resin is accomplished through the http-rewrite-ref.xtp tag placed on a cluster. In effect, the http-rewrite-ref.xtp tag turns a set of servers in a cluster into a software load-balancer. This makes a lot of sense in light of the fact that as the traffic on your application requires multiple servers, your site will naturally be split into two tiers: an application server tier for running your web applications and a web/HTTP server tier talking to end-point browsers, caching static/non-static content, and distributing load across servers in the application server tier.

The best way to understand how this works is through a simple example configuration. The following resin.xml configuration shows servers split into two clusters: a web-tier for load balancing and HTTP, and an app-tier to process the application:

Example: resin.xml for Load-Balancing

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
    <resin:import path="${__DIR__}/app-default.xml"/>
  </cluster-default>

  <cluster id="app-tier">
    <server id="app-a" address="192.168.0.10" port="6800"/>
    <server id="app-b" address="192.168.0.11" port="6800"/>

    <host id="">
      <web-app id="" root-directory="/var/resin/htdocs"/>
    </host>
  </cluster>

  <cluster id="web-tier">
    <server id="web-a" address="192.168.0.1" port="6800">
      <http port="80"/>
    </server>

    <proxy-cache memory-size="256M"/>

    <host id="">

      <resin:LoadBalance regexp="" cluster="app-tier"/>

    </host>
  </cluster>

</resin>
```

In the configuration above, the web-tier cluster server load balances across the app-tier cluster servers because of the cluster attribute specified in the <resin:LoadBalance> tag. The <cache> tag enables proxy caching at the web tier. The web-tier forwards requests evenly and skips any app-tier server that's down for maintenance/upgrade or restarting due to a crash. The load balancer also steers traffic from a single session to the same app-tier server (a.k.a sticky sessions), improving caching and session performance.

Each app-tier server produces the same application because they have the same virtual hosts, web-applications and Servlets, and use the same resin.xml. Adding a new machine just requires adding a new <server> tag to the cluster. Each server has a unique name like "app-b" and a TCP cluster-port consisting of an <IP,port>, so the other servers can communicate with it. Although you can start multiple Resin servers on the same machine, TCP requires the <IP,port> to be unique, so you might need to assign unique ports like 6801, 6802 for servers on the same machine. On different machines, you'll use unique IP addresses. Because the cluster-port is for Resin servers to communicate with each other, they'll typically be private IP addresses like 192.168.1.10 and not public IP addresses. In particular, the load balancer on the web-tier uses the cluster-port of the app-tier to forward HTTP requests.

When no explicit server is specified, Resin will search for local IP addresses that match the configured servers and start all of them. This local IP searching lets Resin's service start script work without modification. When dynamic servers are enabled, and no local server matches, Resin will start a new dynamic server to join the home_cluster cluster.

Starting Local Servers

```
unix> resinctl start-all
```

All three servers will use the same resin.xml, which makes managing multiple server configurations pretty easy. The servers are named by the server id attribute, which must be unique, just as the <IP,port>. When you start a Resin instance, you'll use the server-id as part of the command line:

Starting Explicit Servers

```
192.168.0.10> resinctl -server app-a start
192.168.0.11> resinctl -server app-b start
192.168.0.1> resinctl -server web-a start
```

Since Resin lets you start multiple servers on the same machine, a small site might start the web-tier server and one of the app-tier servers on one machine, and start the second server on a second machine. You can even start all three servers on the same machine, increasing reliability and easing maintenance, without addressing the additional load (although it will still be problematic if the physical machine itself and not just the JVM crashes). If you do put multiple servers on the same machine, remember to change the port to something like 6801, etc so the TCP binds do not conflict.

In the resin-admin.xtp management page, you can manage all three servers at once, gathering statistics/load and watching for any errors. When setting up /resin-admin on a web-tier server, you'll want to remember to add a separate <web-app> for resin-admin to make sure the <rewrite-dispatch> doesn't inadvertantly send the management request to the app-tier.

6.4.3 Sticky/Persistent Sessions

To understand what sticky sessions are and why they are important, it is easiest to see what will happen without them. Let us take our previous example and assume that the web tier distributes sessions across the application tier in a totally random fashion. Recall that while Resin can replicate the HTTP session across the cluster, it does not do so by default. Now lets assume that the first request to the web application resolves to app-a and results in the login being stored in the session. If the load-balancer distributes sessions randomly, there is no guarantee that the next request will come to app-a. If it goes to app-b, that server instance will have no knowledge of the login and will start a fresh session, forcing the user to login again. The same issue would be repeated as the user continues to use the application from in what their view should be a single session with well-defined state, making for a pretty confusing experience with application state randomly getting lost!

One way to solve this issue would be to fully synchronize the session across the load-balanced servers. Resin does support this through its clustering features (which is discussed in detail in the following sections). The problem is that the cost of continuously doing this synchronization across the entire cluster is very high and relatively unnecessary. This is where sticky sessions come in. With sticky sessions, the load-balancer makes sure that once a session is started, any subsequent requests from the client go to the same server where the session resides. By default, the Resin load-balancer enforces sticky sessions, so you don't really need to do anything to enable it.

Resin accomplishes this by encoding the session cookie with the host name that started the session. Using our example, the hosts would generate cookies like this:

INDEX	COOKIE PREFIX
1	a xxx
2	b xxx

On the web-tier, Resin decoded the session cookie and sends it to the appropriate server. So baaX8ZwooOz will go to app-b. If app-b fails or is down for maintenance, Resin will send the request a backup server calculated from the session id, in this case app-a. Because the session is not clustered, the user will lose the session but they won't see a connection failure (to see how to avoid losing the session, check out the following section on clustering).

6.4.4 Manually Choosing a Server

For testing purposes you might want to send requests to a specific servers in the app-tier manually. You can easily do this since the web-tier uses the value of the jsessionid to maintain sticky sessions. You can include an explicit jsessionid to force the web-tier to use a particular server in the app-tier.

Since Resin uses the first character of the jsessionid to identify the server to use, starting with *a* will resolve the request to app-a. If www.example.com resolves to your web-tier, then you can use values like this for testing: http://www.example.com/-test-servlet;jsessionid=aaaXXXXX http://www.example.com/test-servlet;jsessionid=baaXXXXX http://www.example.com/test-servlet;jsessionid=caaXXXXX http://www.example.com/test-servlet;jsessionid=daaXXXXX http://www.example.com/test-servlet;jsess
etc.

You can also use this fact to configure an external sticky load-balancer (likely a high-performance hardware load-balancer) and eliminate the web tier altogether. In this case, this is how the Resin configuration might look like:

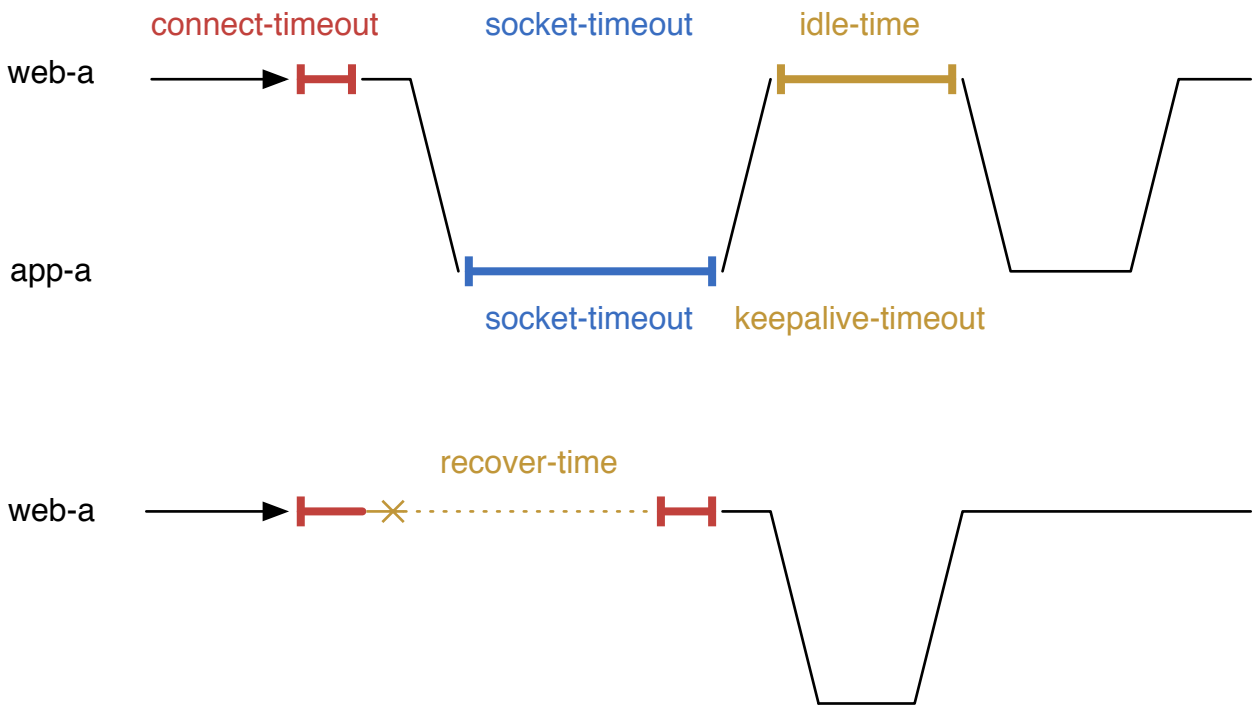
resin.xml with Hardware Load-Balancer

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <server-default>
      <http port='80'/>
    </server-default>

    <server id='app-a' address='192.168.0.1' port="6800"/>
    <server id='app-b' address='192.168.0.2' port="6800"/>
    <server id='app-c' address='192.168.0.3' port="6800"/>

  </cluster>
</resin>
```

Each server will be started as -server a , -server b , etc to grab its specific configuration.

6.4.5 Socket Pooling, Timeouts, and Failover

For efficiency, Resin's load balancer manages a pool of sockets connecting to the app-tier servers. If Resin forwards a new request to an app-tier server and it has an idle socket available in the pool, it will reuse that socket, improving performance and minimizing network load. Resin uses a set of timeout values to manage the socket pool and to handle any failures or freezes of the backend servers. The following diagram illustrates the main timeout values:



load-balance-connect-timeout : the load balancer timeout for the `connect()` system call to complete to the app-tier (5s). load-balance-idle-time : load balancer timeout for an idle socket before closing it automatically (5s). load-balance-recover-time : the load balancer connection failure wait time before trying a new connection (15s). load-balance-socket-timeout : the load balancer timeout for a valid request to complete (665s). keepalive-timeout : the app-tier timeout for a keepalive connection (15s) socket-timeout : the app-tier timeout for a read or write (65s)

When an app-tier server is down due to maintenance or a crash, Resin will use the load-balance-recover-time as a delay before retrying the downed server. With the failover and recover timeout, the load balancer reduces the cost of a failed server to almost

no time at all. Every recover-time, Resin will try a new connection and wait for load-balance-connect-timeout for the server to respond. At most, one request every 15 seconds might wait an extra 5 seconds to connect to the backend server. All other requests will automatically go to the other servers.

The socket-timeout values tell Resin when a socket connection is dead and should be dropped. The web-tier timeout load-balance-socket-timeout is much larger than the app-tier timeout socket-timeout because the web-tier needs to wait for the application to generate the response. If your application has some very slow pages, like a complicated nightly report, you may need to increase the load-balance-socket-timeout to avoid the web-tier disconnecting it.

Likewise, the load-balance-idle-time and keepalive-timeout are a matching pair for the socket idle pool. The idle-time tells the web-tier how long it can keep an idle socket before closing it. The keepalive-timeout tells the app-tier how long it should listen for new requests on the socket. The keepalive-timeout must be significantly larger than the load-balance-idle-time so the app-tier doesn't close its sockets too soon. The keepalive timeout can be large since the app-tier can use the http-server-ref.xtp manager to efficiently wait for many connections at once.

6.4.6 resin:LoadBalance Dispatching

<resin:LoadBalance> is part of Resin's http-rewrite.xtp capabilities, Resin's equivalent of the Apache mod_rewrite module, providing powerful and detailed URL matching and decoding. More sophisticated sites might load-balance based on the virtual host or URL using multiple <resin:LoadBalance> tags.

In most cases, the web-tier will dispatch everything to the app-tier servers. Because of Resin's http-proxy-cache.xtp , the web-tier servers will serve static pages as fast as if they were local pages.

In some cases, though, it may be important to send different requests to different backend clusters. The http://caucho.com/-resin-javadoc/com/caucho/rewrite/LoadBalance.html tag can choose clusters based on URL patterns when such capabilities are needed.

The following http-rewrite.xtp example keeps all **.png, *.gif, and *.jpg files on the web-tier, sends everything in /foo/** to the foo-tier cluster, everything in /bar/* to the bar-tier cluster, and keeps anything else on the web-tier.

Example: resin.xml Split Dispatching

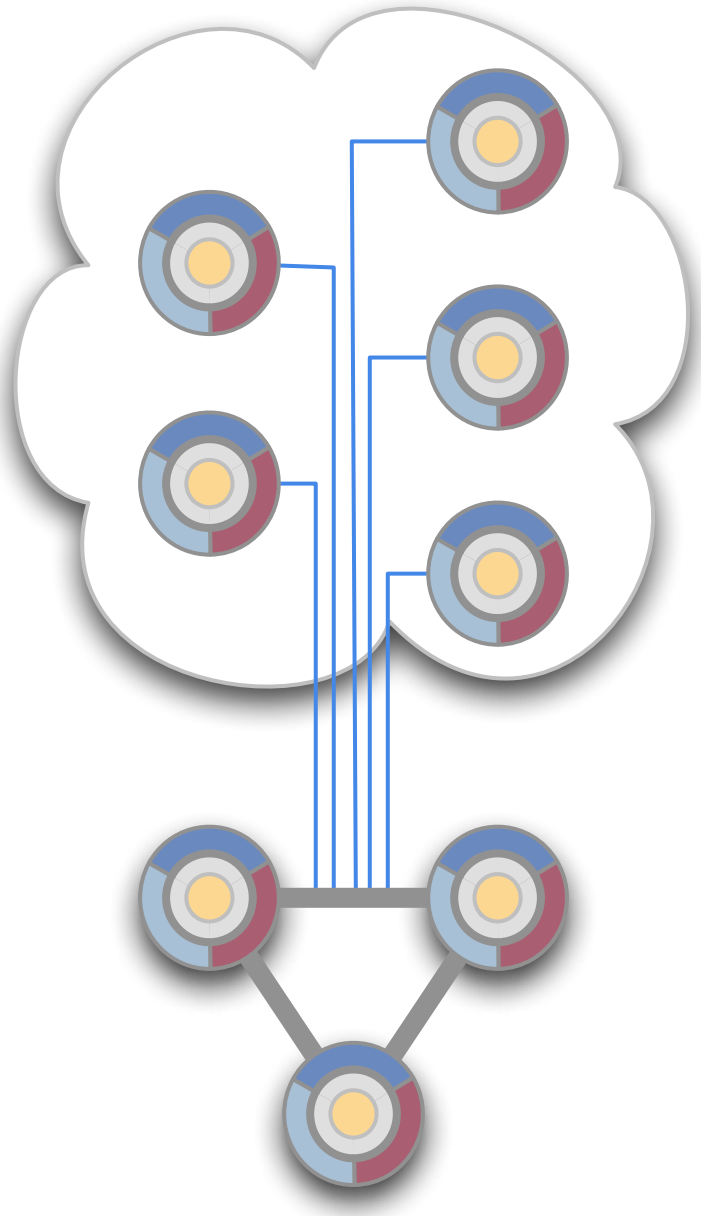
```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
     <resin:import path="classpath:META-INF/caucho/app-default.xml"/>
  </cluster-default>

  <cluster id="web-tier">
    <server id="web-a">
      <http port="80"/>
    </server>

    <proxy-cache memory-size="64m"/>

    <host id="">
      <web-app id="/">

        <resin:Dispatch regexp="(\.png|\.gif|\.jpg)"/>

        <resin:LoadBalance regexp="^/foo" cluster="foo-tier"/>

        <resin:LoadBalance regexp="^/bar" cluster="bar-tier"/>

      </web-app>
    </host>
  </cluster>

  <cluster id="foo-tier">
    ...
```

```
  </cluster>

  <cluster id="bar-tier">
    ...
  </cluster>
</resin>
```

For details on the tags used for clustering, please refer to clustering-ref.xtp .

6.5 Triple Redundant Clustering (the Triad)

After many years of developing Resin's distributed clustering for a wide variety of user configurations, we refined our clustering network to a hub-and-spoke model using a triple-redundant triad of servers as the cluster hub. The triad model provides interlocking benefits for the dynamically scaling server configurations people are using. When you bring down one server for maintenance, the triple redundancy continues to maintain reliability. You can add and remove dynamic servers safely without affecting the redundant state. Your network processing will be load-balanced across three servers, eliminating a single point of failure. Your servers will scale evenly across the cluster pod because each new server only needs to speak to the triad, not all servers. Your large site can split the cluster into independent "pods" of less than 64-servers to maintain scalability.



6.5.1 Cluster Heartbeat

As part of Resin's health check system, the cluster continually checks that all servers are alive and responding properly. Every 60 seconds, each server sends a heartbeat to the triad, and each triad server sends a heartbeat to all the other servers.

When a server failure is detected, Resin immediately detects the failure, logging it for an administrator's analysis and internally prepares to failover to backup servers for any messaging for distributed storage like the clustered sessions and the clustered deployment.

When the server comes back up, the heartbeat is reestablished and any missing data is recovered.

6.6 Clustered Sessions

Because load balancing, maintenance and failover should be invisible to your users, Resin can replicate user's session data across the cluster. When the load-balancer fails over a request to a backup server, or when you dynamically remove a server, the backup can grab the session and continue processing. From the user's perspective, nothing has changed. To make this process fast and reliable, Resin uses the triad servers as a triplicate backup for the user's session.

The following is a simple example of how to enable session clustering for all web-apps using the standard resin.properties:

Example: resin.properties enabling clustered sessions

```
...
session_store : true
...
```

If you want to configure persistent sessions for a specific web-app, you can set the use-persistent-store attribute of the <session-config>.

Example: resin-web.xml enabling clustered sessions

```
<web-app xmlns="http://caucho.com/ns/resin">
  <session-config>
    <use-persistent-store="true"/>
  </session-config>
</web-app>
```

The deploy-ref.xtp#session-config tag under deploy-ref.xtp#session-config is used to enable clustering. Note, clustering is enabled on a per-web-application basis because not all web applications under a host need be clustered. If you want to cluster all web applications under a host, you can place <use-persistent-store> under deploy-ref.xtp#web-app-default . The following example shows how you can do this:

Example: Clustering for All Web Applications

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster>
    <server id="a" address="192.168.0.1" port="6800"/>
    <server id="b" address="192.168.0.2" port="6800"/>

    <web-app-default>
      <session-config use-persistent-store="true"/>
    </web-app-default>
  </cluster>
</resin>
```

The above example also shows how you can override the clustering behaviour for Resin. By default, Resin uses a triad based replication with all three triad servers backing up the data server. THe <persistent-store type="cluster"> has a number of other attributes:

Table 6.1: cluster store attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
always-save	Always save the value	false
max-idle-time	How long idle objects are stored (session-timeout will invalidate items earlier)	24h

The cluster store is valuable for single-server configurations, because Resin stores a copy of the session data on disk, allowing for recovery after system restart. This is basically identical to the file persistence feature discussed below.

6.6.1 always-save-session

Resin's distributed sessions need to know when a session has changed in order to save/synchronize the new session value. Although Resin does detect when an application calls HttpSession.setAttribute , it can't tell if an internal session value has changed. The following Counter class shows the issue:

Counter.java

```
package test;

public class Counter implements java.io.Serializable {
  private int \_count;

  public int nextCount() { return \_count++; }
}
```

Assuming a copy of the Counter is saved as a session attribute, Resin doesn't know if the application has called nextCount . If it can't detect a change, Resin will not backup/synchronize the new session, unless always-save-session is set on the <session-config>. When always-save-session is true, Resin will back up the entire session at the end of every request. Otherwise, a session is only changed when a change is detected.

```
...
<web-app id="/foo">
...
<session-config>
  <use-persistent-store/>
  <always-save-session/>
</session-config>
...
</web-app>
```

6.6.2 Serialization

Resin's distributed sessions relies on Hessian serialization to save and restore sessions. Application objects must implement java.io.Serializable for distributed sessions to work.

6.6.3 No Distributed Locking

Resin's clustering does not lock sessions for replication. For browser-based sessions, only one request will typically execute at a time. Because browser sessions have no concurrency, there's really no need for distributed locking. However, it's a good idea to be aware of the lack of distributed locking in Resin clustering.

For details on the tags used for clustering, please refer to clustering-ref.xtp .

6.7 Distributed Caching

Distributed caching is a useful technique to reduce database load and increase application performance. Resin provides a JSR-107 JCache based distributed caching API. Distributed caching in Resin uses the underlying infrastructure used to support load balancing, clustering and session/state replication.

The first step to using Resin distributed caching is to configure a cache. You configure a named distributed cache at the web application level like this:

resin-web.xml: Configuring Resin Distributed Cache

```
<web-app xmlns="http://caucho.com/ns/resin"
          xmlns:distcache="urn:java:com.caucho.distcache">

  <distcache:ClusterCache>
    <name>test</name>
  </distcache:ClusterCache>

</web-app>
```

You may then inject the cache through CDI and use it in your application via the JCache API:

Using Resin Distributed Cache

```
import javax.cache.*;
...
@Inject
private Cache \_cache;
...

cache.putIfAbsent("discount_rate", discountRate);
```

Chapter 7

Clustering: Server Config

7.1 Overview

The reference for the <server> tag and its schema are at reference.xtp#server . Core configuration: the server's id, local TCP address and port which configure it in the cluster. server-default: shared configuration for all servers belongs in a server-default. cluster/cloud the server configuration in the cluster block defines the cluster. JVM: JVM command-line parameters like -Xmx. HTTP and TCP ports: configuring the server's HTTP ports. Thread Pool: thread pool size and idle thread pool size. Load balancing: timeouts and weights for the load balancer. Watchdog: configuration for the server's watchdog.

7.2 Core Configuration

A cluster server has an "id", a local network "address" and a local network "port". For a standalone server, the address and port can be omitted, but the server cannot participate in a cluster or be managed from a command line.

The following example shows the minimal server for a two-server cluster. These servers do not listen to a HTTP port, so they are either a backend tier with a load-balancer or they are a non-HTTP use of Resin (like a SOA service.)

Example: minimal server in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">

  <server id="a" address="192.168.1.10" port="6800"/>
  <server id="b" address="192.168.1.11" port="6800"/>

  ...

</cluster>
</resin>
```

7.3 Clustering: servers in a cluster

Each <server> becomes part of a cluster, the <cluster> that contains the server tag. Because each server has a local-network address and port, the servers can automatically communicate with each other. Essentially, there's no extra configuration needed for Resin clustering.

A Resin cluster uses a triple-redundant hub called the triad to store shared data and to coordinate cluster services like caching and JMS queues. If you have two servers, those servers will form the hub. The triple redundancy is important for reliability:

the cluster can survive two triad servers going down. The hub-and-spoke topology is important for elastic cloud configurations: adding and removing servers just removes spokes without affecting the hub.

Typically, a site with many servers will configure at least the first three in the resin.xml, and use Resin's elastic-server capability for the remaining servers. Keeping the first three in the resin.xml ensures that all servers can connect to at least one of the triad even if one is down for maintenance or a restart.

Example: triad configuration in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">

  <server id="a" address="192.168.1.10" port="6800"/>
  <server id="b" address="192.168.1.11" port="6800"/>
  <server id="c" address="192.168.1.12" port="6800"/>

  <dynamic-server-enable/>
  <resin:DynamicCloudService/>

  ...

</cluster>
</resin>
```

7.3.1 dynamic servers and --cluster

For more details see the clustering-overview.xtp .

Elastic servers are added to a cluster using the command-line --cluster option. The new server is assigned a server id from the command line, an IP address from the local network. The new server contacts the triad defined in the resin.xml and requests to be added. The triad updates the cluster topology and informs all servers about the new server.

Example: command-line for dynamic server

```
unix> bin/resin.sh --cluster app-tier start \
          -user my-admin -password my-password
```

The new server will use configuration from the <server-default> in the cluster it's joining. It will automatically deploy applications which were deployed in the cluster, and will automatically join clustered caches, JMS queues, and load balancing.

When the server shuts down, the triad will hold its topology spot open for 15 minutes to allow for restarts. After the timeout, the triad will remove the server.

7.4 server-default

Most sites use a shared <server-default> to configure servers because parameters for a cluster's servers are usually identical. The HTTP ports, timeouts, keepalives, JVM and thread configurations can be shared easily. Servers that do need different configuration (like a staging server) can override the configuration in the <server> tag.

For example, the following defines a common set of JVM and HTTP configuration.

Example: basic http configuration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="app">

  <server-default>
    <jvm-arg-line>-Xmx2048m</jvm-arg-line>
```

```
    <accept-thread-min>32</accept-thread-max>
    <accept-thread-min>64</accept-thread-max>

    <port-thread-max>256</port-thread-max>

    <http port="80"/>
  </server-default>

  <server id="a" address="192.168.1.10" port="6800"/>
  <server id="b" address="192.168.1.11" port="6800"/>
  ...
```

7.5 HTTP and TCP ports: managing sockets and timeouts

HTTP ports are typically configured in a <server-default> tag because a Resin servers in a cluster will typically all listen to the same HTTP port. If each server listens to a different HTTP port, the <http> tag is in the individual <server> block instead of the <server-default>.

The following example is a typical configuration for a Resin cluster where each server listens to the same HTTP port.

Example: basic http configuration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <http port="80"/>
  </server-default>

  <server id="a" address="192.168.1.10" port="6800"/>
  ...
```

7.5.1 Accept thread and connection pools

Each HTTP port has a thread pool which listens for new requests and processes them. The thread that accepts a HTTP connection will also process the request, and a new thread will be created to take its place listening for new requests. Since this HTTP thread pool is a child of Resin's main thread pool, the Resin threads limits and management automatically apply.

port-thread-max is the maximum thread count for handling a port's requests. If more requests arrive than threads, the requests will be queued until a thread becomes available. You can use port-thread-max to limit the maximum load on a server.

accept-thread-min sets the minimum number of threads which are concurrently accepting a new connection. If the thread count drops below accept-thread-min, Resin will spawn a new thread to accept() the connection. If you look at a thread dump, there will always be at least this many threads in the accept() method. A larger value for accept-thread-min improves load-spike handling.

accept-thread-max sets the maximum number of threads which are concurrently accepting a new connection. When a request completes, it normally tries to accept a new connection, but when accept-thread-max threads are already listening, the thread will exit instead. A larger gap between accept-thread-min and accept-thread-max reduces thread churning.

connection-max limits the total requests/connections allowed at any time. Typically this will be very large because most sites will never need to throttle the connections.

throttle-concurrent-max limits the concurrent requests from a single IP address. This limit can help with denial-of-service attacks.

The following example show a typical configuration for the HTTP accept pool. The example tags are in the <server-default> instead of the <http> for convenience, since both <http> ports 80 and 81 need the same configuration. If the two <http> ports needed different values, you would put the <accept-thread-min> tag inside the <http> port itself.

Example: configuring the accept thread pool

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <accept-thread-min>32<accept-thread-min> <accept-thread-max>64<accept-thread-max>

    <http port="80"/>
    <http port="81"/>
  </server-default>

  ...
```

7.5.2 HTTP socket timeouts and configuration

socket-timeout configures the read and write timeout for the socket to protect against network hangs and denial-of-service attacks. When Resin reads a HTTP request or POST data, it uses the socket-timeout to limit how long it will wait. Resin also uses socket-timeout on the response writes to disconnect from frozen clients.

The socket-timeout must be at least 65s because of browser requirements.

If you have a long-polling application, you will probably need to increase the default socket-timeout to your long-polling time.

Example: socket-timeout for a HTTP port

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>
    <socket-timeout>120s<socket-timeout>

    <http port="80"/>
    <http port="81"/>
  </server-default>

  ...
```

7.5.3 Keepalive configuration

HTTP keepalives improve performance by letting browsers reuse the same TCP socket for new requests, for example images and javascript files from a main page. You can balance the added efficiency against the extra sockets and threads in the <server> and <http> keepalive configuration.

To handle many keepalive sockets with fewer threads, Resin Pro has a "keepalive-select" manager which detaches the thread from its connection and reuses the thread. The keepalive-select manager allows for tens of thousands of connections as many as your operating system will allow file descriptors.

keepalive-timeout will close the socket if the browser waits too long before sending a new request. Because requests are bursty, even a small value like 10s can improve performance.

keepalive-max defines the maximum connections waiting for keepalive requests. With the keepalive-select manager or a 64-bit system, this can be a large number. It's generally better to limit the keepalives using keepalive-timeout instead of keepalive-max.

keepalive-select-enable lets you disable the keepalive-select manager. Because modern 64-bit operating systems handle threads efficiently, it can be more efficient to use the connection thread to wait for a new request. However, the keepalive-select-thread-timeout may be a better choice in that situation.

keepalive-select-thread-timeout sets a short thread-based keepalive timeout waiting for the next request before going to the select manager. Because keepalive requests are bursty and thread-based keepalive is faster, Resin waits for a short time. You can increase the performance with a slight increased thread count by increasing keepalive-select-thread-timeout .

7.5.4 Openssl and Jsse

For more information see the security-ssl.xtp .

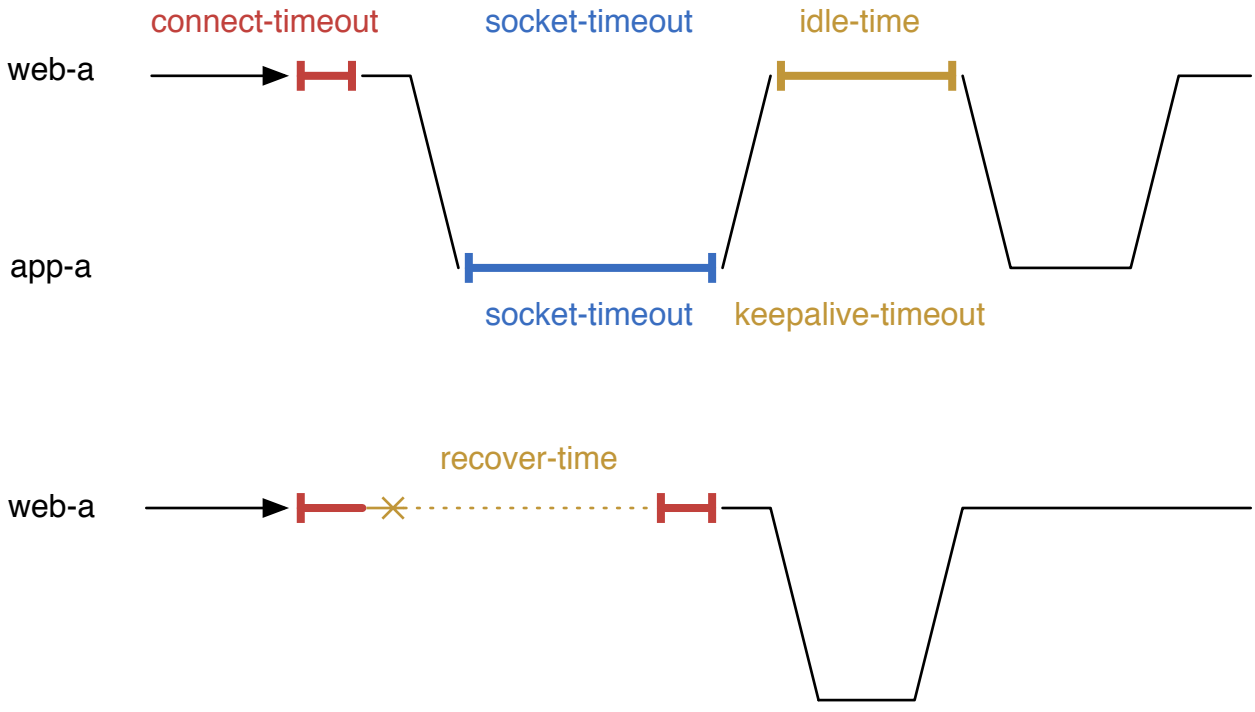
HTTP ports can use either OpenSSL (faster) or JSSE to support SSL ports. The SSL configuration tags belong inside the <http> tag.

Example: OpenSSL HTTP configuration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server-default>

    <http port="443">
      <openssl>
        <certificate-file>keys/mycert.crt</certificate-file>
        <certificate-key-file>keys/mycert.key</certificate-key-file>
        <password>mypassword</password>
      </openssl>
    </http>

  </server-default>

  ...
```

7.6 Load Balancing: Socket Pooling, Timeouts, and Failover

For efficiency, Resin's load balancer manages a pool of sockets connecting to the app-tier servers. If Resin forwards a new request to an app-tier server and it has an idle socket available, it will reuse that socket, improving performance an minimizing network load. Resin uses a set of timeout values to manage those idle sockets and to handle any failures or freezes of the backend servers. The following diagram illustrates the main timeout values:



load-balance-connect-timeout : the load balancer timeout for the `connect()` system call to complete to the app-tier (5s). load-balance-idle-time : load balancer timeout for an idle socket before closing it automatically (5s). load-balance-recover-time :

the load balancer connection failure wait time before trying a new connection (15s). load-balance-socket-timeout : the load balancer timeout for a valid request to complete (665s). keepalive-timeout : the app-tier timeout for a keepalive connection (15s) socket-timeout : the app-tier timeout for a read or write (65s)

7.7 JVM parameters: setting the JVM command line

JVM command-line parameters are configured in the <server> tag. When launching Resin, the watchdog process will read the <server> tag and add the JVM argument to the launched command-line.

Because most servers in a cluster are identical, you'll generally put the JVM configuration in a <server-default>. If you have some servers with different requirements, you can configure the JVM in <server>.

Example: -Xmx2048m in server-default

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">

  <server-default>
    <jvm-arg>-Xmx2048m</jvm-arg>
  </server-default>

  ...

</cluster>
</resin>
```

7.7.1 setuid: user-name and group-name

Because Unix requires root access to bind to port 80 and 443, Resin can change from root (the watchdog) to a specific user (the Resin server). The user-name and group-name tags in the <server> set the user to run as.

Example: user-name and group-name

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">

  <server-default>
    <user-name>resin</user-name>
    <group-name>resin</group-name>
  </server-default>

  ...

</cluster>
</resin>
```

Chapter 8

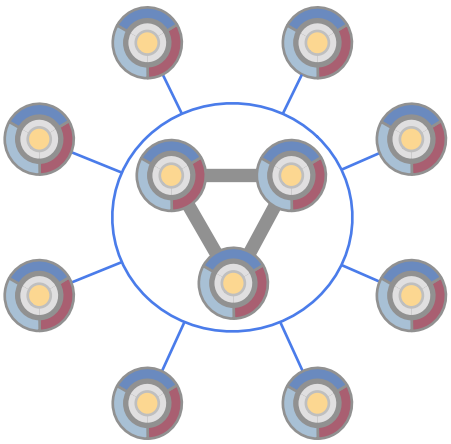
Clustering: Dynamic Server (Cloud)

8.1 Overview

Resin's cloud support is an extension of its clustering. It is the third generation of Resin clustering, designed for elastic clusters: adding and removing servers from a live cluster.

A triple-redundant hub-and-spoke network is the heart of a Resin cluster. The three servers that form the hub are called the *triad* and are responsible for reliability and for load-balancing clustered services like caching. All other servers can be added or removed without affecting the shared data.

Services like clustered deployment, caching, JMS, and load-balancing recognize the new server and automatically add it to the service. When you delete a server, the services will adapt to the smaller network.



The triad hub-and-spoke model solves several clustering issues we've discovered over the years. The three primary issues are the need for triple redundancy, managing dynamic servers (particularly removing servers), and giving an understandable user model for persistent storage.

Triple redundancy is needed because of server maintenance and load-balancing. When you take down a server for scheduled maintenance, the remaining servers have extra load and they are more vulnerable to a server failure. A triple-redundant hub with one server down for maintenance can survive a crash of one of the two remaining servers. The load is less, too. When one server is down for maintenance, the other two servers share an extra 50% load each. With only a backup system, the extra server would double its load.

Dynamic servers benefit from the hub-and-spoke model because the *spoke* servers are not needed for reliability. Because they don't store the primary cache values, the primary application deployment repository, or the queue store, it's possible to add and remove them without affecting the primary system. With other network configurations, removing a server forces a shuffling around backup data to the remaining servers.

The triad also gives a simpler user model for understanding where things are stored: data is stored in the triad. Other servers just use the data. Understanding the triad means you can feel confident about removing non-triad servers, and also know that the three triad servers are worth additional reliability attention.

8.2 Baseline: three static server (triad) with resin.properties

The standard /etc/resin.properties configuration lets you configure a triad and dynamic servers without a few properties.

To configure three static servers for the triad hub, enable dynamic servers, and select the "app" cluster as the default dynamic server, use csomething like the following in your /etc/resin.properties.

Example: resin.properties for a 3-server hub

```
...
# app-tier Triad servers: app-0 app-1 app-2
app_servers      : 192.168.1.10:6800 192.168.1.11:6800 192.168.1.12:6800
...
# Allow elastic nodes to join the cluster (enable for cloud mode)
elastic_cloud_enable : true

# The cluster that elastic nodes should join - each will contact a Triad server
# Use a separate resin.properties file for each cluster
home_cluster : app
```

app_servers: each IP:port in the app-servers adds a new static server. elastic_cloud_enable: enable dynamic servers. home_cluster: select the home cluster for the dynamic server.

To start the server, you can use the "start-all" command. The start-all with start all local servers (by comparing the IP to the IP addresses of the current machine.) If no local servers are found, start-all will start a dynamic server using the <home-cluster>.

CLI: starting the servers

```
unix> resinctl start-all
```

If you've installed Resin as a Unix service, it will be started automatically when the server starts. You can either use the /etc/init.d resin command or the "service" if it's available. The service start is equivalent to a resinctl "start-all".

CLI: service on debian

```
# service resin start
```

8.3 Custom: three static server (triad) in resin.xml

If you are creating a custom resin.xml or modifying the default one, you can configure the servers explicitly in the resin.xml.

The baseline cloud configuration is like a normal Resin configuration: define the three triad servers in your resin.xml, and copy the resin.xml across all servers. You will attach new servers to the cluster when you start it on the command-line. You can still define more servers in the resin.xml, or fewer if you have a small deployment; it's just the basic resin.xml example that uses three servers.

The baseline configuration looks like the following: Define a resin.xml with three static servers as the triad. Start the triad servers as normal. Deploy .wars to the triad with the command-line *deploy* to enable automatic deployment to the new servers. Install a machine (or VM image) with Resin, your licenses, and the resin.xml. Start the new Resin server with an --elastic-server and --cluster command-line option. The new server will load the applications from the triad and join in any clustered services like load-balancing and caching. When load drops, stop the new Resin server as usual.

You can configure the cluster using resin.properties without needing to modify the resin.xml. For the hub servers, add an IP:port for each static server. For the dynamic servers, enable elastic_cloud_enable and home_cluster .

You can also configure the resin.xml directly to add the servers individually.

Example: basic resin.xml for a 3-server hub

```
<resin xmlns="http://caucho.com/ns/resin">
...
  <home-cluster>my-cluster</home-cluster>
...
<cluster id="my-cluster">

  <server id="a" address="192.168.1.10" port="6800"/>
<server id="b" address="192.168.1.11" port="6800"/>
<server id="c" address="192.168.1.12" port="6800"/>

  <resin:ElasticCloudService/>

  ...
</cluster>
...
</resin>
```

The first three <server> tags in a cluster always form the triad. If you have one or two servers, they will still form the hub. One server acts like a Resin standalone; two servers back each other up. More than three <server> tags form static servers acting as spoke servers in the hub-and-spoke model. The static servers are identical to any dynamic servers, but are predefined in the resin.xml

The <resin:ElasticCloudService/> enables dynamic servers. For security, Resin's is to disable dynamic servers.

You can also add a <home-cluster> which provides a default for the --cluster command-line option.

8.3.1 command-line: adding a dynamic server

Before starting a server, your new machine needs the following to be installed: The Java JDK Resin and your licenses Your resin.properties which defines the three triad servers or a custom resin.xml which defines the servers.

Since these three items are the same for each new server, you can make a virtual machine image with these already saved, use the VM image for the new machine and start it.

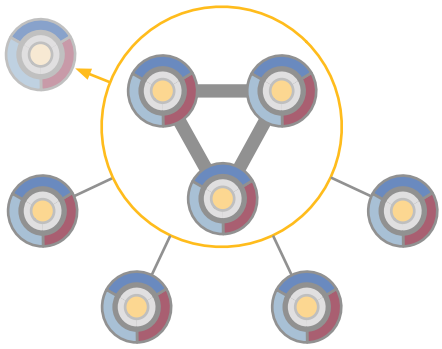
To start a new server, you'll add a *--cluster* option with the name of the cluster you want to join.

Example: command-line starting a dynamic server

```
unix> resinctl --elastic-server --cluster my-cluster start
```

If you don't have a <resin-system-auth-key> in the resin.xml, and you do have admin users defined in the AdminAuthenticator, you will also need to pass a -user and -password arguments.

The new server will join the cluster by contacting the triad. It will then download any deployed applications or data, and then start serving pages.



The triad will inform cluster services about the load balancer, services like caching, admin, JMS, and load-balancer.

8.3.2 Removing a dynamic server

To remove a dynamic server, just stop the server instance. The triad will keep its place in the topology reserved for another 15 minutes to handle restarts, maintenance and outages. After the 15 minutes expire, the triad will automatically remove the server.

8.3.3 Variations: fewer static servers and more servers

Although the three static server configuration is a useful baseline for understanding Resin's clustering, you can configure fewer or more static servers in the resin.xml.

Defining fewer servers than three in the resin.xml is only recommended if you actually have fewer than three servers. Defining more servers than three in the resin.xml depends on your own preference. Having only three servers in the resin.xml means you don't need to change the resin.xml when you add new servers, but listing all servers in the resin.xml makes your servers explicit - you can look in the resin.xml to know exactly what you've configured. It's a site preference.

If you have fewer than three servers, you can define only the servers you need in the resin.xml. You won't get the triple redundancy, but you will still get a backup in the two-server case. For elastic configurations, it's possible to use a single static server in the resin.xml, for example if your load was between one and two servers.

If your site always has two servers active or three servers, you will want to list them in the resin.xml as static servers, even through Resin would let you get away with one. Listing all the servers in the resin.xml ensures that you can connect to at least one in case of a failure. In other words, it's more reliable.

With more static servers than three, you can also add them to the resin.xml. You can also define dynamic servers in the resin.xml if their IP addresses are fixed, because Resin will dynamically adapt to stopped servers. If you use the static/elastic technique, you still need to keep the triad servers up. In other words, you'll adjust load by stopping servers from the end, shutting down server "f" and keeping servers "a", "b", and "c".

8.4 Application Deployment

Cluster deployment works with dynamic servers to ensure each server is running the same application, including the newly spun-up dynamic servers. The new server will ask the triad servers for the most recent application code and deploy it. While it's nice for the convenience (skipping the copy step), it's more important for the extra reliability.

Cloud deployments should generally use the cluster command-line (or browser) deployment instead of dropping a .war in the webapps directory because the cluster deployment automatically pushes deployment to new servers.

With a cluster command-line deployment, the new server will check with the triad hub for the latest deployment. If there's a new version, the dynamic server will download the updates from the triad hub. The cluster deployment ensures all servers are running the same .war deployment. You don't need external scripts to copy versions to each server; that's taken care of by a core Resin capability.

Example: command-line deployment

```
unix> resinctl deploy test.war
```

If you don't have a <resin-system-auth-key> and do have administrator users configured, you will also need to pass the -user and -password parameters.

8.4.1 Configuration for cluster deployment

The basic configuration for cluster deployment is the same as for single-server deployment. Resin uses the same <web-app-deploy> tag to specify where cluster deployment should be expanded. If for some reason you deploy a .war in the webapps directory and deploy one on the clustered command-line, the cluster will take priority.

In the following example, a test.war deployed in the command-line will be expanded to the webapps/test/ directory. The example uses <cluster-default> and <host-default> so every cluster and every virtual host can use the webapps deployment.

Example: resin.xml cluster deployment

```
<resin xmlns="http://caucho.com/ns/resin">
...
<cluster-default>
<host-default>

  <web-app-deploy path="webapps"
expand-preserve-fileset="WEB-INF/work/**"/>

</host-default>
</cluster-default>

<cluster id="app-tier">
  ...
</cluster>
...
</resin>
```

8.5 Load Balancing

See the cluster-load-balancer.xtp documentation for more information about the load balancer.

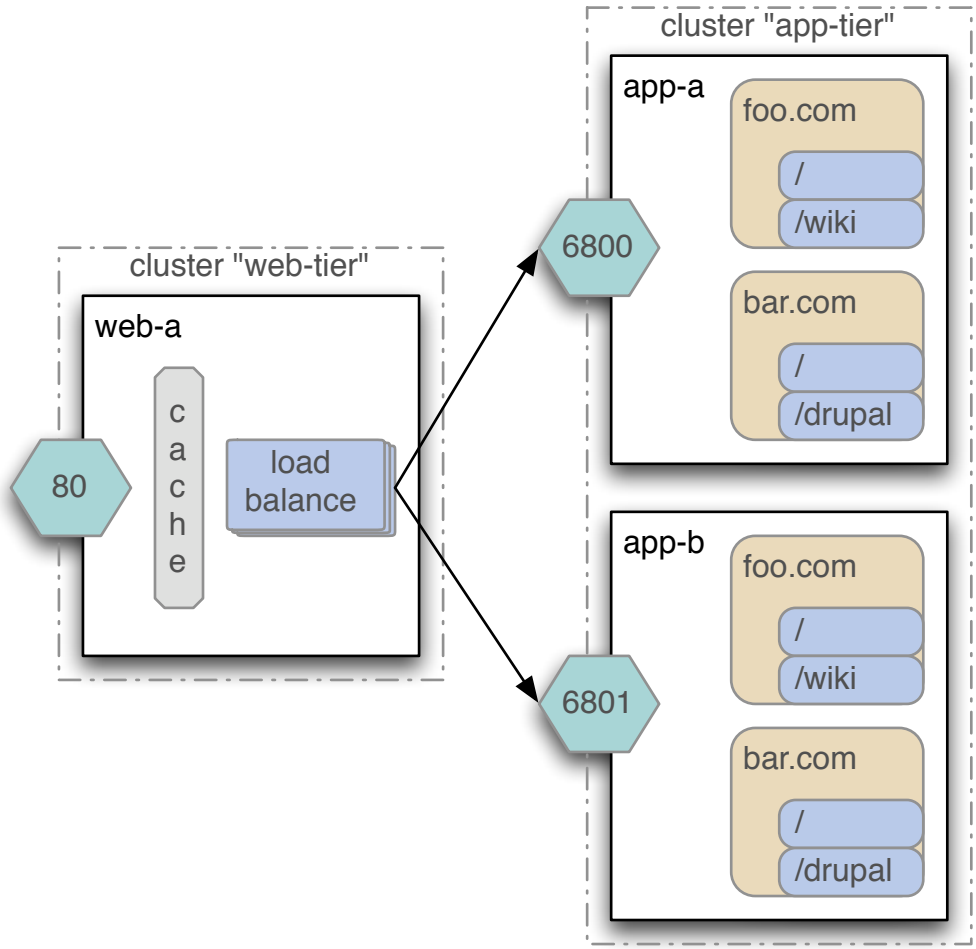
When the new server starts, it needs to receive the new requests from the load balancer. If you're using Resin's load balancer (or Resin's mod_caucho), the load balancer will send HTTP requests to the new server. The load balancer must be configured in the same resin.xml as the application tier because the app-tier cluster needs to communicate with the load balancer tier.

Resin's load balancer is configured in the resin.xml as a web-tier cluster with a http-rewrite.xtp dispatching to the app-tier cluster.

Example: resin.xml for load balancing

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

<cluster id="web-tier">
  <server id="web-a" address="192.168.1.20" port="6800"/>
  <server id="web-b" address="192.168.1.21" port="6800"/>

  <proxy-cache/>

  <host id="">
    <resin:LoadBalance cluster="app-tier"/>
  </host>
</cluster>

<cluster id="app-tier">
  <server id="app-a" address="192.168.1.10" port="6800"/>
  <server id="app-b" address="192.168.1.11" port="6800"/>
  <server id="app-c" address="192.168.1.12" port="6800"/>

  ...
</cluster>

</resin>
```

With the example configuration, Resin will distribute HTTP requests to all static and dynamic servers in the app-tier. Because the HTTP proxy-cache is enabled, Resin will also cache on the web-tier.



If you're not using Resin's load balancer, your cloud will need some way of informing the load balancer of the new server. Some cloud systems have REST APIs for configuring load balancers. Others might require a direct configuration. Resin's load balancer does not require those extra steps.

8.6 Cluster Resources: Cache, Queues

Resin's cluster-aware resources adapt to the added and removed servers automatically. A new server can participate in the same clustered cache as the cluster, see the same cache values, and update the cache with entries visible to all the servers.

The Resin resources that are automatically cache-aware are: jcache: clustered caching distributed sessions JMS queues load-balancing administration deployment

8.6.1 Clustered Caching

Resin's clustered caching uses the jcache API which can either use a jcache method annotation to cache method results or an explicit Cache object injected CDI or JNDI. By minimizing the configuration and API complexities, Resin makes it straightforward to improve your performance through caching.

The jcache method annotation lets you cache a method's results by adding a @CacheResult annotation to a CDI bean (or servlet or EJB.) The following example caches the result of a long computation, keyed by an argument. Because the cache is clustered, all the servers can take advantage of the cached value.

Example: method caching with @CacheResult

```
import javax.cache.CacheResult;

public class MyBean {

  @CacheResult
  public Object doSomething(String arg)
  {
    ...
  }
}
```

When using cached injection, you'll need to configure a cache instance in the resin-web.xml. Your code and its injection are still standards-based because it's using the CDI and jcache standards.

Example: resin-web.xml Cluster cache configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:ClusterCache name="my-cache"/>

</web-app>
```

The cache can be used with CDI and standard injection:

Example: using jcache

```
package mypkg;

import javax.inject.*;
import javax.cache.*;

public class MyClass {
  @Inject
  private Cache<String,String> \_myCache;

  public void doStuff()
  {
    String value = \_myCache.get("mykey");
  }
}
```

Chapter 9

Command-Line

9.1 enabling the commands

By default these commands are disabled. Enabling the command requires ManagerService be registered in resin.xml file.

Since the default resin.xml already includes a <resin:AdminAuthenticator> with a <resin:import>, you can just reuse the admin configuration from the /resin-admin page.

Example: enabling Resin ManagerService

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">
...
<cluster id=''>

  <resin:AdminAuthenticator>
    <user name="admin" password="{SSHA}h5QdSulQyqIgYP7B1J3YfnRSo56kD847"/>
  </resin:AdminAuthenticator>

  <resin:RemoteAdminService/>

  <resin:ManagerService/>

  ...
</cluster>
  ...
</resin>
```

9.2 available commands

Table 9.1: commands

COMMAND	DESCRIPTION
deploy	deploys an application archive
undeploy	un-deploys an application specified by a context
deploy-list	lists all applications deployed on a server
deploy-copy	copies an application from one context to another
start-webapp	starts web application context
stop-webapp	stops web application context
restart-webapp	restarts web application context

Table 9.1: (continued)

COMMAND	DESCRIPTION
heap-dump	produces heap dump
thread-dump	produces thread dump
profile	turn profiling and displays results after sampling completes
jmx-list	lists MBeans, attributes and operations
jmx-dump	dump all MBean attributes and values
jmx-set	sets value of a jmx attribute
jmx-call	calls MBean's method
log-level	changes log-level
pdf-report	generates pdf report

Since all commands require connecting to Resin server remotely list of required parameters includes authentication and remote connection options such as IP and Port.

Table 9.2: common options

ARGUMENT	MEANING	DEFAULT
-conf	configuration file	conf/resin.xml
-address	ip or host name of the server	taken from conf/resin.xml
-port	server http port	taken from conf/resin.xml
-user	user name used for authentication to the server	none, required
-password	password used for authentication to the server	none, required

9.2.1 console: starting in console mode

Start Resin in console mode with `log-level` . Console mode is useful for development or debugging. For production, use `start` or `start-all` instead. Resin will start under control of the control. Log output will go to the console.

```
resinctl console [options]
```

Example: starting in console

```
unix> resinctl console
Resin Professional 4.0.s120731 (built Tue, 31 Jul 2012 03:03:37 PDT)
Copyright(c) 1998-2012 Caucho Technology.  All rights reserved.

  1999999.license -- 1 Resin server Caucho
  ...
[12-08-03 16:04:42.670] {main}
[12-08-03 16:04:42.670] {main} http listening to *:8080
[12-08-03 16:04:42.671] {main}
[12-08-03 16:04:42.672] {main} Resin[id=app-0] started in 1438ms
```

Table 9.3: console options

ARGUMENT/OPTION	MEANING	DEFAULT
--cluster	cluster to join when using --elastic-server	
--elastic-dns	lazy local DNS address binding: retry until succeed	
--elastic-server	start as a dynamic server joining a cluster	
--server	the server name to start	"default" or local IP
--verbose	start with extra debugging	

9.2.2 deploy: deploying a web application

Deploying an application is done with a `deploy` command

```
bin/resin.sh [-conf <file>] deploy [options] <war-file>
```

Example: deploying an application from a hello-world.war archive

```
unix> bin/resin.sh deploy -user admin -password secret /projects/hello-world/hello-world. ↩
    war

Deployed production/webapp/default/hello-world as hello-world.war to http://127.0.0.1:8080/ ↩
    hmtp
```

Table 9.4: deploy options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-name	name of the context to deploy to, defaults to war-file name	[/foo].war
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

9.2.3 deploy-copy: copy application from */mysource* to */mytarget*

Copy a deployed application to a new deployment tag with `deploy-copy` .

```
resinctl deploy-copy [options]
```

Example: deploy-copy

```
unix> resinctl deploy-copy --source mysource --target mytarget

copied production/webapp/default/mysource to production/webapp/default/mytarget
```

Table 9.5: deploy-copy options

ARGUMENT	MEANING	DEFAULT
-source	context to copy application from	none
-source host	host to copy application from	default
-source-stage	source stage	production
-source-version	version of the source application formatted as <major.minor.micro.qualifier>	none
-target	context to copy application to	none
-target-host	host to copy an application to	default
-target-stage	target stage	production
-target-version	version application to use for a target, formatted as <major.minor.micro.qualifier>	none

9.2.4 deploy-list: list deployed applications

List deployed applications with `deploy-list` .

```
resinctl deploy-list [options]
```

Example: deploy-list

```
unix> resinctl deploy-list

production/webapp/default/hello-world
```

9.2.5 heap-dump: producing JVM memory heap dump

To produce a heap dump resin provides `heap-dump` command. With Resin Open Source, `heap-dump` will produce a standard heap dump file and put it into the Resin log directory on the remote machine.

```
bin/resin.sh [-conf <file>] heap-dump [options] [-raw]
```

Example: producing a heap dump on Resin Open Source

```
unix> bin/resin.sh -conf conf/resin.conf heap-dump -user foo -password test -raw

Heap dump is written to '/var/log/resin/heap.hprof'.
To view the file on the target machine use
jvisualvm --openfile /var/log/resin/heap.hprof
```

Resin Pro, when heap-dump is given no `-raw` option is capable of producing a readable summary report.

Example: heap-dump

```
unix: bin/resin.sh -conf conf/resin.conf heap-dump -user foo -password test

Heap Dump generated Fri May 08 02:51:31 PDT 1998
   count   |  self size | child size | class name
        68 |       6528 |    3142736 | com.caucho.util.LruCache
     28768 |     920576 |    2674000 | java.lang.String
     29403 |    2066488 |    2066488 | char[]
        68 |    1927360 |    1927360 | com.caucho.util.LruCache$CacheItem[]
      7346 |     715416 |    1796320 | java.lang.Object[]
```

```
  5710 |    594200 |   1768624 | java.util.HashMap$Entry[]
  2827 |    135696 |   1606264 | java.util.HashMap
 20787 |    665184 |   1489024 | java.util.HashMap$Entry
  9682 |    852016 |   1235984 | java.lang.reflect.Method
 61507 |    984112 |    984400 | java.lang.Object
   337 |     16176 |    889192 | java.util.concurrent.ConcurrentHashMap
  2881 |    161336 |    883584 | java.util.LinkedHashMap
  1596 |    178752 |    702296 | com.caucho.quercus.program.ProStaticFunction
```

Table 9.6: heap-dump options

ARGUMENT/OPTION	MEANING	DEFAULT
-raw	produces a standard .hprof file	

since raw heap dump can be on the order of gigabytes the resulting file is always left on the machine where Resin server is running.

9.3 jmx-list: listing JMX MBeans, attributes and operations

Command `jmx-list` prints out JMX MBeans, attributes and operations registered in a JVM that is running Resin. As its last argument the command accepts <pattern>. The <pattern> follows convention defined for

`javax.management.ObjectName` , defaulting to resin:*, which matches any MBean in resin domain.

```
bin/resin.sh [-conf <file>] jmx-list [options] [<pattern>]
```

Example: listing MBeans

```
unix> bin/resin.sh -conf conf/resin.conf jmx-list -user foo -password test com.acme:*

com.acme:type=Foo
com.acme:type=Bar
```

Table 9.7: jmx-list options

ARGUMENT/OPTION	MEANING
-attributes	outputs a list of attributes for each MBean
-values	outputs a list of attributes and values for each MBean
-operations	outputs a list of operations each MBean
-operations	outputs a list of operations each MBean
-platform	only queries MBeans in java.lang domain(unless pattern is specified)
-all	queries MBeans in any domain(unless pattern is specified)

Example: listing MBeans with values

```
bin/resin.sh -conf conf/resin.conf jmx-list -user foo -password test -values com.acme:*

com.acme:type=Foo
```

```
  attributes:
    javax.management.MBeanAttributeInfo[description=,
            name=Boolean, type=boolean, read-only, descriptor={}]==false
```

9.4 jmx-dump: dump all MBean attributes and values

Command `jmx-dump` produces a complete listing of a MBeans with current attribute values. The output is usually lengthy and can be directed to a file rather than stdout using the -file parameter.

```
bin/resin.sh [-conf <file>] jmx-dump [options] [-file <file>]
```

Table 9.8: jmx-list options

ARGUMENT/OPTION	MEANING	DEFAULT
-file	if specified, writes output to file	

Example: dumping JMX

```
bin/resin.sh -conf conf/resin.conf jmx-dump -user foo -password test
JMX Dump:
java.lang:type=MemoryPool,name=CMS Old Gen {
  Name=CMS Old Gen
  Type=HEAP
...
```

9.5 jmx-set: setting attribute value on MBeans

Command `jmx-set` sets a value on an attribute belonging to a particular JMX MBean.

```
bin/resin.sh [-conf <file>] jmx-set [options] -pattern <pattern> -attribute <attribute>  ↩
    value
```

Example: setting attribute value

```
unix> bin/resin.sh -conf conf/resin.conf jmx-set -user foo -password test \
                   -pattern com.acme:type=Foo -attribute Foo foo-value

value for attribute 'Foo' on bean 'com.acme:type=Foo' is changed from 'null' to 'foo-value'
```

Table 9.9: jmx-set options

ARGUMENT/OPTION	MEANING
-pattern	specifies pattern to match target MBean
-attribute	sets name of the attribute
value	String representation of the value (primitive types and String are supported)

9.6 jmx-call: invoking method on a MBean

Command `jmx-call` calls a method on a specified with < pattern> MBean.

```
bin/resin.sh [-conf <file>] jmx-call [options] -pattern <pattern> -operation <operation>  ↩
    value...
```

Example: invoking method on MBean

```
unix> bin/resin.sh -conf conf/resin.conf jmx-call \
                   -user foo -password test \
                   -pattern com.acme:type=Foo -pattern com.acme:type=Foo \
                   -operation echo hello

method 'echo(java.lang.String)' called on 'com.acme:type=Foo' returned 'hello'.
```

Table 9.10: jmx-call options

ARGUMENT/OPTION	MEANING
-pattern	specifies pattern to match target MBean
-operation	sets name of the operation to invoke
value	space separated list of parameters that will be passed to the method (primitive types and String are supported)

9.6.1 license-add: copy a license to the license directory

The `license-add` is a convenient way to remotely add a license to the correct license directory

```
bin/resin.sh [-conf <file>] license-add -license <license file> [options]
```

Example: copy test.license to the license directory as test.license, overwrite if exists

```
unix> bin/resin.sh license-add -user admin -password secret -license test.license - ↩
    overwrite

add-license wrote test.license successfully
```

Table 9.11: license-add options

ARGUMENT	MEANING	DEFAULT
-license	Path to license file to add	none, required
-to	File name license will be written to	name of license file
-overwrite	Overwrite existing license file if exists	false, true if set
-restart	Restart Resin after license is added	false, true if set

9.6.2 log-level: setting log level

Change the logging level temporarily with `log-level` . The java.util.logging level will change to the new value.

```
resinctl log-level [options] \
            -all|-finest|-finer|-fine|-config|-info|-warning|-severe|-off \
            [-active-time <time-period>] [names...]
```

Example: setting log level

```
unix> resinctl log-level --finer --active-time 5s com.mycom.mypkg

Log level is set to 'FINER', active time 5 seconds: {root}, com.caucho
```

Table 9.12: log-level options

ARGUMENT/OPTION	MEANING	DEFAULT
-active-time	specifies temporary level active time. e.g. 5s	permanent
-<level>	specifies new log level	none, required
value	name of the logger(s). Defaults to root and 'com.caucho' loggers.	

9.6.3 password-encrypt: encrypts a configuration password

Encrypt a configuration password with `password-encrypt` . For a configuration password, you can use the <resin:Password> tag with the password to avoid plaintext.

```
resinctl password-encrypt [options] plaintext
```

The `password-encrypt` encrypts plaintext for a configuration password. See the <resin:Password> for use.

Example: password-encrypt

```
unix> resinctl password-generate mypassword
password: {RESIN}bjuNvBjEDN2m6ynQU8SqQA==
```

Example: resin-web.xml with database

```
<web-app xmlns="http://caucho.com/ns/resin"
   xmlns:resin="urn:java:com.caucho.resin">

  <database jndi-name='jdbc/test_mysql'>
    <driver type="com.mysql.jdbc.Driver">
      <url>jdbc:mysql://localhost:3306/test</url>
      <user>myuser</user>
      <password>
        <resin:Password>{RESIN}bjuNvBjEDN2m6ynQU8SqQA==</resin:Password>
      </password>
    </driver>
  </database>


</web-app>
```

Example: resin.properties openssl_password

```
openssl_password: {RESIN}bjuNvBjEDN2m6ynQU8SqQA==
```

Table 9.13: password-encrypt options

ARGUMENT/OPTION	MEANING	DEFAULT

9.6.4 password-generate: generates an admin password

Generate an admin password with `password-generate` . You can paste the output into the resin.properties file. The admin password is used for /resin-admin, remote deployment and REST administration.

```
resinctl password-generate [options] name password
```

Example: password-generate

```
unix> resinctl password-generate myuser mypassword
admin_user : myuser
admin_password : {SSHA}yAKopu5id740xAoePKZOGyAtu78DpZck
```

Table 9.14: password-generate options

ARGUMENT/OPTION	MEANING	DEFAULT

9.6.5 pdf-report: pdf report generation

Generate a PDF report for the server with `pdf-report` .

```
resinctl pdf-report [options]
```

Example: generate the default watchdog PDF report

```
unix> bin/resin.sh pdf-report -user admin -password secret -watchdog

generated /usr/local/share/resin/log/default-Watchdog-20110801T0251.pdf
```

Table 9.15: pdf-report options

ARGUMENT	MEANING	DEFAULT
-logdir	PDF output directory	Resin log directory
-path	Path to a PDF generating .php file	${resin.home}/doc/admin/pdf-gen.php
-period	Report look back period of time	7D
-snapshot	Take a snapshot of the server before reporting	false
-watchdog	Report on server period to the last restart	false

9.6.6 profile: CPU profiling application

Profile an applications to find bottlenecks and CPU spikes. Resin's

`profile` command turn on sampling for requested period of time and prints out profiling resutls.

```
resinctl profile [options]
```

Example: profiling an application

```
unix> resinctl profile

Profile started at 1998-05-08 02:51:31.001. Active for a total of 5000ms.
Sampling rate 10ms. Depth 16.
   % time  |time self(s)|   % sum    | Method Call
   800.000 |     40.080 |     28.407 | com.caucho.env.thread.AbstractTaskWorker.run()
   300.000 |     15.030 |     39.060 | com.caucho.env.thread.ResinThread.waitForTask()
   200.000 |     10.020 |     46.162 | com.caucho.vfs.JniSocketImpl.nativeAccept()
   116.168 |      5.820 |     50.287 | \_jsp._test__jsp$1.run()
   100.000 |      5.010 |     53.838 | com.caucho.env.thread.AbstractTaskWorker.run()
   100.000 |      5.010 |     57.389 | java.lang.ref.ReferenceQueue.remove()
   100.000 |      5.010 |     60.940 | com.caucho.server.admin.ManagerActor.profile()
   100.000 |      5.010 |     64.491 | java.lang.ref.Reference$ReferenceHandler.run()
   100.000 |      5.010 |     68.042 | java.lang.UNIXProcess.waitForProcessExit()
   100.000 |      5.010 |     71.593 | java.io.FileInputStream.readBytes()
   100.000 |      5.010 |     75.144 | com.caucho.util.Alarm$AlarmThread.run()
   100.000 |      5.010 |     78.694 | com.caucho.env.shutdown. ↩
       ShutdownSystem$ShutdownThread.run()
   100.000 |      5.010 |     82.245 | com.caucho.network.listen.JniSelectManager. ↩
       selectNative()
   100.000 |      5.010 |     85.796 | unknown
   100.000 |      5.010 |     89.347 | com.caucho.vfs.JniSocketImpl.readNative()
   100.000 |      5.010 |     92.898 | com.caucho.test.Test$Timeout.run()
   100.000 |      5.010 |     96.449 | com.caucho.profile.ProProfile.nativeProfile()
   100.000 |      5.010 |    100.000 | java.lang.Thread.sleep()
com.caucho.env.thread.AbstractTaskWorker.run()
  sun.misc.Unsafe.park()
  java.util.concurrent.locks.LockSupport.parkUntil()
  com.caucho.env.thread.AbstractTaskWorker.run()
  com.caucho.env.thread.ResinThread.runTasks()
  com.caucho.env.thread.ResinThread.run()
```

Table 9.16: profile options

ARGUMENT/OPTION	MEANING	DEFAULT
-active-time	specifies profiling time span in ms (defaults to 5000 - 5 sec.)	5s
-sampling-rate	specifies sampling rate (defaults to 10ms)	10ms

9.6.7 restart: restart a daemon server

Restart a Resin daemon server with `restart` .

```
resinctl restart [options]
```

The `restart` command will select the matching server and restart it. If the --server attribute is specified, the matching server will restart. Otherwise, `restart` will look for all configured servers with a matching local IP address. If none match, it will use the default server.

Example: restart a daemon

```
unix> resinctl restart
Resin/4.0.30 restarted -server 'app-0' for watchdog at 127.0.0.1:6600
```

Table 9.17: restart options

ARGUMENT/OPTION	MEANING	DEFAULT
--server	the server name to restart	"default" or local IP

9.6.8 shutdown: shutdown all daemon servers

Shutdown all Resin daemon servers and watchdog with `shutdown` .

```
resinctl shutdown [options]
```

The `shutdown` command shuts down the watchdog manager, and all servers managed by the watchdog.

Example: shutdown watchdog

```
unix> resinctl shutdown
Resin/4.0.30 shutdown watchdog at 127.0.0.1:6600
```

Table 9.18: shutdown options

ARGUMENT/OPTION	MEANING	DEFAULT

9.6.9 start: starting as a daemon

Start Resin as a daemon with `start` . You can also use `start-all` which will start all local servers.

```
resinctl start [options]
```

The `start` command will select a single server. If the --server attribute is specified, the matching server will start. Otherwise, `start` will look for a configured server with a matching local IP address. To start multiple servers on the same machine, use `start-all` If none match, it will use the default server.

Example: starting as a daemon

```
unix> resinctl start
Resin/4.0.30 launching watchdog at 127.0.0.1:6600
Resin/4.0.30 started -server 'app-0' with watchdog at 127.0.0.1:6600
```

To debug any start problems, see the log directory which contains a watchdog-manager.log and jvm-default.log.

Table 9.19: start options

ARGUMENT/OPTION	MEANING	DEFAULT
--cluster	cluster to join when using --elastic-server	
--elastic-dns	lazy local DNS address binding: retry until succeed	
--elastic-server	start as a dynamic server joining a cluster	
--server	the server name to start	"default" or local IP
--verbose	start with extra debugging	

9.6.10 start-all: starting multiple servers as a daemon

Start Resin servers as a daemon with `start-all` . All servers listening to the local IP address will be started.

```
resinctl start-all [options]
```

The `start-all` command will select all matching servers. If the --server attribute is specified, the matching server will start. Otherwise, `start-all` will look for all configured servers with a matching local IP address. If none match, it will use the default server. If none match, and dynamic servers are enabled,

`start-all` will start a dynamic server.

Example: starting as a daemon

```
unix> resinctl start-all
Resin/4.0.30 launching watchdog at 127.0.0.1:6600
Resin/4.0.30 started -server 'app-0' with watchdog at 127.0.0.1:6600
```

To debug any `start-all` problems, see the log directory which contains a watchdog-manager.log and jvm-default.log.

Table 9.20: start-all options

ARGUMENT/OPTION	MEANING	DEFAULT
--cluster	cluster to join when using --elastic-server	
--elastic-dns	lazy local DNS address binding: retry until succeed	
--elastic-server	start as a dynamic server joining a cluster	
--server	the server name to start	"default" or local IP
--verbose	start with extra debugging	

9.6.11 status: status of daemon server

View the status of Resin daemon server with `status` .

```
resinctl status [options]
```

The `status` command prints a status summary for each server managed by the local watchdog.

Example: server status

```
unix> resinctl status
Resin/4.0.30 status for watchdog at 127.0.0.1:6600

watchdog:
  watchdog-pid: 6551

server 'app-0' : ACTIVE
  password: missing
  watchdog-user: caucho
  user: caucho
  root: /var/resin/
  conf: /etc/resin/resin.xml
  pid: 6598
  uptime: 0 days 00h00
```

Table 9.21: status options

ARGUMENT/OPTION	MEANING	DEFAULT

9.6.12 stop: stop a daemon server

Stop a Resin daemon server with `stop` .

```
resinctl stop [options]
```

The `stop` command will select the matching. If the --server attribute is specified, the matching server will stop Otherwise, `stop` will look for all configured servers with a matching local IP address. If none match, it will use the default server.

Example: stop a daemon

```
unix> resinctl stop
Resin/4.0.30 stopped for watchdog at 127.0.0.1:6600
```

Table 9.22: stop options

ARGUMENT/OPTION	MEANING	DEFAULT
--server	the server name to stop	"default" or local IP

9.6.13 thread-dump: producing a thread dump

Dump the JVM's threads with `thread-dump` . The thread dump is produced on a remote sever and printed out locally.

```
resinctl thread-dump [options] [-raw]
```

Example: producing a thread dump

```
unix> resinctl thread-dump -raw

  Thread Dump:
```

```
  "MailboxWorker[manager@resin.caucho]-8" id=31 RUNNABLE
    at sun.management.ThreadImpl.getThreadInfo0 (ThreadImpl.java) (native)
    at sun.management.ThreadImpl.getThreadInfo (ThreadImpl.java:147)
    at com.caucho.util.ThreadDump.threadDumpImpl (ThreadDump.java:88)
    at com.caucho.util.ThreadDump.getThreadDump (ThreadDump.java:62)
    at com.caucho.server.admin.ManagerActor.doThreadDump (ManagerActor.java:148)
    at sun.reflect.NativeMethodAccessorImpl.invoke0 (NativeMethodAccessorImpl.java) (native ↩
        )
    at sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorImpl.java ↩
        :25)
    at java.lang.reflect.Method.invoke (Method.java:597)
    at com.caucho.bam.actor.BamSkeleton$QueryMethodInvoker.invoke (BamSkeleton.java:501)
    at com.caucho.bam.actor.BamSkeleton.query (BamSkeleton.java:215)
    at com.caucho.bam.actor.SkeletonActorFilter.query (SkeletonActorFilter.java:187)
    at com.caucho.bam.query.QueryActorFilter.query (QueryActorFilter.java:95)
    at com.caucho.bam.packet.Query.dispatch (Query.java:86)
    at com.caucho.bam.mailbox.MultiworkerMailbox.dispatch (MultiworkerMailbox.java:268)
    at com.caucho.bam.mailbox.MailboxWorker.runTask (MailboxWorker.java:73)
    at com.caucho.env.thread.AbstractTaskWorker.run (AbstractTaskWorker.java:160)
    at com.caucho.env.thread.ResinThread.runTasks (ResinThread.java:164)
    at com.caucho.env.thread.ResinThread.run (ResinThread.java:130)

  "Signal Dispatcher" id=5 RUNNABLE

  "http://*:8087-1" id=26 RUNNABLE (in native)
    at com.caucho.vfs.JniSocketImpl.readNative (JniSocketImpl.java) (native)
    at com.caucho.vfs.JniSocketImpl.read (JniSocketImpl.java:337)
    at com.caucho.vfs.JniStream.readTimeout (JniStream.java:90)
    at com.caucho.vfs.ReadStream.fillWithTimeout (ReadStream.java:1135)
    at com.caucho.network.listen.TcpSocketLinkListener.keepaliveThreadRead ( ↩
        TcpSocketLinkListener.java:1345)
    at com.caucho.network.listen.TcpSocketLink.processKeepalive (TcpSocketLink.java:767)
    at com.caucho.network.listen.DuplexReadTask.doTask (DuplexReadTask.java:91)
    at com.caucho.network.listen.TcpSocketLink.handleRequests (TcpSocketLink.java:646)
    at com.caucho.network.listen.AcceptTask.doTask (AcceptTask.java:104)
    at com.caucho.network.listen.ConnectionReadTask.runThread (ConnectionReadTask.java:98)
    at com.caucho.network.listen.ConnectionReadTask.run (ConnectionReadTask.java:81)
    at com.caucho.network.listen.AcceptTask.run (AcceptTask.java:67)
    at com.caucho.env.thread.ResinThread.runTasks (ResinThread.java:164)
    at com.caucho.env.thread.ResinThread.run (ResinThread.java:130)

...
```

9.6.14 undeploy: undeploying application

Undeploy an application with `undeploy` . After the undeploy, the application is stopped, removed from the repository and no longer responds to requests.

```
resinctl undeploy [options] <name>
```

Example: undeploy

```
unix> resinctl undeploy undeploy foo

Undeployed foo from http://127.0.0.1:8080/hmtp
```

Table 9.23: undeploy options

ARGUMENT	MEANING	DEFAULT
-host	virtual host of the application	default
-stage	deployment stage of the application	production
-version	version of the application formatted as <major.minor.micro.qualifier>	none

9.6.15 webapp-restart: restarting application

Start an application with `webapp-restart` .

```
resinctl restart-webapp [options] <name>
```

Example: stop web application deployed at context /myapp

```
unix> resinctl webapp-restart myapp

'production/webapp/default/myapp' is restarted
```

Table 9.24: webapp-restart options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

9.6.16 webapp-start: starting application

Start an application with `webapp-start`

```
resinctl webapp-start [options] <name>
```

Example: start web application deployed at context /myapp

```
unix> resinctl webapp-start myapp

'production/webapp/default/myapp' is started
```

Table 9.25: webapp-start options

ARGUMENT	MEANING	DEFAULT
-host	virtual host of the application	default
-stage	deployment stage of the application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

9.6.17 webapp-stop: stopping application

Stop an application with `webapp-stop` .

```
resinctl webapp-stop [options] <name>
```

Example: stop web application /myapp

```
unix> resinctl webapp-stop myapp

'production/webapp/default/myapp' is stopped
```

Table 9.26: webapp-stop options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

Chapter 10

Configuration

10.1 Overview

All Resin configuration is ultimately managed by the resin.xml. To keep configuration manageable, the resin.xml imports properties files like resin.properties, and xml files like cluster-default.xml and health.xml. Many sites will never need to change the resin.xml and can just modify the resin.properties.

10.2 Concepts and naming conventions

cluster - a collection of identically-configured servers. environment - isolated class-loader contexts with shared resources: server, host and web-app are the main environments. host - a HTTP virtual host. proxy cache - HTTP proxy cache. resource - drivers or services available to the application though JNDI or CDI like databases, JMS queues, custom CDI-configured service. Resin-specific resources include security, authenticators, health-checks and the rewrite/dispatch system. rewrite/dispatch - the configuration for dispatching HTTP URLs to servlets and response codes like Apache's mod_rewrite. server - a Resin JVM instance. There may be multiple servers on a machine. watchdog - a JVM instance which watches over the Resin server and restarts the server if necessary. web-app - a HTTP web-application which runs servlets.

10.3 resin.properties

The config-resin-properties.xtp defines properties used by the resin.xml for the Resin server and cluster. Many sites will only need to modify the resin.properties for their configuration.

Example: key resin.properties

```
# web-tier Triad servers: web-0 web-1 web-2
# web_servers      : 192.168.1.20:6810
# web.http          : 80

# app-tier Triad servers: app-0 app-1 app-2
app_servers      : 192.168.1.10:6800 192.168.1.11:6800

app.http          : 8080

setuid_user   : resin
setuid_group  : resin

home_cluster : app
elastic_cloud_enable : true
```

```
jvm_args  : -Xmx2048m -XX:MaxPermSize=256m

web_admin_enable  : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
```

The properties in resin.properties are convered into JSP EL variable

Properties can be qualified by the server's name or by its variable specific to the "app" cluster. You can also use app-0.http to define a custom http port for a specific server.

10.4 resin.xml

The config-resin-xml.xtp is the root of Resin's configuration. Other files like the resin.properties and cluster-default.xml are imported files to organize the configuration. Conceptually, they're all delegated from the resin.xml.

The resin.xml configures clusters, servers, virtual hosts, web-app deployment, resources, health checks, and class-loaders. Almost all of Resin's behavior is configured by the resin.xml.

The standard resin.xml distributed with Resin and its property file reisn.properties configures a standard, flexible clustering system including an application server tier, a http web tier, and virtual hosts. Many sites will not need to modify the standard resin.xml.

The resin.xml is flexible enough to configure non-http servers. If you have a non-http Java service, it can take advantage of Resin's watchdog, lifecycle and health system by being configured as a standard CDI service in a non-http Resin system.

Example: resin.xml outline

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
    <!-- shared configuration across all clusters -->

    <resin:import path="classpath:META-INF/caucho/app-default.xml"/>

    <resin:import path="${__DIR__}/health.xml" optional="true"/>
  </cluster-default>

  <cluster id="my-cluster">

    <server-default>
      <!-- thread limits, JVM config, keepalives, ports, HTTP -->

      <http port="8080"/>
    </server-default>

    <server id="server-a" address="192.168.1.10" port="6800"/>
    <server id="server-b" address="192.168.1.11" port="6800"/>

    <host id="www.myhost.com" root-directory="hosts/myhost.com">

      <resin:MovedPermanently regexp="/old-file" target="/new-path"/>

      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>

      <web-app id="/custom">
      </web-app>
```

```
    </host>

  </cluster>

</resin>
```

10.5 Next Steps

config-resin-properties.xtp configuration shows how to customize basic configuration. config-resin-xml.xtp configuration shows more advanced and specialized configuration. resin-admin-command-line.xtp describes using the resinctl command line interface. deploy.xtp deploying applications to a running server. clustering.xtp describes clustering, cloud, and dynamic servers. resin-admin.xtp describes the /resin-admin browser-based interface. health.xtp describes the Resin health and monitoring system. resin-admin-rest.xtp describes the REST interface for remote third-party admin integration. http.xtp describes fast, scalable HTTP web server. http-rewrite.xtp describes Resin's URL rewriting (like mod_rewrite). security.xtp describes authentication, authorization and SSL.

Chapter 11

Configuration: resin.properties

11.1 Overview

The resin.properties file is a condensed configuration for the most common Resin configurations. Since it is a set of variables used by the main resin.xml confguration file, the variables are defined by convention. The sections following show some of the most common configurations, followed by a description of the variables.

The resin.properties is in the same directory as resin.xml. In unix systems as /etc/resin.properties. In systems deployed in a single directory, it will be ${resin.home}/conf/resin.properties.

Example: key resin.properties

```
...
app_servers : 127.0.0.1:6800

app.http : 8080

setuid_user : resin
setuid_group : resin

jvm_args : -Xmx2G -XX:MaxPermSize=256m
...
```

11.1.1 Qualifying properties by cluster or server

Properties can be restricted to a server or a cluster by prefixing the variable by the cluster or server name. For example, a http port can be specific to the server named *app-0* by using app-0.http . The port can be specific to the cluster *app* with app.http .

Example: http restrictions

```
http       : 8080    # default value if unspecified
app-0.http : 8081    # server 'app-0' uses 8082
app-1.http : 8082    # server 'app-0' uses 8082
app.http   : 8089    # servers in 'app' cluster use 8089
web.http   : 8090    # servers in 'web' cluster use 8090
```

11.1.2 Extending resin.properties in custom resin.xml

Because the resin.properties variables are defined by the resin.xml, sites that use their own custom resin.xml can define their own variables in their resin.xml and set those variables in the resin.properties.

For custom users, the resin.properties is read using a <resin:properties> tag in the resin.xml. The variables are used with the standard EL values.

Example: resin.xml import and use of resin.properties

```
<resin xmlns="http://caucho.com/ns/resin"
    xmlns:resin="urn:java:com.caucho.resin">

  <resin:properties path="${__DIR__}/resin.properties" optional="true"/>

  <dependency-check-interval>${my_var?:'2s'}</dependency-check-interval>

  ...

</resin>
```

The previous example defines the dependency-check-interval using a resin.properties variable my_var and a default value of 2s.

11.2 Application Server Key Properties

A basic application server configuration needs a few key properties to configure the HTTP port, the servers in the cluster, the operating system user and the JVM arguments for memory and GC.

Each server in the cluster can have the same resin.xml and resin.properties. On start, Resin will detect which server is the local one, select it and start it.

Example: key resin.properties

```
...
app_servers : 127.0.0.1:6800

app.http : 8080

setuid_user : resin
setuid_group : resin

jvm_args : -Xmx2G -XX:MaxPermSize=256m
...
```

The server names configured by app_servers are generated automatically by position: app-0, app-1, . . . , app-n.

PROPERTY	DESCRIPTION
app_servers	A list of IP:port addresses for each server in the app-tier cluster.
app.http	The HTTP port for each app-tier server.
setuid_user	The operating system user name for the Resin instance.
setuid_group	The operating system group name for the Resin instance.
jvm_args	The Java arguments for the Resin instance.

app.http configures the HTTP port for the *app* cluster. The http defines the variable used in the resin.xml, and the app restricts it to servers in defined by app_servers . This system allows for different http ports when starting multiple servers on the same machine, like a web-tier load balancer, an app-tier server, and an memcached server.

11.3 Web Tier (Load Balancer) Key Properties

This web tier configuration has one web server that handles the HTTP, load balancing, and proxy caching, and one application server that runs the servlet applications. The web server load balances and proxies HTTP requests to the backend application

cluster. To scale up, add more servers to app_servers .

Example: web-tier resin.properties

```
...
web_servers : 127.0.0.1:6810
web.http    : 80

proxy_cache_enable : true
proxy_cache_size   : 256m

app_servers : 127.0.0.1:6800
app.http    : 8080

setuid_user : resin
setuid_group : resin

jvm_args : -Xmx2G -XX:MaxPermSize=256m
...
```

The server names configured by web_servers are generated automatically by position: web-0, web-1, . . . , web-n.

Example: CLI starting the servers

```
# resinctl start-all
```

The start-all command-line will start both servers in the example because both servers are listening to a local port.

11.4 Memcached Tier Key Properties

To configure Resin as a memcached server, add IP:port addresses to the memcached_servers configuration.

Example: memcached-tier resin.properties

```
...
memcached_servers   : 127.0.0.1:6820
memcached_port      : 11211

memcached.http      : 8080
```

The memcached.http is used for the /resin-admin managment and REST administration.

11.5 property reference

11.5.1 accept_thread_max

accept_thread_max limits the maximum threads waiting for a new connection on the port. Higher values are more resiliant to connection spikes because the idle threads are ready and waiting.

In general, the default value is reasonable and does not usually need changing.

Example: accept_thread_max

```
...
accept_thread_max : 32
accept_thread_min : 4
...
```

11.5.2 accept_thread_min

accept_thread_min triggers a new thread to listen for connections when the number of waiting threads drops below the limit. It works with accept_thread_max to manage the threads waiting in the accept state. Higher values can improve connection spike resposiveness.

In general, the default value is reasonable and does not usually need changing.

Example: accept_thread_min

```
...
accept_thread_max : 32
accept_thread_min : 4
...
```

11.5.3 admin_password

admin_password is the password used with admin_user for /resin-admin administration, REST, and remote resinctl CLI administration.

admin_password is generated with either the CLI resinctl generate-password or the /resin-admin login screen.

Example: admin_user in resin.properties

```
...
admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB

web_admin_enable   : true
web_admin_ssl      : true
web_admin_external : false
remote_cli_enable  : false
rest_admin_enable  : true
rest_admin_ssl     : true
...
```

Example: resinctl generate-password

```
unix> resinctl generate-password my-admin my-password
admin_user : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
```

11.5.4 admin_user

admin_user creates a login user name for /resin-admin administration, REST, and remote resinctl CLI administration. It is used with admin_password and web_admin_enable .

admin_password is generated with either the CLI resinctl generate-password or the /resin-admin login screen.

Example: admin_user in resin.properties

```
...
admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB

web_admin_enable   : true
web_admin_ssl      : true
web_admin_external : false
remote_cli_enable  : false
rest_admin_enable  : true
rest_admin_ssl     : true
...
```

Example: resinctl generate-password

```
unix> resinctl generate-password my-admin my-password
admin_user : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
```

11.5.5 app.http

The HTTP port for the *app* cluster. The "app." is a prefix to the http property, restricting it to the named cluster, "app".

Example: app.http

```
...
app.http : 8080
...
```

11.5.6 app.https

The HTTPS port for the *app* cluster. The "app." is a prefix to the https property, restricting it to the named cluster, "app".

app.https is generally used with openssl_file , openssl_key and openssl_password .

Example: app.https

```
...
app.https : 8443
...
```

11.5.7 cluster_system_key

cluster_system_key configures a shared secret across Resin servers in a cluster. The shared secret is checked when servers in the cluster connect. The key can be any string.

Example: cluster_system_key

```
cluster_system_key : ms8cntp8743z
```

11.5.8 dependency_check_interval

dependency_check_interval sets how often Resin will check for updated files. In development this will be a small value. In deployment it can be a larger value.

Example: dependency_check_interval

```
dependency_check_interval : 2m
```

11.5.9 elastic_cloud_enable

elastic_cloud_enable lets dynamic servers join a cluster. If the "start" command either has an "--elastic-server" flag or if the elastic_server property is set, Resin will connect to the cluster as a new dynamic server.

The cluster is either specified by "--cluster foo" or the home_cluster property. The cluster triad hub servers are specified by the app_servers property.

Example: elastic_cloud_enable

```
app_servers : 182.168.1.10:6800

elastic_server       : true
elastic_cloud_enable : true
home_cluster         : app
```

To start a dynamic server, use resinctl with start and specify --elastic-server to force a dynamic server. The --cluster is optional if a home-cluster has been defined.

Example: elastic_cloud_enable

```
# resinctl start --elastic-server --cluster app
```

11.5.10 elastic_dns

elastic_dns is used in cloud environments that assign IP addresses or DNS names after the server starts, for example in an EC2 environment. When elastic_dns is enabled, Resin will start the server, wait and retry until the local IP addresses or DNS resolve to a matching server.

For example, a Resin cluster might have a fixed pre-allocated DNS name of "app0.mydomain" and app_servers might include that address. At server boot time, the cloud assigns the dynamic IP 192.168.2.114. Later the cloud or the user will attach the IP to the DNS name "app0.mydomain". Resin will then recognize it and start with the correct value. Reserve DNS address "app0.mydomain" with cloud provider. Configure resin.properties with that fixed address and elastic_dns . Start Resin instance in the cloud (with address 192.168.2.114) [Resin is in wait mode until the following step because it doesn't lookup "app0.mydomain" to 192.168.2.114 in the configuration files.] Assign the DNS name "app0.mydomain" to 192.168.1.10 with the cloud provider. Resin will detect the new assignment and start as that server.

Example: elastic_dns

```
app_servers : app0.mydomain:6800

elastic_dns           : true
```

You can specify the elastic_dns in the resin.properties or equivalently specify it on the command line as --elastic-dns .

Example: CLI elastic-dns

```
# resinctl start-all --elastic-dns
```

11.5.11 elastic_server

elastic_server starts the server as a dynamic server and joins a cluster. The dynamic server must have a hub server configured in the cluster to register itself. The cluster must also have the elastic_cloud_enable set to enable dynamic servers.

When the "start" command either has an "--elastic-server" flag or if the elastic_server property is set, Resin will connect to the cluster as a new dynamic server.

The cluster is either specified by "--cluster foo" or the home_cluster property. The cluster triad hub servers are specified by the app_servers property. Either elastic_server in the resin.properties or --elastic-server on the command line to start an elastic server. Either home_server in the resin.properties or --cluster on the command line to specify the cluster. elastic_cloud_enable in the resin.properties to enable dynamic servers. At least one static server defined in app_servers in the resin.properties as the hub. Three servers is preferred.

Example: elastic_server

```
app_servers : 182.168.1.10:6800

elastic_server       : true
elastic_cloud_enable : true
home_cluster         : app
```

To start a dynamic server, use resinctl with start and specify --elastic-server or elastic_server in the resin.properties to force a dynamic server. The --cluster is optional if a home-cluster has been defined.

Example: elastic_server

```
# resinctl start --elastic-server --cluster app
```

11.5.12 email

email is an admin email address used to send automatic reports like weekly PDF reports.

Example: email

```
email : myuser@admin.example.org
```

11.5.13 http

The HTTP port for all servers, usually restricted to the cluster or server as app.http or web.http . If the plain http property is used, it is the default http for all servers.

Example: http

```
...
http     : 8080
app.http : 8081   # overrides for a server in cluster 'app'
...
```

11.5.14 http_address

http_address selects a specific IP address and port for HTTP. It can be useful when each server needs to bind to a different address.

The http_address can be qualified by the cluster or the individual server like other properties. For example, app-0.http_address configures the address for server app-0 in the app cluster.

Example: http_address

```
...
app-0.http_address     : 192.168.1.10
app-1.http_address     : 192.168.1.11

app.http               : 8080
...
```

11.5.15 http_ping_urls

http_ping_urls is a list of URLs to check for server reliability. Resin's health system will periodically check the URLs on the current server. If the server does not respond, the health system will force an automatic restart of Resin.

Example: http_ping_urls

```
http_ping_urls : http://www.myfoo.com/my-test http://www.mybar.com/my-test2
```

11.5.16 jvm_args

jvm_args defines the JDK memory and GC parameters to start the Resin process.

Example: jvm_args

```
jvm_args : -Xmx2G -XX:MaxPermSize=256m
```

11.5.17 openssl_file

openssl_file configures the OpenSSL *.crt certificate file when using SSL. The file location is relative to the resin.xml file.

Example: openssl_file

```
app.https         : 8443

openssl_file      : keys/foo.crt
openssl_key       : keys/foo.key
openssl_password  : my-password
```

11.5.18 openssl_key

openssl_key configures the OpenSSL *.key key file when using SSL. The file location is relative to the resin.xml file.

Example: openssl_key

```
app.https         : 8443

openssl_file      : keys/foo.crt
openssl_key       : keys/foo.key
openssl_password  : my-password
```

11.5.19 openssl_password

openssl_password configures the password for OpenSSL key file when using SSL.

Example: openssl_key

```
app.https         : 8443

openssl_file      : keys/foo.crt
openssl_key       : keys/foo.key
openssl_password  : my-password
```

11.5.20 port_thread_max

port_thread_max restricts the active threads available for a single port. Connections beyond the limit will use the async/select manager, and queue for an available thread.

port_thread_max can be used to limit the maximum load on an overloaded system. If more connection arrive than port_thread_max and beyond the select manager, they will be delayed instead of overloading the system.

Example: port_thread_max

```
...
port_thread_max   : 256
accept_thread_max : 32
accept_thread_min : 4
```

11.5.21 properties_import_url

properties_import_url adds a new properties file to be processed after the resin.properties. Cloud servers using EC2-style /user-data can use the url to customize properties for each server.

Example: properties_import_url for EC2

```
...
properties_import_url : http://169.254.169.254/latest/user-data
```

11.5.22 proxy_cache_enable

proxy_cache_enable enables Resin's HTTP proxy cache. The proxy cache improves performance by saving the server's generated pages. It can be used for formatted web pages or REST-style GET pages that change infrequently.

Example: proxy_cache_enable

```
proxy_cache_enable : true
proxy_cache_size  : 256m
```

11.5.23 proxy_cache_size

proxy_cache_size is the size of the in-memory proxy cache size. The actual maximum size is much larger, because the memory is used as a block cache for a disk based store. The memory block cache is also used for distributed caching.

Example: proxy_cache_size

```
proxy_cache_enable : true
proxy_cache_size  : 256m
```

11.5.24 remote_cli_enable

remote_cli_enable enables the CLI resinctl to be used for remote servers. By default the CLI is restricted to the local network.

Example: remote_cli_enable

```
remote_cli_enable : true
```

11.5.25 rest_admin_enable

rest_admin_enable enables REST-based administration of a Resin server. REST can be used to integrate Resin with remote administration consoles or status scripts.

rest_admin_enable requires an admin_user and admin_password for security. It can optionally be restricted to SSL ports with resin_admin_ssl .

Example: rest_admin_enable in resin.properties

```
...
rest_admin_enable  : true
rest_admin_ssl     : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
...
```

11.5.26 rest_admin_ssl

rest_admin_ssl requires an SSL connection for REST-based administration of a Resin server. REST can be used to integrate Resin with remote administration consoles or status scripts.

Example: rest_admin_ssl in resin.properties

```
...
rest_admin_enable  : true
rest_admin_ssl     : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
...
```

11.5.27 sendfile

sendfile enables operating system sendfile() call, which will send an entire file without requiring Resin to read the file itself. sendfile is particularly useful when much of the traffic is large static files.

Example: sendfile

```
...
sendfile   : true
tcp_cork   : true
```

11.5.28 session_store

session_store enables clustered persistent sessions for failover.

Example: session_store

```
...
session_store : true
...
```

11.5.29 setuid_user

On unix, setuid_user runs the Resin instance as the specified user for security.

Example: setuid_user

```
...
setuid_user : resin
setuid_group : resin
...
```

11.5.30 setuid_group

On unix, setuid_group runs the Resin instance as the specified group for security.

Example: setuid_group

```
...
setuid_user : resin
setuid_group : resin
...
```

11.5.31 tcp_cork

tcp_cork enables advanced TCP flow control on Linux systems for improved performance of large files. When it is enabled, sent data will be buffered in the operating system until the buffers fill, instead of being sent out with partial buffers. An application flush() will still force the data to be sent out.

Example: tcp_cork

```
...
tcp_cork   : true
```

11.5.32 web_admin_enable

web_admin_enable enables /resin-admin for a Resin server. /resin-admin can show the server status, report statistics graphs, deploy applications, and generate PDF reports.

web_admin_enable requires an admin_user and admin_password for security. It can optionally be restricted to SSL ports with web_admin_ssl .

Example: web_admin_enable in resin.properties

```
...
web_admin_enable  : true
web_admin_ssl     : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
...
```

11.5.33 web_admin_external

web_admin_external enables /resin-admin access for servers outside the local network. If enabled, it should be always used with web_admin_ssl .

Example: web_admin_external in resin.properties

```
...
web_admin_enable   : true
web_admin_external : true
web_admin_ssl      : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
...
```

11.5.34 web_admin_ssl

web_admin_ssl requires an SSL connection for /resin-admin administration of a Resin server. /resin-admin can show the server status, report statistics graphs, deploy applications, and generate PDF reports.

Example: web_admin_ssl in resin.properties

```
...
web_admin_enable  : true
web_admin_ssl     : true

admin_user     : my-admin
admin_password : {SSHA}G3UOLv0dkJHTZxAwmhrIC2CRBZ4VJgTB
...
```

11.5.35 webapp_multiversion_routing

webapp_multiversion_routing is a deployment versioning system where Resin selects the most recent deployed application. Web-apps named with numeric suffixes, e.g. foo-10.0.war and can be browsed as /foo. When a new version of the web-app is deployed, Resin continues to route active session requests to the previous web-app version while new sessions go to the new version, so users will not be aware of the application upgrade.

Example: webapp_multiversion_routing

```
...
webapp_multiversion_routing : true
...
```

Chapter 12

Configuration: resin.xml

12.1 Concepts and naming conventions

cluster - a collection of identically-configured servers. environment - isolated class-loader contexts with shared resources: server, host and web-app are the main environments. host - a HTTP virtual host. proxy cache - HTTP proxy cache. resource - drivers or services available to the application though JNDI or CDI like databases, JMS queues, custom CDI-configured service. Resin-specific resources include security, authenticators, health-checks and the rewrite/dispatch system. rewrite/dispatch - the configuration for dispatching HTTP URLs to servlets and response codes like Apache's mod_rewrite. server - a Resin JVM instance. There may be multiple servers on a machine. watchdog - a JVM instance which watches over the Resin server and restarts the server if necessary. web-app - a HTTP web-application which runs servlets.

12.2 resin.xml outline

Example: resin.xml outline

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
    <!-- shared configuration across all clusters -->

    <resin:import path="classpath:META-INF/caucho/app-default.xml"/>

    <resin:import path="${__DIR__}/health.xml" optional="true"/>
  </cluster-default>

  <cluster id="my-cluster">

    <server-default>
      <!-- thread limits, JVM config, keepalives, ports, HTTP -->

      <http port="8080"/>
    </server-default>

    <server id="server-a" address="192.168.1.10" port="6800"/>
    <server id="server-b" address="192.168.1.11" port="6800"/>

    <host id="www.myhost.com" root-directory="hosts/myhost.com">

      <resin:MovedPermanently regexp="/old-file" target="/new-path"/>
```

```
      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>

      <web-app id="/custom">
      </web-app>

    </host>

  </cluster>

</resin>
```

<cluster> is an environment container for servers, HTTP virtual hosts, and common resources. All Resin servers in a cluster share configuration.

12.3 defaults: server-default, cluster-default, web-app-default

When configuration is the same across several servers, like a thread-max or http ports, you can save typing by putting the common configuration in a server-default. Similarly, if all your clusters share common configuration, or all virtual-hosts are organized in the same way, you can use a cluster-default, host-default, and web-app-default to group the shared configuration.

The following example from the sample resin.xml uses a cluster-default and a <resin:import> to share common web-app configuration and the health-system configuration across all clusters in the system.

Example: cluster-default for health

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
    <resin:import path="classpath:META-INF/caucho/app-default.xml"/>

    <resin:import path="${__DIR__}/health.xml" optional="true"/>
  </cluster-default>

  <cluster id="my-cluster-a">
    ...
  </cluster>

  <cluster id="my-cluster-b">
    ...
  </cluster>

</resin>
```

12.4 Server: configuring a JVM instance

See cluster-server.xtp for more details on configuring the server.

The <server> tag configures specific JVMs in a cluster, the server-specific configuration, not the shared cluster configuration. Because the watchdog uses the <server> tag to launch the Resin JVM, the <server> tag also configures JVM parameters. the server id used to identify the server. the IP local address and port for communication within the server. JVM parameters like -Xmx thread limits like thread-idle-min and thread-idle-max. TCP ports including the HTTP ports, keepalive timeouts and the cluster port. load balancing configuration, including timeouts and weighting. user-name and group-name for Unix setuid security.

The <server> tags configure the servers in a cluster. Dynamic servers launched with the --elastic-server and --cluster command-line arguments will use the same <server-default> configuration as other servers.

The first three servers in the cluster automatically form the triad which is Resin's triple-redundant cluster hub. If you have fewer than three servers, Resin will create a smaller hub.

12.5 Watchdog: protecting the server

The watchdog JVM watches over a Resin server and automatically restarts it when something goes wrong. The watchdog is responsible for setting the JVM arguments for the Resin server, logging its output, and handling the Unix setuid for user-name and group-name.

The configuration for the watchdog itself is inside the <server> that it manages like <watchdog-port>.

12.6 Cluster: multiple servers performing a common task

See clustering.xtp for more details on configuring the clustering.

A Resin cluster gathers a set of servers to perform a common task, like serving HTTP requests, or load balancing, or processing a JMS queue. All Resin servers belong to a cluster. Even if you have one server, it belongs to a cluster of one.

The shared task is configured inside the <cluster> tag, whether it's traditional HTTP requests or JMS queues or SOA-style services. While HTTP has special configuration tags like <host> and the rewrite system, non-HTTP resources can use CanDI-style configuration for their services.

Example: cluster for http servlets

```
<resin xmlns="http://caucho.com/ns/resin">
  ...

  <cluster id="http">
    <server id="a" address="192.168.1.10" port="6800">
      <http port="8080"/>
    </server>

    <host id="">
      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>
    </host>
  </cluster>
</resin>
```

A Resin cluster can also serve non-HTTP resources. The JVM lifecycle, watchdog restarts, health checks, and cluster communication will be managed by Resin, while the service itself will be managed by application code.

Example: cluster for http servlets

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:myservice="urn:java:com.mycom.myservice">
  ...

  <cluster id="http">
    <server id="a" address="192.168.1.10" port="6800"/>

    <myservice:MyService address="${resin.address}">
      ... <!-- CanDI configuration for the service -->
    </myservice:MyService>

  </cluster>
</resin>
```

The <cluster> tag forms a class-loader resource environment that is shared across all hosts and web-apps on the server. Shared CDI resources like databases, queues, and caches can be configured in the <cluster> tag and will be visible in all environment contexts.

12.7 Host: HTTP virtual hosts

See http-virtual-hosts.xtp for more details on configuring the virtual host.

A HTTP virtual host is configured with the <host> tag. The host name is given by the "id" attribute or the "host-name" attribute. The host with an empty name, like id="", is the default host.

Hosts can be configured with a specific <host> tag or implicitly with a <host-deploy> tag. Hosts share common configuring using the <host-default> tag. <host-deploy> specifies a "hosts" directory which implicitly creates hosts like the "webapps" directory for <web-app-deploy>.

The <host> tag dispatches HTTP urls to a virtual host. The host will typically contain one or more web-apps, which will be configured with the <web-app-default>, <web-app> and <web-app-deploy> tags.

For clustered deployment, the <host-deploy is used as a destination for a command-line deployment. The command-line deployed host will expand across the cluster into the "hosts" directory for each server in the cluster.

Since the host defines a resource environment, it can define shared resources like databases, caches, CDI services, and queues. The host can also define rewrite-dispatch URL rules.

12.8 Web-App: HTTP/servlet applications

See deploy.xtp for more details on configuring the web-app.

A web-app is an application based around HTTP requests using and servlets to process the request. Web-apps are typically configured with a WEB-INF/web.xml and a WEB-INF/resin-web.xml and are typically deployed with a my-web-app.war file either by a command-line cluster deploy or by placing it directly in a webapps directory.

Web-apps can be configured with a specific <web-app> tag in a resin.xml <host> or implicitly with a <web-app-deploy> tag. Web-apps share common configuring using the <web-app-default> tag. <web-app-deploy> specifies a "webapps" directory.

Since the web-app defines a resource environment, it can define shared resources like databases, caches, CDI services, and queues. The host can also define rewrite-dispatch URL rules.

12.9 Rewrite: controlling URL dispatch

See http-rewrite.xtp for more details on configuring the rewrite/dispatch.

When you need to control URL formatting and dispatching, because URLs moved, or for better-looking URLs or for marketing tests, Resin's rewrite tags can help. The rewrite capabilities are similar to Apache's mod_rewrite and are integrated with Resin's HTTP and servlet dispatching.

As a simple example, sites like wikis often change readable URLs into a servlet URL and a query string. The following example passes through *.php, *.gif, etc. files and any URL that maps to a file. It rewrites any other URL to use the /index.php file.

Example: dispatch for a wiki

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

<resin:Dispatch target="\.(php|gif|css|png|js)"/>
<resin:Dispatch>
  <resin:IfFileExists/>
</resin:Dispatch>

<resin:Dispatch regexp="^" target="/index.php"/>

</web-app>
```

Resin's load balancing, http proxying and fast-cgi is configured with the rewrite-dispatch configuration. This means you can forward HTTP requests for a single web-app to a separate cluster of backend servers.

Example: load balancing

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

<resin:LoadBalance regexp="^/test" cluster="backend-cluster"/>

</web-app>
```

12.10 Load balancing: using backend servers for HTTP

See cluster-load-balancer.xtp for more details on configuring the load balancer.

HTTP load balancing is integrated in Resin using adaptive round-robin scheduling with failover to spread the HTTP traffic across backend servers. Because it is integrated with Resin's clustering, the load-balancer will automatically be informed when servers are added or removed elastically.

The load balancer is dispatched with a standard rewrite action. The socket parameters and timeouts are configured as part of the target <server>. The load-balancer will read the server's keepalive, timeouts and weights.

Example: load balancing dispatch

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

<resin:LoadBalance regexp="^/test" cluster="backend-cluster"/>

</web-app>
```

Example: load balancing timing configuration

```
<resin xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

<cluster id="backend-cluster>
  <server id="a" address="192.168.1.10" port="6800">
    <load-balance-socket-timeout>60<load-balance-socket-timeout>
    <load-weight>200<load-balance-weight>
  </server>

  ...
```

12.11 Logging: JDK java.util.logging

See logging.xtp for more details on configuring logging.

Resin's logging configuration is based on the JDKs java.util.logging capabilities. In addition, the watchdog saves the standard output from the JDK in a log/jvm-server.log file.

JDK logging has two components: log handlers and loggers. Log handlers take log messages and process them. Resin's primary log handler formats messages and saves them to a log file. JDK loggers are used by Java code to send messages. Each logger has a name, often the same as the Java class name that logs the message, like "com.caucho.util.ThreadPool".

Logging is configured by attaching log-handler to logger names and enabling logging levels. The logging level for both a log-handler and the logger must match for the item to be logged. So some handlers might accept all logging messages, but the loggers only receive "info" messages.

The following log-handler sends all logging messages to the JVM's standard output. It also enables all loggers at the "info" level and the MyBean logger at the "finer" level. So the standard output will contain both the info and the finer logs.

Example: basic logging configuration

```
<resin xmlns:resin="urn:java:com.caucho.resin">

  <log-handler name="" level="all" path="stdout:"
               timestamp="[%y-%m-%d %H:%M:%S.%s] {%{thread}} "/>

  <logger name="" level="info"/>
  <logger name="com.foo.MyBean" level="finer"/>

  ...
</resin>
```

12.12 Resources: databases, queues, caches, and custom

See database.xtp for more details on configuring databases.

Resources can be configured using a general XML syntax that will support any Java-based resource, for example an ActiveMQ queue. Resin will register the resource with CDI or JNDI, letting your application pick up the resource.

Some common resources like databases have their own Resin configuration tags. Most will use a CDI-based syntax.

Resources are stored in class-loader environments like the web-app, host or cluster. Since the environments are isolated from each other, a database "foo" configured in web-app /foo will not affect a web-app /bar.

The CDI-style configuration is a straight mapping between XML and Java classes. You can instantiate Java beans, configure their CDI annotations and properties, and even inject other beans using JSP/EL.

For example, the following configures a Resin clustered jcache, making it available as a CDI injection for any application with the same @Inject @Named. In other words, your application can be written using standard CDI injection with standard jcache and use Resin's configured implementation.

The ClusterCache implementation shares cache data across the cluster.

Example: jcache configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
        xmlns:resin="urn:java:com.caucho.resin"
        xmlns:ee="urn:java:ee">

  <resin:ClusterCache ee:Named="test-cache" resin:Jndi="cache/test">
    <expire-timeout>1h</expire-timeout>
  </resin:ClusterCache>

</web-app>
```

12.13 Health: monitors, actions, and reports

See health.xtp for more details on configuring the health system.

Resin's health system continually monitors the server's health. sensors: gather data from Resin internal state. meters: save sensor and JMX data in an internal database. health-checks: evaluate the health of a Resin system. actions: take actions based on the Resin health like sending mail, gathering more detailed information, or restarting the server. reports: prints pdf reports for the server's history.

12.13.1 Health Checks

Health checks are run every few minutes, check the JVM state, and return a simple response whether the health is OK, WARNING, CRITICAL or FATAL. If you've used the Nagios administration tool, the concept is similar. Resin has several built-in health checks, and it's straight-forward to write a custom health check.

The health check results will be logged if they are not okay, and can be used to trigger actions like sending mail or restarting the server.

In the following example, the JvmDeadlockHealthCheck asks the JVM to check for thread deadlocks. If a deadlock is detected, the health check returns a fatal. The MemoryTenuredHealthCheck looks at the JVM's free memory. If the free memory is too low, it will force a heap garbage-collection, and if that fails it will return a "critical" result.

Example: Health Checks

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster-default>

  <health:JvmDeadlockHealthCheck/>

  <health:MemoryTenuredHealthCheck>
    <memory-free-min>1m</memory-free-min>
  </health:MemoryTenuredHealthCheck>

</cluster-default>

</resin>
```

12.13.2 Health Actions

Health actions can be triggered based on the health check results, JMX values, or as timed events. Each minute Resin will run through the health actions and will execute any that match their predicates.

Built-in actions include restarting, generating reports, sending mail, and gathering information like thread dumps, executing custom PHP pages. Custom actions are straightforward to implement.

The following Restart action will force a Resin restart if any health check returns a fatal or if the health checks are critical for more than five minutes.

Example: Restart Actions

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster-default>

  <health:Restart>
    <health:Or>
      <health:IfHealthFatal/>
      <health:IfHealthCritical time="5m"/>
    </health:Or>
  </health:Restart>

...
</cluster-default>
...
</resin>
```

12.13.3 Health Meters

Health meters gather data every minute and record the data in an internal database for graphing in the /resin-admin or reporting as a PDF.

Two examples from the standard health.xml configuration store JDK data from internal JDK MBeans. The first is a direct meter which saves the operating system's physical memory. The second is a delta meter which saves the JIT compilation time for the 60s interval.

Example: JMX meters

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster-default>

  <health:JmxMeter>
    <name>OS|Memory|Physical Memory Free</name>
    <objectName>java.lang:type=OperatingSystem</objectName>
    <attribute>FreePhysicalMemorySize</attribute>
  </health:JmxMeter>

  <health:JmxDeltaMeter>
    <name>JVM|Compilation|Compilation Time</name>
    <objectName>java.lang:type=Compilation</objectName>
    <attribute>TotalCompilationTime</attribute>
  </health:JmxDeltaMeter>

...
</cluster-default>
...
</resin>
```

12.13.4 Health Reports

The PDF reports can be customized to select the graphs and meters displayed. Each graph consists of a set of meters. For example, a graph about threads might show the JDK thread state counts (runnable, blocked, etc) or it might show the state of the Resin thread pool.

Example: Summary Report

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster-default>

  <health:MeterGraphPage>
    <name>Summary</name>
    <period>6h</period>
    <columns>3</columns>

    <graph name="Request Count">
      <meter>Resin|Http|Request Count</meter>
    </graph>

    <graph name="Threads">
      <meter>JVM|Thread|JVM Thread Count</meter>
      <meter>Resin|Thread|Thread Count</meter>
      <meter>Resin|Thread|Thread Idle Count</meter>
      <meter>JVM|Thread|JVM Runnable Count</meter>
      <meter>JVM|Thread|JVM Blocked Count</meter>
      <meter>JVM|Thread|JVM Native Count</meter>
      <meter>JVM|Thread|JVM Waiting Count</meter>
```

```
    </graph>

</cluster-default>
</resin>
```

12.14 Security: Authenticators and Contraints

See security.xtp for more details on configuring security.

Security can be configured using CanDI-style resources or with the standard servlet <security-constraints. authenticator : checks credentials (password) for users. login : gets credentials from HTTP/servlet and passes to the authenticator. constraints : allows or disallows access to resources.

The following example uses form-based login to protect a /admin.jsp page. Only users with the "admin" role are allowed access. The <resin:XmlAuthenticator> configures a file-based authenticator. The <resin:FormLogin> configures form-based login. And <resin:Allow> restricts access to the page.

Example: form-based authentication

```
<web-app xmlns:resin="urn:java:com.caucho.resin">

  <resin:XmlAuthenticator>
    <user name="admin"
              password="{SSHA}h5QdSulQyqIgYo7BIJ3YfnRSY56kD847"
              role="user,admin"/>
  </resin:XmlAuthenticator>

  <resin:FormLogin>
     <login-page>/login.jsp</login-page>
     <error-page>/form_error.jsp</error-page>
  </resin:FormLogin>

  <resin:Allow url-pattern="/admin.jsp">
     <resin:IfUserInRole role="admin"/>
  </resin:Allow>

</web-app>
```

Chapter 13

Configuration: CDI and XML

13.1 Overview

Because Resin's configuration (CanDI) creates and updates Java objects, each XML tag exactly matches either a Java class or a Java property. For security and rewrite rules, the JavaDoc helps document the configuration. Your own application's configuration can use its JavaDoc to describe the XML tags, and Resin's configuration files like the META-INF/resin-web.xml to customize your application. Selecting drivers and services: by specifying the Java class, like choosing a JDBC database driver, or application services. The Java classes should be self-documented in the application's JavaDoc. Configuring properties: with XML values for each important property of the Java class, like setting URLs, timeouts, and connection limited. More complicated configuration is supported allowing configuration similar to domain-specific languages, for example Resin's rewrite rules and security configuration. Registration for the application: the application looks up configured resources either through Java Injection or in JNDI. The application will already have defined names and binding types for the XML configuration to use. Linking references: in some more complicated applications, the configuration can link resources and services together using the JSP expression language, specifying resources by name like

13.2 XML configuration

Services, drivers, and third-party libraries are registered and configured using the standard Resin configuration files resin.xml or resin-web.xml files as well as the META-INF/beans.xml. Application services and libraryes are treated as first-class components just like Resin-internal resources, because Resin's own configuration uses the same CanDI configuration. Even standard JavaEE configuration like servlets, JSP .tld files, and EJBs are configured with CanDI.

The configuration in Resin is smaller than some other dependency frameworks because only components which need customization need to be in the XML. If your application is using Java Injection internally, most of the wiring occurs automatically through Java code annotations instead of being configured in XML. The annotation focus makes the Java code self-describing, and also simplifies the management by shrinking the needed XML.

The XML-configuration lets you customize your application for a particular environment, e.g. setting configuration parameters. For example, Resin's `<database>` needs to select a database driver and configure the URL, user and password of the database as well as configuring connection pooling parameters. Some application services will also need configuration.

In addition, the XML-configuration documents the services you've enabled. For heavyweight services, this documentation is critical, while lightweight components do not need this extra housekeeping overhead.

13.2.1 Component overview

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:mypkg="urn:java:com.mycom.mypkg">

  <mypkg:MyBean>
```

```
    [class annotations, service, and registration]

    [bean constructor args]

    [property configuration]

    [method annotation configuration]
  </mypkg:MyBean>

</web-app>
```

The Bean/Component name and package is the Java class which implements the resource you're configuring, either your own application class, or a library class. The class annotations register up the bean using a binding-type, or a name and may register the bean as a service, like a servlet, or EJB or remote service. The optional bean constructor args are needed for some specialized beans. The application or library JavaDocs will document if you need to use the constructor. The properties configure the bean, like setting a database's url or max-connections. Each XML tag represents a bean property, and matches exactly with the bean's JavaDoc. The optional and rare method annotations are for special cases like aspect-oriented interception. Typically, these annotations are configured by the library developers.

13.3 Service and component registration

The `<my:MyBean>` tags register application classes with Resin. The default scope of a `<my:MyBean>` is `@Dependent`. A `<my:MyBean>` will create a new instance each time it's injected or referenced.

Example: bean and component META-INF/beans.xml

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:ee="urn:java:ee"
        xmlns:example="urn:java:example">

  <example:MyService>
    <ee:ApplicationScoped/>
  </example:MyService>

  <example:MyComponent>
  </example:MyComponent>

</beans>
```

The `<my:MyBean>` tags can configure fields and annotations:

Table 13.1: XML annotations

ATTRIBUTE	DESCRIPTION
<ee:Named>	the @javax.inject.Named annotations for EL naming
<ee:Qualifier>	any @javax.inject.Qualifier annotations for injection
my:myField	optional configuration, using bean-style assignment
<ee:SessionScoped>	specifies scope of the instances: request, conversation, session, application, or singleton

13.4 References and EL Expressions

Some services and components need a name because they're used as a JSP or JSF reference, or because the configuration needs a reference to the component. Resin configuration files can use EL expressions to get references to resources, beans, system properties, and calculate generatal expressions based on those values. Because all Resin's resources are added to the CanDI registry automatically, application components have access to anything they need.

Both the JSP immediate syntax and deferred syntax are supported (`${...}` vs `#{...}`). Currently, there is no distinction between the two, but the deferred syntax is preferred, because CanDI initializes beans lazily to handle circular references.

Example: circular references in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
       xmlns:ee="urn:java:ee">
       xmlns:qa="urn:java:qa">

  <qa:FooBean>
    <ee:Named>a</ee:Named>
    <qa:bar>#{b}</qa:bar>
  </qa:FooBean>

  <qa:FooBean>
    <ee:Named>b</ee:Named>
    <qa:bar>#{a}</qa:bar>
  </qa:FooBean>

</web-app>
```

You can also use beans as factories in the EL expressions, because Resin's EL implementation allows method expressions. If the bean's create method is named `create()` , the EL expression will look something like `#{foo.create()}` .

13.5 Property configuration

Resin's Java Injection configuration uses the standard JavaBeans patterns to configure properties. Resin uses the same mechanism for all of its own configuration parsing, including every JavaEE configuration file, the resin-web.xml and the resin.xml itself. So your application will have all the configuration flexibility it needs.

Since the component beans can use Java Injections, injected components are typically not configured in the resin-web.conf, avoiding the need for tags like `<ref>` .

Example: Hello.java

```
package example;

public class Hello {
  public void setGreeting(String greeting);
}
```

The basic example sets a `greeting` property of a hello, world bean. Resin will apply the configuration to the instance as part of the creation process.

Example: META-INF/beans.xml configuring a singleton

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:example="urn:java:example">

  <example:Hello>
    <example:greeting>Hello, World</example:greeting>
  </example:Hello>

</beans>
```

13.5.1 primitive conversions

Resin automatically converts XML values to the Java property types for most primitive values. For other primitive types, it also supports the JavaBeans `PropertyEditor` and custom converters.

Table 13.2: Built-in String Converters

TYPE	EXAMPLES
String	hello, world
boolean	true, false
numbers	3, -4, 3.14, 9.3e-20
char	a, b
String[]	foo, bar, baz
Class	com.foo.MyBean
Path	file:/var/data/file
File	/var/data/file
URL	http://hessian.caucho.com/test
Pattern	a+[bcd]*
Date	2009-07-14 10:13
Properties	a=value
Enumeration	RED, BLUE

13.5.1.1 enumerations

Enumerations are automatically converted from their string representation.

13.5.2 compound types

Full sub-bean configuration is also available when a service needs a more complicated configuration than primitives allow. The service class can add sub-beans as properties and the sub-beans themselves are configured recursively using Resin's configuration. Since all JavaEE configuration files like the web.xml and the *.tld files are configured using Resin's recursive sub-bean configuration, your application has full access to a powerful recursive configuration.

A sophisticated application can use Resin's sub-bean configuration to create a full domain-specific language, allowing extensive user control of the application.

13.5.2.1 custom sub-beans

Example: sub-bean configuration example

```
<web-app xmlns="http://caucho.com/ns/resin">

  <example:Theater xmlns:example="urn:java:example">
    <example:name>Balboa</example:name>

    <example:movie title="The Princess Bride"/>

    <example:movie title="The Maltese Falcon"/>
  </example:Theater>

</web-app>
```

In this example, the `Theater` is configured with multiple `Movie` classes, each added with the `addMovie` method.

Example: Theater.java

```
public class Theater {
  public void setName(String name);

  public void addMovie(Movie movie)
}
```

Example: Movie.java

```
public class Movie {
  public void setTitle(String title);
}
```

13.5.2.2 list

Setters taking a `List` or array argument can be configured with list values.

List items are specified directly with <value> elements. There is no extra <list> element required. The <list> element is only used when creating a sub-list or sub-element (see below.)

Example: MyBean.setValues(List)

```
<my-bean>
  <values>
    <value>a</value>
    <value>b</value>
    <value>c</value>
  </values>
</my-bean>
```

Example: MyBean.setValues(String [])

```
<my-bean>
  <values>
    <value>a</value>
    <value>b</value>
    <value>c</value>
  </values>
</my-bean>
```

In the following example, the argument is an object, so we need a <list> element to tell Resin to create a list. The object created will be an `ArrayList`.

Example: MyBean.setValues(Object)

```
<my-bean>
  <values>
    <list>
      <value>a</value>
      <value>b</value>
      <value>c</value>
    </list>
  </values>
</my-bean>
```

Resin can always use the `addXXX` pattern to add a variable number of items. Normally, the `addXXX` pattern is easier and more maintainable than the `addList` pattern. In particular, validation of the item values is quicker and more accurate with `addXXX`.

Example: MyBean.addValue(String)

```
<my-bean>
  <value>a</value>
  <value>b</value>
  <value>c</value>
</my-bean>
```

13.5.2.3 map

Generic maps can use an <entry> syntax to define property values.

Example: MyBean.setValues(Map)

```
<my-bean>
  <values>
    <entry key="a" value="one"/>
    <entry key="b" value="two"/>
    <entry key="c" value="three"/>
  </values>
</my-bean>
```

13.6 Constructors

Some beans require constructor configuration because the service or library designer prefers constructors to method configuration. Resin's configuration can support these constructor beans with the

`<new>` syntax.

Example: MyBean configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:example="urn:java:example">

  <example:MyBean>
    <new>
      <value>first arg<value>
      <value>second arg<value>
    </new>
  </example:MyBean>

</web-app>
```

Example: MyBean with constructor

```
public class MyBean {
  public MyBean(String a, String b)
  {
    ...
  }
}
```

13.6.1 Single constructor

As a convenience, Resin provides short form of the constructor configuration if there's only a single argument. You can either omit the <value> and just use the <new> tag, or you can even omit the <new> tag entirely.

Example: MyBean short form

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:example="urn:java:example">

  <example:MyBean>
    <new>single arg</new>
  </example:MyBean>

</web-app>
```

Example: MyBean ultra-short form

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:example="urn:java:example">

  <example:MyBean>single arg<example:MyBean>

</web-app>
```

13.6.2 valueOf

For classes which implement a static `valueOf(String)` method, Resin will automatically convert to the given type using the `valueOf` method.

Example: MyBean with valueOf

```
public class MyBean {
  ...

  public static MyBean valueOf(String text)
  {
    MyBean bean = new MyBean();
    bean.setTextValue(text);
    bean.init();
    return bean;
  }
}
```

13.6.3 setValue

For objects with a `setValue` or `addText` method and a zero-argument constructor, Resin-CanDI will convert using the following steps: Create the object Inject any dependencies Call `setValue` or `setText` with the string Call any @PostConstruct Return the configured bean

13.7 Custom Services

Some of your application's beans will be configured as custom services, like Java servlets or Hessian services, using CanDI to configure a service annotation. These custom services combine your own Java code with Resin capabilities, usually to expose them as external web services.

Table 13.3: Custom Service Configuration

ANNOTATION	DESCRIPTION	SAMPLE
http://caucho.com/resin-4.0-javadoc/-com/caucho/remote/BamService.html	BAM service	---- <mypkg:MyBean> <resin:BamService name="my-service"/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-com/caucho/remote/-HessianClient.html	Hessian client proxy	---- <mypkg:MyApi> <resin:HessianClient url="http://localhost:8080/test"/> </mypkg:MyApi> ----
http://caucho.com/resin-4.0-javadoc/-com/caucho/remote/-HessianService.html	Hessian service	---- <mypkg:MyBean> <resin:HessianService urlPattern="/test"/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-com/caucho/jms/-JmsMessageListener.html	JMS based EJB message driven bean	---- <mypkg:MyBean> <resin:JmsMessageListener destination="my_queue"/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-javax/ejb/MessageDriven.html	EJB message driven bean	---- <mypkg:MyBean> <ee:MessageDriven> <ee:activationConfig propertyName="destination" propertyValue="my_queue"/> </ee:MessageDriven> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-javax/ejb/Startup.html	Start on initialization	---- <mypkg:MyBean> <ee:Startup/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-javax/ejb/Stateful.html	EJB stateful session bean	---- <mypkg:MyBean> <ee:Stateful/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-javax/ejb/Stateless.html	EJB stateless session bean	---- <mypkg:MyBean> <ee:Stateless/> </mypkg:MyBean> ----
http://caucho.com/resin-4.0-javadoc/-javax/servlet/annotation/-WebServlet.html	Servlet mapping	---- <mypkg:MyServlet> <ee:WebServlet value="/test"/> </mypkg:MyServlet> ----

13.8 Resin CanDI

Resin is designed around the Java Contexts and Dependency Injection specification (Java CanDI, JSR-299), an inversion-of-control framework used for all configuration and resources including servlets, EJBs, messaging, remoting, and databases. Applications can take advantage of Java Injection using standard annotations and interfaces.

Since Resin-CanDI is used for servlets, Java objects and EJBs, any application bean can use EJB annotations like @TransactionAttribute or CanDI @InterceptionTypes or event @Observes capabilities, in addition to the dependency injection and IoC configuration.

The dependency injection framework is type-safe, meaning the registry is organized around Java types, not a flat namespace, which gives more power and flexibility for component assembly. Since injection is annotation-based, most components can avoid XML configuration, while XML is still available for components.

13.9 Overview

Resin's Java Injection support is integrated with EJB 3.1 Lite and the core components like Servlets, Filters and remote objects. This integration means plain Java beans can use EJB annotations and interception, EJBs can use Java Injection annotations, and both kinds of beans can be configured directly from the `resin-web.xml`

or discovered by classpath scanning.

So it's best to think of Java Injection as a set of orthogonal capabilities that are available to any registered bean. The basic capability types are: Lifecycle model: Java, `@Stateless` , `@Stateful` , or `@MessageDriven` . Resin-managed objects like Servlets and Filters are Java model beans. Dependency injection: injection annotations from `javax.inject` : `@Default` , `@Named` , `@Qualifier` , `@EJB` , `@PersistenceUnit` , etc are available to all beans. Registration: all beans are registered in a unified typed-namespace registry (i.e. the registration half of dependency injection.) Lifecycle events: the `@PostConstruct` and `@PreDestroy` Predefined aspects: the `@TransactionAttribute` , `@RunAs` , `@RolesAllowed` , etc. annotations are available to all beans. Custom interceptors: EJB-style `@AroundInvoke` , and `@Interceptors` , as well as Java Injection `@Interceptor` , `@InterceptorBindingType` , and `@Decorator` are available to all beans. Event handling: the Java Injection `javax.enterprise.event` , `@Observes` capability is available to all beans.

13.10 XML configuration

You can register your components and services with Resin using the resin.xml or resin-web.xml files as well as the META-INF/beans.xml and WEB-INF/beans.xml. Since the Java Injection registry is integrated with Resin, your services be treated as first-class components along with the Resin resources. Although most components will not need XML, there are a few advantages for the small number of services which do use XML.

The XML-configuration lets you customize your application for a particular environment, e.g. setting configuration parameters. For example, Resin's `<database>` needs to select a database driver and configure the URL, user and password of the database as well as configuring connection pooling parameters. Some application services will also need configuration.

In addition, the XML-configuration documents the services you've enabled. For heavyweight services, this documentation is critical, while lightweight components do not need this extra housekeeping overhead.

13.10.1 bean and component registration

The `<my:MyBean>` tags register application classes with Resin. The default scope of a `<my:MyBean>` is `@Dependent` . A `<my:MyBean>` will create a new instance each time it's injected or referenced.

Example: bean and component META-INF/beans.xml

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:example="urn:java:example">

  <example:MyService>
    <ee:ApplicationScoped/>
  </example:MyService>

  <example:MyComponent>
  </example:MyComponent>

</beans>
```

The `<my:MyBean>` tags can configure fields and annotations:

Table 13.4: XML annotations

ATTRIBUTE	DESCRIPTION
<ee:Named>	the `@javax.inject.Name` annotations for EL naming
<ee:Qualifier>	any `@javax.inject.Qualifier` annotations for injection
my:myField	optional configuration, using bean-style assignment
<ee:SessionScoped>	specifies scope of the instances: request, conversation, session, application, or singleton

13.10.2 Bean property configuration

Resin's Java Injection configuration uses the standard JavaBeans patterns to configure properties. Resin uses the same mechanism for all of its own configuration parsing, including every JavaEE configuration file, the resin-web.xml and the resin.xml itself. So your application will have all the configuration flexibility it needs.

Since the component beans can use Java Injections, injected components are typically not configured in the resin-web.conf, avoiding the need for tags like `<ref>` .

Example: Hello.java

```
package example;

public class Hello {
  private String \_greeting = "default";

  public void setGreeting(String greeting) { \_greeting = greeting; }
  public String getGreeting() { return \_greeting; }
}
```

The basic example sets a `greeting` property of a hello, world bean. Resin will apply the configuration to the instance as part of the creation process.

Example: META-INF/beans.xml configuring a singleton

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:example="urn:java:example">

  <example:Hello>
    <example:greeting>Hello, World</example:greeting>
  </example:Hello>

</beans>
```

Resin's configuration uses 5 basic bean patterns, extending the JavaBeans conventions. It can configure literal values like string and integers as well as configuring other beans. Any component bean configured by Resin has full access to `@Qualifier` injection as well as the standard `@PostConstruct` annotations. Sub-beans are not automatically registered with Java Inection, i.e. they act like the servlet configuration.

(Currently the patterns are name-based like JavaBeans, since Resin was designed before annotations. We may add configuration annotations in the future.

Example: Bean configuration patterns

```
  public void setFoo(String data);

  public void setFoo(Movie data);

  public void addFoo(Movie data);

  public Movie createFoo();

  public void setText(String data);
```

`setFoo(String)` configures a standard JavaBeans-style literal. `setFoo(Movie)` creates a new instance of `Movie` and recursively configures it. `addFoo(Movie)` also creates a new instance of `Movie` and recursively configures it. `addFoo` is an easy way of configuring lists. `Movie createFoo()` lets the bean create the `Movie` instance. Many beans can use `createFoo` with inner classes to handle complex configuration. `setText` is called with the text contents of the XML. Value-style beans will use this. (somewhat rare).

As mentioned above, Resin uses these 5 patterns to handle all of the JavaEE configuration files. In particular, the `createFoo`

pattern returning inner classes is very handy for some complicated configuration cases, and for cases where a sub-tag needs information about the parent.

Example: sub-bean configuration example

```
<web-app xmlns="http://caucho.com/ns/resin">

  <example:Theater xmlns:example="urn:java:example">
    <example:name>Balboa</example:name>

    <example:movie title="The Princess Bride"/>

    <example:movie title="The Maltese Falcon"/>
  </example:Theater>

</web-app>
```

In this example, the `Theater` classes uses an inner `Movie` class to illustrate the use of the `create` pattern.

Example: Theater.java

```
public class Theater {
  String \_name;

  ArrayList<Movie> \_movies = new ArrayList<Movie>();

  public void setName(String name) { \_name = name; }

  public Movie createMovie()
  {
    return new Movie(this);
  }

  public void addMovie(Movie movie)
  {
    \_movies.add(movie);
```

```
  }

  public static class Movie {
    private Theater \_theater;
    private String \_title;

    Movie(Theater theater)
    {
      \_theater = theater;
    }

    public void setTitle(String title) { \_title = title; }
  }
}
```

13.10.3 Base configuration: string conversions

Java Injection provides a number of built-in string conversion types as well as supporting JavaBeans `PropertyEditor` and custom converters.

Table 13.5: Built-in String Converters

TYPE	DESCRIPTION
boolean, Boolean	Java boolean
byte, Byte	Java byte
short, Short	Java short
int, Integer	Java integer
long, Long	Java long
float, Float	Java float
double, Double	Java double
char, Character	Java char
String[]	String array separated by commas
Class	Java classes
Path	Resin VFS Paths
File	java.io.File
URL	java.net.URL
Pattern	java.util.regex.Pattern
Locale	java.util.Locale
Date	java.util.Date
Properties	java.util.Properties
RawString	com.caucho.config.type.RawString

13.10.3.1 enumerations

Enumerations are automatically converted from their string representation.

13.10.3.2 String constructor

Resin-CanDI will automatically convert a string to an object if the object has a single String argument constructor.

Example: MyBean with constructor

```
public class MyBean {
  public MyBean(String value)
  {
    ...
  }
}
```

13.10.3.3 valueOf

For classes which implement a static `valueOf(String)` method, Resin will automatically convert to the given type using the `valueOf` method.

Example: MyBean with valueOf

```
public class MyBean {
  ...

  public static MyBean valueOf(String text)
  {
    MyBean bean = new MyBean();
    bean.setTextValue(text);
    bean.init();
    return bean;
  }
}
```

13.10.3.4 setValue

For objects with a `setValue` or `addText` method and a zero-argument constructor, Resin-IoC will convert using the following steps: Create the object Inject any dependencies Call `setValue` or `setText` with the string Call any @PostConstruct Return the configured bean

13.10.4 list

Setters taking a `List` or array argument can be configured with list values.

List items are specified directly with <value> elements. There is no extra <list> element required. The <list> element is only used when creating a sub-list or sub-element (see below.)

Example: MyBean.setValues(List)

```
<my-bean>
  <values>
    <value>a</value>
    <value>b</value>
    <value>c</value>
  </values>
</my-bean>
```

Example: MyBean.setValues(String [])

```
<my-bean>
  <values>
    <value>a</value>
    <value>b</value>
    <value>c</value>
  </values>
</my-bean>
```

In the following example, the argument is an object, so we need a <list> element to tell Resin to create a list. The object created will be an `ArrayList` .

Example: MyBean.setValues(Object)

```
<my-bean>
  <values>
    <list>
      <value>a</value>
      <value>b</value>
      <value>c</value>
    </list>
  </values>
</my-bean>
```

Resin-CanDI can always use the `addXXX` pattern to add a variable number of items. Normally, the `addXXX` pattern is easier and more maintainable than the `addList` pattern. In particular, validation of the item values is quicker and more accurate with `addXXX` .

Example: MyBean.addValue(String)

```
<my-bean>
  <value>a</value>
  <value>b</value>
  <value>c</value>
</my-bean>
```

13.10.5 map

Generic maps can use an <entry> syntax to define property values.

Example: MyBean.setValues(Map)

```
<my-bean>
  <values>
    <entry key="a" value="one"/>
    <entry key="b" value="two"/>
    <entry key="c" value="three"/>
  </values>
</my-bean>
```

13.10.6 References and EL Expressions

Resin-CanDI configuration files can use EL expressions to get references to resources, beans, system properties, and calculate generatal expressions based on those values. Since all Resin's resources are added to the WebBeans registry automatically, application components have access to anything they need.

Both the JSP immediate syntax and deferred syntax are supported (${...} vs #{...}). Currently, there is no distinction between the two, but the deferred syntax is preferred, since Resin-IoC initializes beans lazily to handle circular references.

Example: circular references in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:ee="urn:java:ee"
  xmlns:qa="urn:java:qa">

  <qa:FooBean ee:Named="a">
    <bar>#{b}</bar>
  </qa:FooBean>
```

```
  <qa:BarBean ee:Named="b">
    <foo>#{a}</foo>
  </qa:BarBean>

</web-app>
```

Because Resin's EL implementation allows method expressions, you can use beans as factories in the EL expressions.

13.11 IoC annotations

13.11.1 @EJB

@EJB requests the injection of an EJB session bean.

```
@Target({TYPE, METHOD, FIELD})
public @interface EJB {
  String name() default "";
  Class beanInterface() default Object.class;
  String beanName() default "";
  String mappedName() default "";
}
```

13.11.2 @PersistenceContext

@PersistenceContext requests the injection of a JPA PersistenceContext.

```
@PersistenceContext(unitName="test")
EntityManager \_em;
```

```
package javax.persistence;

@Target({TYPE, METHOD, FIELD})
public @interface PersistenceUnit {
  String name() default "";
  String unitName() default "";
  PersistenceContextType type() default TRANSACTION;
  PersistenceProperty[] properties() default {};
}
```

13.11.3 @PersistenceUnit

@PersistenceUnit requests the injection of a JPA PersistenceUnit.

```
@PersistenceUnit(unitName="test")
EntityManagerFactory \_emf;
```

```
package javax.persistence;

@Target({TYPE, METHOD, FIELD})
public @interface PersistenceUnit {
  String name() default "";
  String unitName() default "";
}
```

13.11.4 @PostConstruct

javax.annotation.PostConstruct tells the assembler to call a method after the bean has been built, but before it is active.

```
package javax.annotation;

@Target(value={PACKAGE,TYPE})
public @interface PostConstruct {
}
```

13.11.5 @PreDestroy

javax.annotation.PreDestroy tells the container to call the annotated method before it is destroyed.

```
package javax.annotation;

@Target(value={PACKAGE,TYPE})
public @interface PreDestroy {
}
```

13.11.6 @Resource

@Resource tells the assembler to retrieve a resource an assign it to a field or property. Typically, the resource will be stored in JNDI.

Table 13.6: Known types

TYPE	DESCRIPTION
DataSource	Configured by < database >
UserTransaction	User XA interface provided by Resin
Executor	JDK 1.5 thread pool interface (see java.util.concurrent)

```
package javax.annotation;

@Target({TYPE, METHOD, FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Resource {
  String name() default "";

  Class<?> type() default Object.class;
  AuthenticationType authenticationType() default CONTAINER;
  boolean shareable() default true;
  String mappedName() default "";
  String description() default "";
}
```

13.12 Bean Configuration

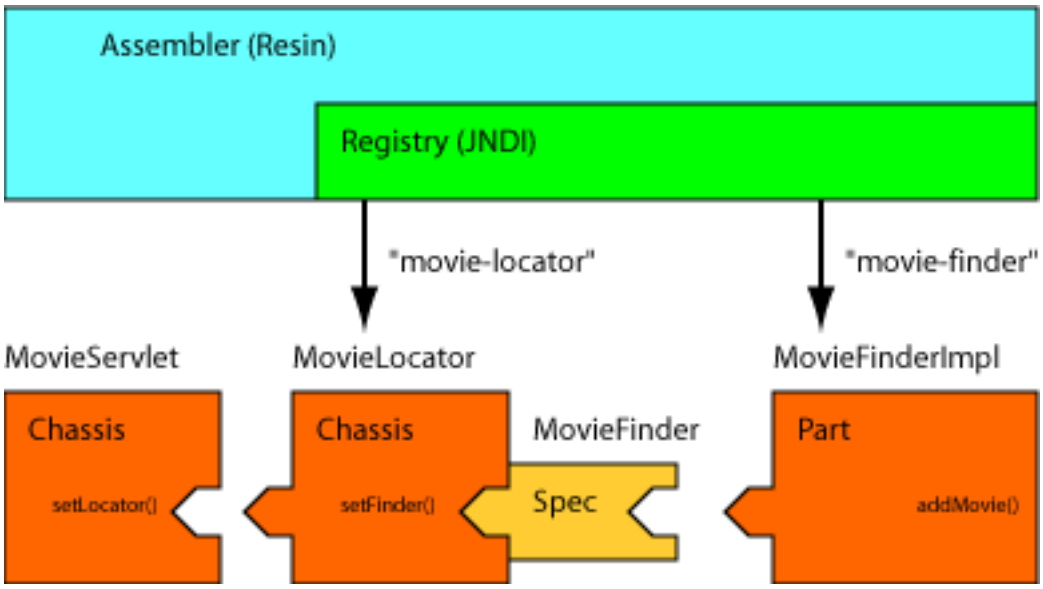
Resin configures beans using bean-style (setter injection) patterns, supporting the Inversion-of-Control design pattern. A "bean" is any plain-old Java object which follows standard configuration patterns. Because Resin can find the bean-style setters from looking at the class, it can configure those setters in a configuration file like the web.xml.

Resin's configuration follows the Assembly Line or Dependency Injection pattern.

13.13 Overview

The Assembly Line pattern gives configuration responsibility to the container where it belongs, while keeping the application code independent of the container. Bean-style configuration setters for simple properties form the foundation for the Assembly Line pattern. If an application follows the bean patterns, it can be configuration in any container following the Assembly Line (setter injection) pattern.

We strongly recommend following the Assembly Line pattern throughout an application, even if your application does not use Resin to configure itself. Following the Assembly Line pattern makes application code easier to understand, maintain, configure and test.



13.14 Property Configuration: setXXX

The bean configuration form the foundation of the Assembly Line pattern. Since most applications already follow the bean patterns, they get property configuration with no changes.

Each configuration parameter foo has a corresponding setter method

`setFoo` with a single argument for the value. Resin looks at the class using Java's reflection to find the `setFoo` method.

Bean-style configuration for a single value setter

```
<init>
  <greeting>Hello, World!</greeting>
  <another-greeting>Hello, Mom!</another-greeting>
</init>
```

Bean-style java code for a single value setter

```
public class MyBean {
  private String \_greeting;
  private String \_anotherGreeting;

  public void setGreeting(String greeting)
  {
```

```
    \_greeting = greeting;
  }

  public void setAnotherGreeting(String anotherGreeting)
  {
    \_anotherGreeting = anotherGreeting;
  }
}
```

13.14.1 Type conversion

A setter can have a parameter that has a type other than

`String` . Resin will perform any type conversion necessary, so you can use integers and doubles as well as strings.

Bean-style configuration for type conversion

```
<init>
  <host>www.gryffindor.com</host>
  <port>80</port>
</init>
```

Bean-style java code for type conversion

```
public class MyBean {
  private String \_host;
  private int \_port;

  public void setHost(String host)
  {
    \_host = host;
  }

  public void setPort(int port)
  {
    \_port = port;
  }
}
```

13.14.2 Compatibility

Property configuration is very portable. Any serious configuration system will configure bean-style properties.

13.15 Setter Injection: setXXX

Setter injection connects resources following the same bean-style setter pattern. Where bean properties configure simple values like strings and integers, setter injection configures other resources like databases and application components.

Resin uses JNDI to store the intermediate resources, e.g. storing a database in java:comp/env/jdbc/test. The configuration file specifies the JNDI resource using the JSP expression language and jndi.

Configuration for Setter Injection

```
<init>
  <data-source>\${jndi("jdbc/test")}<data-source>
</init>
```

Setter Injection for a DataSource

```
public class MyBean {
  private DataSource \_dataSource;

  public void setDataSource(DataSource ds)
  {
    \_dataSource = ds;
  }
}
```

13.15.1 Compatibility

Setter injection is portable to containers which support dependency injection.

13.16 Container Properties: addXXX

Resources often act as containers for lists of values and map values. The `addXXX` pattern adds multiple values for a single property.

A setter method `addFoo` allows multiple values to be specified from the configuration.

Bean-style configuration for setting multiple values

```
<init>
  <greeting>Hello, World!</greeting>
  <greeting>Hello, Mom!</greeting>
</init>
```

Bean-style java code for setting multiple values

```
public class MyBean {
  private LinkedList \_greetings = new LinkedList();

  public void addGreeting(String greeting)
  {
    \_greetings.add(greeting);
  }

}
```

13.17 Validation and Assembly: @PostConstruct

Well-written resources will validate their configuration and may perform additional assembly tasks. Resin calls methods marked with the @PostConstruct annotation after all the setter methods have been called.

Bean-style @PostConstruct

```
import javax.annotation.PostConstruct;

public class MyBean {
  private String \_language;
  private String \_country;
  Locale locale;

  public void setLanguage(String language)
  {
    \_language = language;
```

```
  }

  public void setCountry(int country)
  {
    \_country = country;
  }

  @PostConstruct
  public void init()
  {
    locale = new Locale(language, country);
  }
}
```

13.17.1 Validation Exceptions

If an exception is thrown from any of the methods in the bean, Resin will attach a file name and line number that correspond to the configuration file.

Bean-style exceptions

```
import java.util.Locale;
import javax.annotation.PostConstruct;

public class MyBean {
  private String \_language;
  private String \_country;
  Locale \_locale;

  public void setLanguage(String language)
    throws Exception
  {
    if (language.length() != 2)
      throw new Exception("'language' must be a two-character string");
    \_language = language;
  }

  public void setCountry(int country)
    throws Exception
  {
    if (country.length() != 2)
      throw new Exception("'country' must be a two-character string");
    \_country = country;
  }

  @PostConstruct
  public void init()
  {
    if (_country == null)
      throw new Exception("'country' is required");
    if (_language == null)
      throw new Exception("'language' is required");

    \_locale = new Locale(language,country);
  }
}
```

```
500 Servlet Exception

WEB-INF/web.xml:9: java.lang.Exception: 'country' must be a two-character string
```

13.18 Nested Beans: createXXX

Beans can be nested, allowing a bean to have setters that have other sub-beans as the type.

Bean-style configuration for sub-beans

```
<init>
  <table>
    <name>Foo</name>
    <timestamp-field>tstamp</timestamp-field>
  </table>

  <table name="Bar" timestamp-field="ts"/>
</init>
```

Bean-style java code for sub-beans

```
import javax.annotation.PostConstruct;
import javax.sql.*;

// a class to periodically clean old log records from the database
public class LogCleaner {
  List \_logTables = new LinkedList();

  // the createXXX method is optional, and allows use something other than
  // the default constructor for a sub-bean
  public LogTable createTable()
  {
    return new LogTable();
  }

  // you could also use setTable(LogTable logTable)
  public void addTable(LogTable logTable)
  {
    \_logTables.add(logTable);
  }

  public class LogTable {
    String \_name;
    String \_timestampField;

    public void setName(String name)
    {
      \_name = name;
    }

    public void setTimestampField(String timestampField)
    {
      \_timestampField = timestampField;
    }

    @PostConstruct
    public void init()
        throws Exception
    {
      if (_name == null)
        throw new Exception("'name' is required");
      if (_timestampField == null)
        throw new Exception("'timestamp-field' is required");
    }

    public void cleanTable(DataSource pool)
```

```
    {
      Connection conn = null;
      try {
        conn = pool.getConnection();
        ...
      } catch (SQLException e) {
        throw new ServletException(e);
      } finally {
        try {
          if (conn != null)
            conn.close();
        } catch (SQLException e) {
        }
      }
    }
  }

  ...

}
```

13.19 Setting with the body text

The `addText()` method will capture the body of the tag for a bean setter.

Bean-style configuration for setting with the body text

```
<init>
  <message>This is the message</message>
</init>
```

Bean-style java code for setting with the body text

```
public class MyBean {
  Message \_msg;

  public Message createMessage() { return new Message(); }

  public void setMessage(Message msg) { \_msg = msg; }

  public class Message {
    String \_text;
    public void addText(String text) { \_text = text; }
    public String getText() { return \_text; }
  }
}
```

13.20 Returning a different object

There are some unusual cases where the configured bean is just a configuration object and you want to return a different bean. The bean can implement a method `Object replaceObject()` to return a different object. Called after the @PostConstruct.

13.21 Inline custom Beans

Inline custom beans

```
<beans xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.resin"
  xmlns:example="urn:java:com.foo.example">

<example:MessageBean resin:JndiName="env/message">
  <message>This is message 1</message>
  <message>This is message 2</message>
  <message>
     <example:CustomMessage/>
   </message>
</example:MessageBean>

</beans>
```

MessageBean.java

```
package example;

import java.util.*;

public class MessageBean {
  List \_msgs = new LinkedList();

  public void addMessage(Message msg)
  {
    \_msgs.add(msg);
    System.out.println("MessageBean.addMessage(): " + msg);
  }

  // this never get's called, because MessageBean has no parent
  public void setParent(Object obj)
  {
    System.out.println("MessageBean.setParent(): " + obj);
  }

}
```

Message.java

```
package example;

public class Message {
  String \_text;
  public void addText(String text)
  {
    \_text = text;
    System.out.println("Message.addText(): " + text);
  }

  public void setParent(Object obj)
  {
    System.out.println("Message.setParent(): " + obj);
  }

  public String toString()
  {
    return super.toString() + ": " + \_text;
  }
}
```

Message.java

```
package example;

public class CustomMessage extends Message {
  public void addText(String text)
  {
    \_text = text;
    System.out.println("CustomMessage.addText(): " + text);
  }

  public void setParent(Object obj)
  {
    System.out.println("CustomMessage.setParent(): " + obj);
  }
}
```

```
Message.setParent(): example.MessageBean@ffb35e
Message.addText(): This is message 1
MessageBean.addMessage(): example.Message@1591b4d: This is message 1
Message.setParent(): example.MessageBean@ffb35e
Message.addText(): This is message 2
MessageBean.addMessage(): example.Message@10f965e: This is message 2
CustomMessage.setParent(): example.MessageBean@ffb35e
CustomMessage.addText(): This is message 3
MessageBean.addMessage(): example.CustomMessage@12164ea: This is message 3
```

In practice, it may make more sense to use `createSubBean` or

`addSubBean` to set a parent-child relationship for beans, instead of `setParent`. The possible issue with addSubBean is that @PostConstruct methods are called before addSubBean. The possible issue with createSubBean is that it's not possible to use a <mypkg:MySubBean> with createSubBean. So the setParent is useful when the @PostConstruct method needs the parent, and you need to use <mypkg:MySubBean>.

13.22 Configuring beans from XML files

The facilities provided by Resin make it very easy to read XML files. Java classes that follow the java bean pattern are defined to match the schema of the xml file.

rss-example.xml

```
<rss version="0.91">
  <channel>
    <title>Hogwarts</title>
    <link>http://hogwarts.com</link>
    <description>Hogwart's News</description>
    <image>
      <title>Hogwarts</title>
      <url>http://hogwarts.com/images/logo.gif</url>
      <link>http://hogwarts.com</link>
      <width>88</width>
      <height>31</height>
      <description>Hogwart's News</description>
    </image>
    <item>
      <title>New Course Additions</title>
      <link>http://hogwarts.com/news/00123.html</link>
      <description>New course's are now available at Hogwart's.</description>
    </item>
    <item>
      <title>Dumbledore is back!</title>
```

```
      <link>http://hogwarts.com/news/00122.html</link>
      <description>
        After a short hiatus, Professor Dumbledore is back as
        Headmaster of Hogwart's.
      </description>
    </item>
  </channel>
</rss>
```

example/rss/Rss.java

```
package example.rss;

import java.util.ArrayList;

public class Rss {
  private String \_version;
  private ArrayList<Channel> \_channels = new ArrayList<Channel>;

  public void setVersion(String version)
  {
    \_version = version;
  }

  public String getVersion()
  {
    return \_version;
  }

  public void addChannel(Channel channel)
  {
    \_channels.add(channel);
  }

  public ArrayList<Channel> getChannels()
  {
    return \_channels;
  }
}
```

example/rss/Channel.java

```
package example.rss;

import java.util.ArrayList;

public class Channel {
  private String \_title;
  private String \_link;
  private String \_description;
  private String \_language;
  private Image \_image;
  private ArrayList<Item> \_items = new ArrayList<Item>;

  public void setTitle(String title)
  {
    \_title = title;
  }

  public String getTitle()
  {
    return \_title;
```

```
  }

  public void setLink(String link)
  {
    \_link = link;
  }

  public String getLink()
  {
    return \_link;
  }

  public void setDescription(String description)
  {
    \_description = description;
  }

  public String getDescription()
  {
    return \_description;
  }

  public void setImage(Image image)
  {
    \_image = image;
  }

  public Image getImage()
  {
    return \_image;
  }

  public void addItem(Item item)
  {
    \_items.add(item);
  }

  public ArrayList<Items> getItems()
  {
    return \_items;
  }
}
```

example/rss/Image.java

```
package example.rss;

public class Image {
  private String \_title;
  private String \_url;
  private String \_link;
  private int \_width;
  private String \_height;
  private String \_description;

  public void setTitle(String title)
  {
    \_title = title;
  }

  public String getTitle()
  {
```

```
    return \_title;
  }

  public void setLink(String link)
  {
    \_link = link;
  }

  public String getLink()
  {
    return \_link;
  }

  public void setWidth(int width)
  {
    \_width = width;
  }

  public int getWidth()
  {
    return \_width;
  }

  public void setHeigth(int height)
  {
    \_height = height;
  }

  public int getHeight()
  {
    return \_height;
  }

  public void setDescription(String description)
  {
    \_description = description;
  }

  public String getDescription()
  {
    return \_description;
  }
}
```

example/rss/Item.java

```
package example.rss;

public class Item {
  private String \_title;
  private String \_link;
  private String \_description;

  public void setTitle(String title)
  {
    \_title = title;
  }

  public String getTitle()
  {
    return \_title;
  }
```

```
  public void setLink(String link)
  {
    \_link = link;
  }

  public String getLink()
  {
    return \_link;
  }

  public void setDescription(String description)
  {
    \_description = description;
  }

  public String getDescription()
  {
    return \_description;
  }
}
```

13.22.1 NodeBuilder

com.caucho.config.NodeBuilder is used to configure beans from an xml file.

NodeBuilder constructs beans from rss-example.xml

```
  import com.caucho.config.NodeBuilder;
  import com.caucho.vfs.Path;

  ...

  Rss rss = new Rss();
  NodeBuilder builder = new NodeBuilder();

  Path rssPath = new Path("WEB-INF/rss-example.xml");

  builder.configure(rss,rssPath);
```

13.22.2 RNC validation

RNC validation

```
  import com.caucho.config.NodeBuilder;
  import com.caucho.vfs.Path;

  ...

  Rss rss = new Rss();
  NodeBuilder builder = new NodeBuilder();

  Path rssPath = new Path("WEB-INF/rss-example.xml");
  Path schemaPath = new Path("WEB-INF/rss.rnc") builder.setCompactSchema(schemaPath);

  builder.configure(rss,rssPath);
```

13.22.3 resin:import

resin:import is used to read configuration information from another file.

rss-example.xml

```
<rss version="0.91">
 <resin:import>
   <fileset dir="channels/">
     <include name="*.xml"/>
   </fileset>
  </resin:import>
```

channels/hogwarts.xml

```
<channel>
  <title>Hogwarts</title>
  <link>http://hogwarts.com</link>
  <description>Hogwart's News</description>
  <image>
    <title>Hogwarts</title>
    <url>http://hogwarts.com/images/logo.gif</url>
    <link>http://hogwarts.com</link>
    <width>88</width>
    <height>31</height>
    <description>Hogwart's News</description>
  </image>
  <item>
    <title>New Course Additions</title>
    <link>http://hogwarts.com/news/00123.html</link>
    <description>New course's are now available at Hogwart's.</description>
  </item>
  <item>
    <title>Dumbledore is back!</title>
    <link>http://hogwarts.com/news/00122.html</link>
    <description>
      After a short hiatus, Professor Dumbledore is back as
      Headmaster of Hogwart's.
    </description>
  </item>
</channel>
```

13.22.3.1 See also

config-el-ref.xtp#resin:import http://wiki4.caucho.com/Building_a_simple_listing_in_JSP#BookListServlet_listing

13.23 Administration

Table 13.7: Administration

NAME	DESCRIPTION
<resin:JmxService>	Enables cluster-wide JMX administration.
<resin:LogService>	Stores high-priority log messages in a database.
<resin:PingMailer>	Mails a notification when a ping check fails.
<resin:PingThread>	Checks status of Resin sutomatically.
<resin:StatService>	Gathers timed runtime status of Resin for graphing.
<resin:XaLogService>	Transaction log service.
<resin:RemoteAdminService>	Enables administration by remote agents, like an eclipse console.

13.24 Clustered Caching (JCache)

Table 13.8: Caches

NAME	DESCRIPTION
<resin:ByteStreamCache>	Distributed cache of InputStream/OutputStream byte streams across a cluster pod.
<resin:ClusterCache>	JCache-style distributed object cache across a cluster pod (java.util.Map).
<resin:GlobalCache>	JCache-style distributed object cache across the entire Resin system.

13.25 JMS

Table 13.9: JMS

NAME	DESCRIPTION
<resin:JmsConnectionFactory>	Combined Queue and Topic ConnectionFactory.
<resin:ClusterQueue>	Clustered queue.
<resin:FileQueue>	Filesystem based queue.
<resin:FileTopic>	Filesystem based topic.
<resin:MemoryQueue>	Memory based queue.
<resin:MemoryTopic>	Memory based topic.

13.26 Protocols

Table 13.10: Protocols

NAME	DESCRIPTION
<resin:FastCgiPort>	FastCGI requests, e.g. from nginx.

13.27 Rewrite

Table 13.11: Dispatch rules

NAME	DESCRIPTION
<resin:Dispatch>	Normal servlet dispatching with optional target rewriting.
<resin:FastCgiProxy>	Proxies the request to a backend server using FastCGI as a proxy protocol.

Table 13.11: (continued)

NAME	DESCRIPTION
<resin:Forbidden>	Send a HTTP forbidden response.
<resin:Forward>	Forwards to the new URL using RequestDispatcher.forward with the target URL.
<resin:HttpProxy>	Proxies the request to a backend server using HTTP as a proxy protocol.
<resin:LoadBalance>	Load balance to a cluster of backend Resin servers.
<resin:Redirect>	Send a HTTP redirect to a new URL specified by target .
<resin:SendError>	Send a HTTP error response.
AbstractTargetDispatchRule	Base class for custom dispatch rules.

Table 13.12: Rewrite filters

NAME	DESCRIPTION
<resin:SetHeader>	Sets a response header.
<resin:SetRequestSecure>	Marks the request as secure.
<mypkg:MyFilter>	Servlet filters.

13.28 Repository

Table 13.13: Repository

NAME	DESCRIPTION
<resin:ProjectJarRepository>	Maven-style library jar management for webapps.

13.29 Request Conditions

Table 13.14: Basic conditions

NAME	DESCRIPTION
<resin:IfAuthType>	Checks for the authentication type, request.getAuthType().
<resin:IfCookie>	Checks for the presence of a named HTTP cookie from request.getCookies().
<resin:IfCron>	Matches if the current time is in an active range configured by cron-style times.
<resin:IfFileExists>	Matches if the URL corresponds to an actual file.
<resin:IfHeader>	Tests for a HTTP header and value match.
<resin:IfLocale>	Tests for a Locale match from the HTTP request.
<resin:IfLocalPort>	Compares the local port of the request, request.getLocalPort().
<resin:IfMethod>	Compares the HTTP method, request.getMethod().
<resin:IfNetwork>	Compares the remote IP address to a network pattern like 192.168/16.

Table 13.14: (continued)

NAME	DESCRIPTION
<resin:IfQueryParam>	Tests for a HTTP query parameger, request.getParameter().
<resin:IfRemoteAddr>	Tests against the remote IP address, request.getRemoteAddr().
<resin:IfRemoteUser>	Tests against the remote user, request.getRemoteUser().
<resin:IfSecure>	True for SSL requests, i.e. if request.isSecure() is true.
<resin:IfUserInRole>	Tests is the user is in the servlet security role.
RequestPredicate	Interface for custom request predicates.

Table 13.15: Combining conditions

NAME	DESCRIPTION
<resin:And>	Matches if all children match.
<resin:Or>	Matches if any children match.
<resin:Not>	Matches if the child does not match.
<resin:NotAnd>	Matches if any child does not match.
<resin:NotOr>	Matches if all the children do not match.

13.30 Scheduling

Table 13.16: Scheduling

NAME	DESCRIPTION
<resin:ScheduledTask>	cron-style task scheduling.

13.31 Security

Table 13.17: Authenticators

NAME	DESCRIPTION
<resin:AdminAuthenticator>	Resin administration authentication (same syntax as XmlAuthenticator).
<resin:DatabaseAuthenticator>	Authentication using a JDBC database schema.
<resin:JaasAuthenticator>	Java authentication service authenticator.
<resin:LdapAuthenticator>	LDAP authentication using JNDI.
<resin:PropertiesAuthenticator>	.properties file authentication.
<resin:XmlAuthenticator>	.xml file authentication.
AbstractAuthenticator	Abstract class for custom authentication.

Table 13.18: Single Signon

NAME	DESCRIPTION
<resin:ClusterSingleSignon>	Cluster-based single signon.
<resin:MemorySingleSignon>	Memory-based single signon.

Table 13.19: Login managers

NAME	DESCRIPTION
<resin:BasicLogin>	HTTP basic authentication.
<resin:DigestLogin>	HTTP digest authentication.
<resin:FormLogin>	Servlet form authentication.
AbstractLogin	Abstract class for custom login.

Table 13.20: Authorization rules

NAME	DESCRIPTION
<resin:Allow>	Allows access to a URL pattern.
<resin:Deny>	Denies access to a URL pattern.

Table 13.21: Permission Mapping

NAME	DESCRIPTION
<resin:XmlRoleMap>	Role to group permission mapping.
AbstractRoleMap	Abstract class for custom role to group permission mapping.

13.32 Examples

13.32.1 Servlet/Filter initialization

EL expressions can be used to configure servlets and filters. init-param values can use JSP EL expressions, including the ability to use system properties.

Servlets, filters, and resources can be configured like beans with setter methods are called directly (See config-candi.xtp .

One example use for the bean-style servlet initialization is to avoid JNDI lookup inside the servlet code. For example, a servlet that that uses a JDBC DataSource might look like:

Servlet using JDBC

```
package test;

...
```

```
public class TestServlet extends HttpServlet {
  private DataSource \_dataSource;

  /**
   * Bean setter is called to configure the servlet
   * before the init() method.
   */
  public void setDataSource(DataSource dataSource)
  {
    \_dataSource = dataSource;
  }

  ...
}
```

The servlet is configured as follows:

Example configuration

```
<web-app>
  <allow-servlet-el/>

  <servlet servlet-name='test'
           servlet-class='test.TestServlet'>
    <init>
      <data-source>${jndi("java:comp/env/jdbc/test")}</data-source>
    </init>
  </servlet>

  ...
</web-app>
```

The `<data-source>` xml tag corresponds to the

`setDataSource` method of the bean. More infomation on this powerful pattern is in the config-candi.xtp section of the documentation.

13.32.2 Environment variables

Environment variables inherited by the process from the operting system are available as variables in el expressions, for example
.

13.32.3 fmt.sprintf()

Format a string using a sprintf-like format string.

```
fmt.sprintf(format[,arg1, arg2 ... argN])
```

PARAMETER	DESCRIPTION	DEFAULT
format	the format string (see below)	required
arg1..argN	the values used for the conversions in the format string	n/a

`sprintf` accepts a series of arguments, applies to each a format specifier from 'format', and returns the formatted data as a string. 'format' is a string containing two types of objects: ordinary characters (other than '*%), which are copied unchanged to the output, and conversion specifications, each of which is introduced by* `%. (To include `%' in the output, use '%%' in the format string).

A conversion specification has the following form:

```
%[FLAGS][WIDTH][.PREC][TYPE]
```

TYPE is required, the rest are optional.

The following TYPE's are supported:

CODE	MEANING
%%	a percent sign
%c	a character with the given number
%s	a string, a null string becomes "#null"
%z	a string, a null string becomes the empty string ""
%d	a signed integer, in decimal
%o	an integer, in octal
%u	an integer, in decimal
%x	an integer, in hexadecimal
%X	an integer, in hexadecimal using upper-case letters
%e	a floating-point number, in scientific notation
%E	a floating-point number, like %e with an upper-case "E"
%f	a floating-point number, in fixed decimal notation
%g	a floating-point number, in %e or %f notation
%G	a floating-point number, like %g with an upper-case "E"
%p	a pointer (outputs a value like the default of toString())

Intepret the word 'integer' to mean the java type long. Since java does not support unsigned integers, all integers are treated the same.

The following optional FLAGS are supported:

CODE	MEANING
0	If the TYPE character is an integer leading zeroes are used to pad the field width instead of spaces (following any indication of sign or base).
+	Include a '+' with positive numbers.
(a space)	use a space placeholder for the '+' that would result from a positive number
-	The result of is left justified, and the right is padded with blanks until the result is 'WIDTH' in length. If you do not use this flag, the result is right justified, and padded on the left.
#	an alternate display is used, for 'x' and 'X' a non-zero result will have an "0x" prefix; for floating point numbers the result will always contain a decimal point.
j	escape a string suitable for a Java string, or a CSV file. The following escapes are applied: " becomes \", newline becomes \n, return becomes \r, \ becomes \\.
v	escape a string suitable for CSV files, the same as 'j' with an additional " placed at the beginning and ending of the string
m	escape a string suitable for a XML file. The following escapes are applied: < becomes <, > becomes > & becomes & ' becomes &#039, " becomes &034;

The optional WIDTH argument specifies a minium width for the field. Spaces are used unless the '0' FLAG was used to indicate 0 padding.

The optional PREC argument is introduced with a '*., and gives the maximum number of characters to print; or the minimum number of digits to print for integer and hex values; or the maximum number of significant digits for* `g and `G'; or the number of digits to print after the decimal point for floating points.

13.32.4 fmt.timestamp()

Format a timestamp string.

```
fmt.timestamp(format[,date])
```

PARAMETER	DESCRIPTION	DEFAULT
format	the format string (see below)	required
date	an object with java.util.Date or java.util.Calendar or com.caucho.util.QDate	the current date and time

```
msg="The current date and time is ${fmt.timestamp('%Y/%m/%d %H:%M:%S.%s')}"
msg="time=${fmt.timestamp('[%Y/%m/%d %H:%M:%S.%s]')}"
```

`format` contains regular characters, which are just copied to the output string, and percent codes which are substituted with time and date values.

CODE	MEANING
%a	day of week (short)
%A	day of week (verbose)
%b	day of month (short)
%B	day of month (verbose)
%c	Java locale date
%d	day of month (two-digit)
%H	24-hour (two-digit)
%I	12-hour (two-digit)
%j	day of year (three-digit)
%m	month (two-digit)
%M	minutes
%p	am/pm
%S	seconds
%s	milliseconds
%W	week in year (three-digit)
%w	day of week (one-digit)
%y	year (two-digit)
%Y	year (four-digit)
%Z	time zone (name)
%z	time zone (+/-0800)

13.32.5 host

Table 13.22: host properties

VARIABLE	MEANING
name	The name of the host
regexp	Regular expression values for host regexp matches
root	The root directory of the host
url	The canonical url of the host

13.32.6 Example

```
<host regexp="www.([^.]+).com">
  <root-directory>/opt/www/${host.regexp[1]}</root-directory>

  <context-param server-id="${server.name}"/>

  <web-app id="/">
    <document-directory>webapps/ROOT</document-directory>
  </web-app>
</host>
```

13.32.7 java

Table 13.23: java properties

VARIABLE	MEANING
version	The JDK version

13.32.8 jndi()

The configuration EL supports the static function jndi:lookup. jndi:lookup can be used to lookup a JNDI value for the configuration.

configuring JNDI

```
<servlet servlet-name='foo'
         servlet-class='qa.FooServlet'>
  <init>
    <data-source>${jndi:lookup("java:comp/env/jdbc/test")}</data-source>
  </init>
</servlet>
```

13.32.9 resin

Table 13.24: resin properties

VARIABLE	MEANING
address	The local IP address
conf	Path to the configuration file
home	The the location of the Resin executables
hostName	The local hostname as returned by InetAddress
root	The location of the content, specified at startup with --root-directory
serverId	The identity of the active <server>, specified at startup with --server
version	The resin version, e.g. 3.1.0

13.32.10 server

Values related to the active < > .

Table 13.25: server properties

VARIABLE	MEANING
address	the bind address of the cluster and load balancing port
id	The identity of the active <server>, specified at startup with --server
port	the cluster and load balancing port
httpAddress	the bind address of the http listener, INADDR_ANY for all addresses
httpPort	the port number of the http listener
httpsAddress	the bind address of the ssl http listener, INADDR_ANY for all addresses
httpsPort	the port number of the ssl http listener

13.32.11 System properties

System properties are available as variables in el expressions. Many system property names are not valid el identifiers; in that case the system variable is used, for example

`${system[java.io.tmpdir]}` .

A full list of standard java system properties is provided in the javadoc for java.lang.System .

Table 13.26: Standard system properties

VARIABLE	MEANING
java.io.tmpdir	Default temp file path
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

13.32.12 webApp

Table 13.27: webApp properties

VARIABLE	MEANING
name	The name of the web-app
contextPath	The context path of the web-app
regexp	Regular expression values for web-app regexp matches

Table 13.27: (continued)

VARIABLE	MEANING
root	The root directory of the web-app
url	The canonical url of the web app

Chapter 14

Database

14.1 See also

http://wiki.caucho.com/ contains sample configurations for several database drivers http://wiki4.caucho.com/Java_EE_Servlet_tutorial_:
Using http://wiki4.caucho.com/Java_EE_Servlet_tutorial_:_Adding_MySQL_and_JDBC_to_bookstore_example#BookRepositoryJDB

14.2 Basic Configuration

A basic <database> configuration specifies the following: The JNDI name where the configured DataSource will be stored The database driver's main class The driver-specific url for the database. Any user and password information.

Example: mysql configuration

```
<web-app xmlns="http://caucho.com/ns/resin">

<database jndi-name='jdbc/test_mysql'>
  <driver type="com.mysql.jdbc.Driver">
    <url>jdbc:mysql://localhost:3306/test</url>
    <user></user>
    <password></password>
  </driver>
</database>

</web-app>
```

This <database> will configure a javax.sql.DataSource and store it in JNDI at java:comp/env/jdbc/test_mysql. To use the data source, follow the database use pattern in the http://wiki4.caucho.com/Java_EE_Servlet_tutorial_:_Adding_MySQL_and_JDBC_to_boo
config.22.2Fresin.xml

Although some deployments will specify driver and connection pool parameters, the default values will be fine for most applications.

14.3 Core Concepts

14.3.1 Connection

An established channel of communication between a client and a server. The client and the server may be on separate machines, on the same machine, or even running in the same JVM. Often the connection is established using TCP/IP as the transport of communication.

A database connection is used to allow the Java program, running in a JVM, to communicate with a database server.

14.3.2 Connection Pool

A set of connections maintained so that the connections can be reused when there is a future need for the conneciton.

Connection pools are used to reduce the overhead of using a database. Establishing a connection to the database is a costly operation. A connection pool keeps a pool of open connections, each connection can be used for a time as needed, and then released back to the pool. A connection that has been released back to the pool can then be reused.

Connection pooling is especially important in server applications. The overhead of opening a new connection for each new client request is too costly. Instead, the database pool allows for a connection to be opened once and then reused for many requests.

14.3.3 DataSource

A JDBC term (and interface name) used for a factory that is used to obtain connections.

Resin provides an implementation of `DataSource` . Resin's implementation of `DataSource` is a connection pool.

14.3.4 Driver

An implemetation of a defined interface that hides the details of communication with a device or other resource, such as a database.

A Driver provides an interface and is responsible for the communication with the database. Every different database (i.e Oracle, MySQL) has their own means of enabling communication from the client (in this case Resin and you applications) and the database. The Driver provides a common interface that hides the details of that communication.

14.3.5 Transaction

A transaction is used to mark a group of operations and provide a guarantee that all of the operations happen, or none of them happen. Transactions protect the integrity of the database.

Transactions are especially important in server applications where many threads of processing may be interacting with the database at the same time.

For a simple example, imagine a set of operations that reads a value, calculates a new value, and then updates the database.

Example: simple set of database operations

```
read value A=1
calculate  A=A+1
update     A=2

read value A=2
calculate  A=A+1
update     A=3
```

Imagine if one thread is performing this operation, and in the middle of this read/calculate/update, another thread performs an update. The data that the first thread obtained from the read and is using for the calculation and update is no longer valid.

Example: 2 Threads with database race condition

```
 Thread 1                 Thread 2
 --------                 --------
 read value A=1           read value A=1
 calculate  A=A+1         calculate A=A+1
                         update A=2
 update     A=2
```

Placing the read/calculate/update operations in a transactions guarantees that only one thread can perform those operations at a time, if a second thread comes along and tries to perform the operation, it will have to wait for the first thread to finish before it can begin.

Example: 2 Threads protected with transactions

```
Thread1                Thread 2
-------                --------
read value A=1
calculate  A=A+1       (tries to read A, but has to wait for thread 1)
update     A=2
                       read value A=2
                       calculate A=A+1
                       update A=3
```

14.3.6 Distributed Transaction

A distributed transaction is a transaction that involves more than one connection.

If the guarantees that transactions apply need to apply to operations that occur on two databases within the same transaction, distributed transactions are needed.

If `A` and `B` in the following example are in two different databases, then a distributed transaction is needed:

Example: Simple set of database operations

```
read value db1.A=1
read value db2.B=99
calculate  A=A+1
calculate  B=B-A
update     db1.A=2
update     db2.B=97
```

Distributed transactions are rarely needed, and few databases really support them.

14.4 Core Configuration

14.4.1 database

Configure a database resource, which is a database pool that manages and provides connections to a database.

ATTRIBUTE	DESCRIPTION	DEFAULT
jndi-name	JNDI name to store the pool under. Servlets, jsp, and other java code use this name. The path is relative to `java:comp/env`	
driver	Configure the database #driver .	
backup-driver	Configure a backup #driver .	
max-connections	#pooling - maximum number of allowed connections	1024
max-idle-time	#pooling - maximum time an idle connection is kept in the pool	30 sec
max-active-time	#pooling - maximum time a connection allowed to be active	6 hours
max-pool-time	#pooling - maximum time a connection is kept in the pool	24 hours
connection-wait-time	#pooling - how long to wait for an idle connection (Resin 1.2.3)	30 sec

ATTRIBUTE	DESCRIPTION	DEFAULT
max-overflow-connections	#pooling - how many "overflow" connection are allowed if the connection wait times out.	1024
ping-table	#reliability - The database table used to "ping", checking that the connection is still live.	n/a
ping	#reliability - test for live connections before allocating them from the pool.	false
ping-interval	#reliability - set the minimum interval for pings, instead of doing a ping every time	1s
prepared-statement-cache-size	A cache that holds prepared statements, a reused prepared statement avoids the overhead of the driver making the prepared statement	0
spy	A debugging aid, if true, generate `info` level log events that reveal the SQL that is used with the connections.	false

All times default to seconds, but can use longer time periods:

Table 14.1: Time suffixes

SUFFIX	DESCRIPTION
s	seconds
m	minutes
h	hours
D	days

The class that corresponds to <database> is com.caucho.sql.DBPool

14.5 Driver Configuration

14.5.1 driver

Configure a database driver . The driver is a class provided by the database vendor, it is responsible for the communication with the database.

The jar file with the driver in it can be placed in `WEB-INF/lib` , although it is often best to place your datbase driver's jar file in

`$RESIN_HOME/lib/local/` , which makes the driver available to all of your web applications.

The class that corresponds to <driver> is com.caucho.sql.DriverConfig

ATTRIBUTE	DESCRIPTION
type	The Java class name of the database driver.
	url
The driver specific database url.	
user	The username to give the database driver.
	password
The password to give the database driver.	
init-param	Set #init-param not known to Resin.

14.5.2 Choosing a driver class for <type>

Database vendors usually provide many different classes that are potential candidates for type . The JDBC api has developed over time. The driver you choose depends on the options the vendor offers, and whether or not you need distributed transactions.

14.5.2.1 JDBC 2.0 - ConnectionPoolDataSource

JDBC 2.0 defined the interface ConnectionPoolDataSource . A

`ConnectionPoolDataSource` is not a connection pool, but it does provide some extra information that helps Resin to pool the connection more effectively.

A driver that implements ConnectionPoolDataSource is better than a JDBC 1.0 driver that implements Driver .

14.5.2.2 JDBC 2.0 - XADataSource

JDBC 2.0 defined the interface XADataSource for connections that can participate in distributed transactions . A distributed transaction is needed when transactions involve multiple connections. For example, with two different database backends, if the guarantees that transactions apply need to apply to operations that occur on both databases within the same transaction, distributed transactions are needed.

Distributed transactions are rarely needed, and few databases really support them. Some vendors will provide `XADataSource` drivers even though the database does not really support distributed transactions. Often,

`XADataSource` drivers are slower than their

`ConnectionPoolDataSource` counterparts.

`XADataSource` should only be used if distributed transactions are really needed, and can probably be safely ignored for most applications.

14.5.2.3 JDBC 1.0 - Driver

Driver is the original JDBC interface, and is the least desirable kind of driver to use. Resin can still pool database connections using these drivers, but it will not be as efficient as the newer drivers.

14.5.3 Set driver properties with init-param

init-param is used to set properties for java.sql.Driver database drivers. More modern drivers like ConnectionPoolDataSource will not use init-param, but will use IoC-style tags directly.

For example, MySQL drivers accept the `useUnicode` parameter, if true the driver will use Unicode character encodings when handling strings.

Example: mysql configuration

```
<database>
  <jndi-name>jdbc/mysql</jndi-name>
  <driver>
    <type>com.mysql.jdbc.Driver</type>
    <url>jdbc:mysql://localhost:3306/dbname</url>
    <user>username</user>
    <password>password</password>

    <init-param useUnicode="true"/>
  </driver>
  ...
</database>
```

Example: mysql ConnectionPoolDataSource

```
<database>
  <jndi-name>jdbc/mysql</jndi-name>
  <driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://localhost:3306/dbname</url>
    <user>username</user>
    <password>password</password>

    <useUnicode/>
  </driver>
  ...
</database>
```

14.6 Pooling Configuration

Pooling configuration controls the behaviour of Resin's pooling of database connections. For most applications and databases the only needed change is to increase the max-connections value to meet high demand. Other pooling parameters have defaults that are based on our years of experience with many different databases in many different applications. Changes from the defaults should only be done in response to specific problems, and with a good understanding of how pooling works.

14.7 Reliability Configuration

14.7.1 ping

Resin's database pool can test if the pooled database connection is still alive by configuring a ping query. This is typically only necessary if the #pooling parameters are changed from their default values.

If the pool is configured with a long max-idle-time the database connection may become stale if the database is restarted, or if the database is configured with a shorter connection timeout value than the configuration of the Resin pool. Normally when a database connection is returned to the pool it will wait there until the next request or the idle-time expires. If the database goes down in the meantime or closes the connection, the connection will become stale. The ping configuration can test the database connection.

When pinging, Resin's DBPool will test a table specified with the ping-table parameter before returning the connection to the application. If the ping fails, the connection is assumed to be no good and a different connection from the pool is returned. For a ping-table of BROOMS, Resin will use the query `select 1 from BROOMS where 1=0`

Example: <ping> configuration

```
  <database jndi-name="...">
    <driver type="...">
      ...
    </driver>

    <ping>true</ping>
    <ping-table>BROOMS</ping-table>
  </database>
```

You can test the ping using the following steps: Configure the database with ping-table and ping. Execute some servlet that queries the database. Restart the database server. Execute another servlet that queries the database.

14.7.2 <driver> list

If there is a pool of database servers available that can be used for database operations, Resin can be configured with a list of <driver> tags. Resin uses a round robin algorithm to cycle through the list of drivers when obtaining connections. If a particular

<driver> fails to provide a connection, Resin continues the attempt to obtain a connection. If all of the configured drivers fail to provide a connection the exception is propogated to the caller.

Example: A <driver> list

```
<database jndi-name="jdbc/hogwarts">
  <driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 110 :3306/hogwarts</url>
    ...
  </driver>

  <driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 111 :3306/hogwarts</url>
    ...
  </driver>
  ...
</database>
```

14.7.2.1 <backup-driver> list

Drivers in a driver list can be marked as backups. The drivers configured with <backup-driver> are used only if all of the drivers configured with <driver> have failed.

Each time a new connection is needed Resin goes through the process of first attempting to use one of the <driver> configured drivers to get a connection, and if that fails then the <backup-driver> are used. A new connection is needed from the driver if the pool of connections that is maintained by Resin does not contain an idle connection. The #pooling and the usage pattern of the application determine how often a connection is obtained from a driver. The pooling configuration typically allows a single real connection to be reused by the application many times.

The lifetime of a connection obtained from a <backup-driver> is determined by the #pooling , thus even if the main <driver> becomes available again a connection previously obtained from a <backup-driver> will continue to be used until it expires from the pool.

Example: A <backup-driver> list

```
<database jndi-name="jdbc/hogwarts">
  <driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 110 :3306/hogwarts</url>
    ...
  </driver>

  <driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 111 :3306/hogwarts</url>
    ...
  </driver>

  <backup-driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 112 :3306/hogwarts</url>
    ...
  </backup-driver>

  <backup-driver>
    <type>com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource</type>
    <url>jdbc:mysql://192.168.0. 113 :3306/hogwarts</url>
    ...
  </backup-driver>
  ...
```

```
</database>
```

14.8 Obtaining and using a database connection

14.8.1 Getting the DataSource

The `DataSource` is a factory that is used to obtain a connection. The `DataSource` is obtained using the jndi-name specified when configuring the database resource.

Ideally, the JNDI lookup of `DataSource` is done only once, the

`DataSource` obtained from the lookup can be stored in a member variable or other appropriate place. The stored `DataSource` can then be used each time a connection is needed. If it is not stored, there will be an impact on performance from having to do the lookup each time you want to get a connection.

Example: Obtaining a DataSource

```
import javax.sql.*;
import javax.webbeans.*;

public class .... {
  @Named("jdbc/test") DataSource \_pool;

  ...
}
```

14.8.2 Getting a Connection

A connection is obtained from the `DataSource` . The connection is used as needed, and then released with a call to close() so that Resin knows it is available for a subsequent request.

It is very important that the `close()` is always called, even if there as an exception. Without the `close()` , Resin's database pool can loose connections. If you fail to close() a connection, Resin does not know that it is available for reuse, and cannot allocate it for another request. Eventually, Resin may run out of connections.

The following example shows the use of a `finally` block that contains the `close()` . Because the `close()` is in a finally block, it will happen even if the code using the connection throws an exception.

Example: Getting a connection from the DataSource

```
package javax.webbeans.*;
package javax.sql.*;

public class MyBean()
{
  @In DataSource \_pool;

  public void doStuff()
  {
    Connection conn = null;
    try {
      conn = \_pool.getConnection();

      Statement stmt = conn.createStatement();

      ResultSet rs = stmt.executeQuery(" ... ");

      ...
```

```
      rs.close();
      stmt.close();
    } catch (SQLException e) {
      throw new ServletException(e);
    } finally {
      try {
        if (conn != null)
          conn.close();
      } catch (SQLException e) {
      }
    }
  }
}
```

14.8.3 Getting the underlying driver connection

The connection obtained by `pool.getConnection()` is an instance of com.caucho.sql.UserConnection .

`UserConnection` is a wrapper around the real driver connection, it allows Resin to intercept the close() call and manage the underlying driver connection.

In rare circumstances it is necessary to obtain the real connection returned by the driver. Typically this is a requirement for situations where the driver provides a specialized API that is not available with the standard JDBC API.

Example: Getting the underlying driver connection

```
Connection driverConn = ((com.caucho.sql.UserConnection) connection).getConnection();

// never do this: driverConn.close()
```

14.9 Protecting the database password

Resin provides facilities that allow you to plugin your own custom code that returns a password to Resin. However any solution is vulnerable, unless you require a person to type in a password every time Resin starts (or restarts). Typically the security of the machine hosting Resin, and proper permissions on the readability of the resin.xml file, are sufficient to protect your database password.

The solution shown below is not really secure because you can disassemble the Password code to get the decryption key, but it may be marginally better than plaintext.

Example: password encryption

```
<driver type="...">
  <password xmlns:hogwarts="urn:java:com.hogwarts">
    <hogwarts:Password value="mX9aN9M=="/>
  </password>
  ...
```

You will need to provide com.hogwarts.Password:

Example: Password class

```
package com.hogwarts;

public class Password {
  private String \_value;

  public void setValue(String value)
  {
```

```
    \_value = value;
  }

  public Object replaceObject()
  {
    return decrypt(_value);
  }

  private String decrypt(String encrypted)
  {
    ... custom code ...
  }
}
```

This solution is completely general, you can use <mypkg:MyClass/> anywhere in the configuration files where a string value is allowed.

Resin does not provide the equivalent of com.hogwarts.Password because it's not really secure. Providing that kind of solution would lead some to believe it was a secure solution.

Chapter 15

Deployment

15.1 Overview

Resin's .war application deployment can be as simple as copying a .war file to a webapps/ directory on a local machine, and as powerful as cloud deployment, archiving, staging, rollback. cloud deployment: Resin's cloud deployment will distribute a new web-application to all servers in the cloud, using a transactional store to ensure consistency. activation control: deployment and activation can be controlled independently, allowing applications to be deployed and verified on all servers, and then activated at once or on a rolling-server basis. archiving and rollback: deployed web-applications can be archived and quickly rolled-back if a deployment needs to be reversed. staging: an application can be deployed to the cluster as a "preview" stage, which will be used only by a preview server. graceful version upgrading: web-applications can be deployed with versions, letting current user continue with the old version and moving new users to the new version. command-line: web-applications can be deployed to the cloud from the command-line for scriptable deployment. browser: for convenience, applications can also be deployed from a browser using the /resin-admin site.

15.2 Basic Deployment Methods

15.2.1 Webapps Directory Deployment

For a simple deployment, you can copy a .war archive file containing your application to a webapps directory. Resin will detect the .war archive, expand it, and start serving requests.

The webapps directory is configured and enabled by the <web-app-deploy/> tag in the resin.xml.

Example: web-app-deploy in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">
  ...
  <host id="">

    <web-app-deploy path="webapps"
expand-preserve-fileset="WEB-INF/work/**"/>

  </host>
</cluster>
</resin>
```

The expand-preserve-fileset attribute lets you keep files for a redeployment, which can improve restart times. Normally, Resin will delete all files for an update, including the compiled files for JSP, which forces Resin to recompile JSP files, even if they haven't changed in the update. If you add the expand-preserve-fileset , Resin will only recompile the JSP files that have changed.

15.2.2 Command-Line Deployment

Command-line deployment uses the same resin.xml <web-app-deploy> configuration as a webapps deployment and expands the archive to the same directory. Instead of looking for the .war file in a directory, Resin will look in an internal repository using the web-app's identifier.

Example: command-line deployment

```
unix> resinctl deploy test.war
```

The default deployment is to the default host with the war's name as a prefix. Both can be changed with deploy options.

15.2.3 Command-Line vs Directory Priority

The command-line deployment takes priority over the directory deployement, because a deployment to a cluster ensures that all servers have the same version of the application. This means you must undeploy a web-app from the command-line if you decide to switch from command-line deployment to directory deployment.

Example: command-line undeployment

```
unix> resinctl undeploy test.war
unix> cp test.war webapps
```

15.2.4 Deployment Expansion

The <web-app-deploy> controls the expansion based on web-app ids like "production/webapps/default/test" which is the same as the web-app identifier. The deployment expansion process looks like the following: Look in the internal repository for "production/webapps/default/test" (uploaded by command-line). If an archive exists in the internal repository, skip to step #4. If that fails, look for a webapps/test.war If the webapps/test.war exists, copy the test.war into the internal repository as "server/[server-id]/webapps/default/test" Use the archive in "production/webapps/default/test" (or "server/[server-id]/webapps/default/test") as the repository source archive. Delete the old webapps/test directory (saving some directories when expand-preserve-fileset is configured.) Expand the repository source archive files into the webapps/test directory. Restart the webapp

The deployment identifier matches the web-app id that Resin logs at startup time. The identifier a repository tag that lets Resin have a general cloud repository system and also handle web-app deployments, versioning, and staging.

For the example web-app tag "production/webapp/default/test", the "production" is the deployment stage, "webapp" is because it's a webapp deployment, "default" is the virtual-host name and "test" is the web-app name.

15.3 Cloud Deployment

Resin deploys a web-application to all servers in the cluster when you deploy using the command-line or the browser. This process happens automatically and does not require any configuration beyond Resin's normal cluster configuration.

This cloud deployment does not occur when you deploy using the filesystem by placing a .war in a webapps directory. Cloud deployment only occurs on the command-line or browser.

Resin replicates the deployed application to all three triad servers, or to all available server if you have less than three servers. Any server beyond the triad will copy the deployed application from the triad. Normally, you don't need to be aware of the triad, except to make sure at least one of the first three servers is always available.

When you add a new dynamic server or restart a server, the server will check with the triad and update to the most recent copy of the application. The system is reliable if servers crash or even if a server crash or network outage occurs during deployment. Since the deployment is transactional, the new version is only activated when every file has been copied and verified on the server.

15.4 Archiving and Rollback

Resin's deployment supports archiving and rollback as part of the underlying system. Like the Subversion version-control-system, you can copy the tag for an application to a unique name. So you can permanently save your webapp as "archive/webapp/foo/2011-05-19" and later rollback to that version if it becomes necessary.

The deployment system associates each .war archive with a tag. A deployed application might look like "production/webapp/my-host.com/my-app". The tag name is the same as the WebApp name in the Resin logs. To archive an application, you copy the tag to an archive name like "archive/webapp/my-app/2011-05-09".

To rollback a version, just copy the archive tag back to the web-app tag.

The following command-line example deploys a war first to the archive and then makes it live using a deploy-copy. The "-stage archive" selects the archive tag.

Example: archive deploy and update

```
unix> resinctl deploy -stage archive -version 1.2.13 foo.war
Deployed archive/webapp/default/foo-1.2.13 from /tmp/caucho/qa/foo.war \
  to http://127.0.0.1:8087/hmtp

unix> resinctl deploy-copy \
  -source-stage archive -source-version 1.2.13 -source foo \
  -target foo
```

Internally, the archived war files are stored with a different tag name. In the previous example, the repository might contain three tags: the archive foo-1.2.12, the archive foo-1.2.13, and the production webapp, which is a copy of foo-1.2.13.

Example: repository tags for archive

```
archive/webapp/default/foo-1.2.12   -- archived foo.war contents
archive/webapp/default/foo-1.2.13   -- archived foo.war contents
production/webapp/default/foo       -- production foo.war (a copy of 1.2.13)
```

The current web-app production/webapp-default/foo can be rolled-back by copying it from a previous version.

Example: rolling back to foo.1.2.12

```
unix> resinctl deploy-copy \
  -source-stage archive -source-version 1.2.12 -source foo \
  -target foo
```

15.5 Staging

As a final quality check before making a deployment live, you can deploy the application to a staging "preview" server in the deployment cluster. The preview version will only be visible on the preview server until you copy it to a live machine.

Example: deploying a preview stage

```
unix> resinctl deploy -stage preview foo.war

unix> resinctl start -stage preview -elastic-server -server dyn1 -cluster app-tier
```

Internally, Resin stores the preview stage as a repository tag preview/webapp/default/foo . When you deploy the .war with -stage preview , Resin save the deploy under the preview tag name. When you start Resin with -stage preview , you ask it to look in the preview tag for the web-app.

Example: repository tags for staging

```
preview/webapp/default/foo        -- preview foo.war contents
production/webapp/default/foo     -- production foo.war contents
```

After you've verified that the previewed application works, you can deploy it to production using the deploy-copy.

Example: deploying a preview as production

```
unix> resinctl deploy-copy \
  -source-stage preview -source foo \
  -target foo
```

15.6 Activating deployed applications

Applications can be activiated independent of their deployment, letting you upload a new deployment to a cloud, and then activating all servers at once or creating a rolling activation.

By default, activation is automatic. When an application is deployed, it is automatically activated. You can change the default behavior by setting the startup-mode and redeploy-mode for the web-app-deploy.

15.6.1 Startup and Redeploy Modes

The startup-mode is used in a number of places to determine the behaviour of a resource when the server starts.

The startup-mode has three values: "automatic", "lazy", and "manual". automatic - starts the resource on server start (default) lazy - starts the resource on the first use manual - waits for JMX to start the resource

The redeploy-mode is used in a number of places to determine the behaviour of a resource when it is replaced or modified while the server is running.

The redeploy-mode has two values: "automatic", and "manual". automatic - restarts the resource when it is modified (web.xml, *.class, *.war, etc). manual - waits for JMX to restart the resource when changes occur.

15.6.2 Deploying to a live server without interruption

It may be possible to deploy a web application to a live server without interruption to service if certain conditions are met. The session objects for your users are being persisted. The usage of session scoped objects between the old version and the new is compatible. The usage of application scoped objects between the old version and the new is compatible. Database schema changes are not required.

Resin allows you to have a backup instance running. The idea is that this backup instance of Resin takes over if your primary Resin instance goes down.

If you are using a load balancer to distribute your load to multiple primary servers, each primary server has a backup server.

You can use this feature to deploy to a live server without interruption of service. shutdown primary server(s) (backup server(s) takes over) deploy new war to primary server(s) start primary server(s). As soon as the primary server starts, the user will be using the new version of the application. deploy new war to backup server(s)

15.7 Versioning and Graceful Upgrades

Resin can deploy multiple versions of a web-app simultaneously, simplifying any application upgrades. The old version of the web-app will continue to receive old sessions, while the new version will get the new requests. So any user will see a consistent version as the web site update occurs with no downtime required.

The versioning requires <web-app-deploy>, i.e. it works with the webapps directory. The versioning is numerically-based, allowing dotted notation, to determine the most recent version. A simple deployment process might use foo-101 to upgrade from foo-100 . A more complicated one might use foo-10.3.14 to upgrade from foo-10.3.13 .

The versioning attribute of the <web-app-deploy> enables versioning:

Example: resin.xml for webapps versioning

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">
<host id="">

  <web-app-deploy path="webapps" versioning="true"
expand-preserve-fileset="WEB-INF/work/**"/>

</host>
</cluster>
</resin>
```

15.8 Deployment Methods

15.8.1 Filesystem deployment: Custom web-app with .war file

In this scenario, you want to configure a web-app with a specific root-directory and specify the location of the .war file. As usual, when Resin sees any changes in the .war file, it will expand the new data into the root-directory and restart the web-app. This capability, gives sites more flexibility where their directories and archive files should be placed, beyond the standard webapps directory.

The optional `archive-path` argument of the <web-app> will point to the .war file to be expanded.

Table 15.1: web-app deployment options

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-path	path to the .war file which contains the web-app's contents	
dependency-check-interval	how often Resin should check for changes in the web-app for a redeployment	2s
expand-preserve-fileset	a set of files/directories Resin should preserve on a redeploy when it deletes the expanded directory	
id	unique identifier for the web-app and the default context-path value	
redeploy-check-interval	how often Resin should check the .war file for changes	60s
redeploy-mode	how Resin should handle redeployment: automatic, lazy, or manual	automatic
root-directory	path to the expanded web-app directory	id as a sub-directory of the virtual-hosts's root

Example: resin.xml for custom web-app

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">
  <host id="">

    <web-app id="/foo" root-directory="/var/resin/foo"
expand-preserve-fileset="WEB-INF/work/**"
archive-path="/usr/local/stage/foo.war"/>
```

```
  </host>
</cluster>
</resin>
```

15.8.2 Command-Line Remote Deployment

The "deploy" command deploys a .war file to the default virtual host. In a cloud environment, Resin will copy the deployed .war to all servers in the cluster. Since the AdminAuthenticator requires a user and password, you'll need to pass those arguments.

Example: deploy hello.war

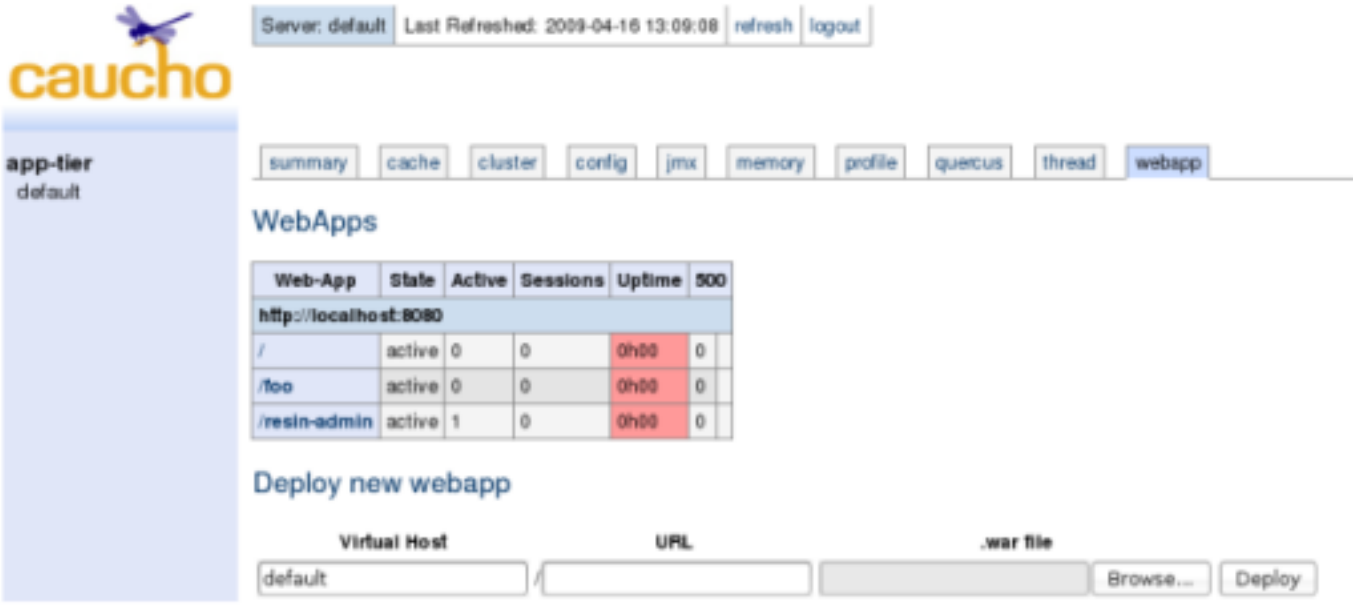
```
unix> resinctl deploy hello.war \
           -user foo -password test
```

15.8.3 Browser-based Remote Deployment

As of Resin 4.0.0, it is now possible to deploy web applications remotely to a shared repository that is distributed across the cluster. This feature allows you to deploy once to any triad server and have the application be updated automatically across the entire cluster. When a new clustering-overview.xtp joins the cluster, the triad will populate it with these applications as well.

To deploy an application remotely: log into the resin-admin console on any triad server. Make sure you are connecting over SSL, as this feature is not available over a non-encrypted channel. Browse to the "webapp" tab of the resin-admin server and at the bottom of the page, enter the virtual host, URL, and local .war file specifying the web application, then press "Deploy".

The application should now be deployed on the server. In a few moments, all the servers in the cluster will have the webapp.


caucho
Server: default
Last Refreshed: 2009-04-16 13:09:08
refresh
logout
app-tier
default
summary
cache
cluster
config
jmx
memory
profile
quercus
thread
webapp
WebApps
Web-App
State
Active
Sessions
Uptime
500
http://localhost:8080
/
active
0
0
0h00
0
/foo
active
0
0
0h00
0
/resin-admin
active
1
0
0h00
0
Deploy new webapp
Virtual Host
URL
.war file
default
/
Browse...
Deploy

Chapter 16

Deployment: Cloud

16.1 Deployment

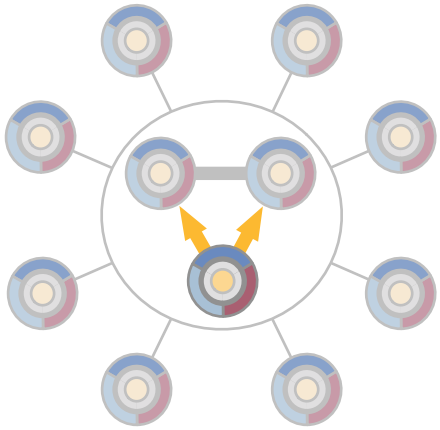
16.1.1 Cluster Deployment

To deploy an application to your cluster, use the same command-line deploy as you would for a single server. The deployment process is the same because Resin treats a standalone server as single server in a cluster.

Example: command-line deployment

```
unix> bin/resin.sh deploy test.war
```

That command-line deploy will send the test.war to the cluster's triad-server repository, and then copy the repository to all three servers in the triad hub. If you have only two servers in the cluster, Resin will copy the application to both. Once all three triad hub servers have the deployed .war, Resin will update all the spoke servers in the cluster.



The cluster command-line deployment uses the <web-app-deploy> tag in the resin.xml to configure and control where the deployed application should be expanded. Typically, the deployment will use the webapps/ directory.

Example: web-app-deploy in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">
  ...
  <host id="">
```

```
    <web-app-deploy path="webapps"
expand-preserve-fileset="WEB-INF/work/**"/>

  </host>
</cluster>
</resin>
```

When you're using virtual hosts, you'll add a -host tag to specify the virtual host to deploy to.

The default deployment is to the default host with the war's name as a prefix. Both can be changed with deploy options.

Example: virtual host deployment

```
unix> bin/resin.sh deploy -host www.foo.com test.war
```

16.1.2 Controlling Restarts

By default, a Resin server will detect an updated application automatically and restart the web-app immediately. You can delay the restart by putting it on manual control. In manual mode, Resin will only look for a new version when you use a command-line restart-webapp .

Example: command-line to restart the web-app

```
unix> bin/resin.sh restart-webapp test
```

The manual control is configured by setting <restart-mode< to manual in the web-app-deploy:

Example: web-app-deploy in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">
  <host id="">

    <web-app-deploy path="webapps"
                restart-mode="manual"
                expand-preserve-fileset="WEB-INF/work/**"/>

  </host>
</cluster>
</resin>
```

16.1.3 Zero Downtime Deployment (Versioning)

You can configure Resin's cluster deployment in a versioning mode where users gracefully upgrade to your new application version. Since new user sessions use the new version and old user sessions use the old application version, users will not need to be aware of the version upgrade.

By default, Resin restarts the web-app on a new deployment, destroying the current user sessions before starting them on the new deployment. You can change that behavior by setting versioning to true and deploying with a -version command-line option.

Example: web-app-deploy with versioning

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="">
  <host id="">

    <web-app-deploy path="webapps"
                versioning="true"
```

```
                  expand-preserve-fileset="WEB-INF/work/**"/>

  </host>
</cluster>
</resin>
```

For versioning to work, you'll deploy with a named version of your application. Resin will send new sessions to the most recent version and leave old sessions on the previous version.

Example: command-line deploy with versioning

```
unix> bin/resin.sh deploy -version 2.1.3 test.war
```

Internally, the application repository has both versions active.

Example: internal repository tags

```
production/webapp/default/test-2.1.2
production/webapp/default/test-2.1.3
```

16.2 Deployment Reliability

Resin's deployment system is designed around several reliability requirements. Although the user-visible system is simple, the underlying architecture is sophisticated — we're not just copying .war files. Predictable - all servers run the same deployed application by design, whether the server has restarted, been taken off line for maintenance, started and stopped for dynamic load management, or started from scratch from a fresh VM image. Transactional (all or nothing) - all of the update files are copied and verified in the background before the web-app restarts. While the update is occurring, Resin continues to serve the old application. Even if a network glitch occurs or a server restarts before the upgrade completes, Resin will continue to use the old web-app. Replicated - all deployments are replicated to all three servers in the triad hub. If a triad server restarts, it will update itself to the latest repository version from the backup servers. As long as at least one triad server is available, the active servers will have access to the latest repository. Elastic - the system supports dynamic adding and removal of servers. A new spoke server will contacts the triad hub for the latest application deployment and update itself. Staging, Archiving, and Versioning - the deployment system supports these through naming conventions of the deployed tag, allowing multiple versions of the same web-app to be saved in the repository and deployed as appropriate. Straightforward - the user-view of cloud deployment needs to be as simple as a single-server deployment. It needs to look simpler than it is. It needs to just work.

16.3 Deployment Architecture

The following is a description of the underlying architecture of Resin's deployment system. It's not necessary to understand or even read any of this section to use Resin's deployment. But for those who are curious, some details might be interesting.

16.3.1 .git control system architecture

The main repository is based on the distributed version control system .git, which is used for large programming projects like the Linux kernel. The .git format gives Resin the key transactional repository to make the cloud deployment reliable.

Each file in the repository is stored by its secure document hash (SHA-1). The secure hash lets Resin verify that a file is completely copied without any corruption. If verifying the hash fails, Resin will recopy the file from the triad or from the deploy command. Since the file is not saved until it's validated, Resin can guarantee that the file contents are correct.

Files are never overwritten in in Resin's repository. It's essentially write-only. Two versions of the same file are saved as two separate file: a test.jsp (version 23) which replaces a test.jsp (version 22). So there's never a case where an older version of the file can be partially overwritten.

Since the repository itself is organized as a .git self-validating file, its own updates are validated before any changes occur. Essentially, Resin verifies every file in a repository update, and then verifies every directory, and then verifies the repository itself

before making any changes visible. Resin detects that a new repository version is available (it continues to use the old repository) by checking with the triad. It checks for any new file updates and copies the new files from the triad (Resin continues to use the old files and repository.) When all the new files are verified, it copies and verifies the new directories and archives from the traid (Resin continues to use the old files and repository.) It now copies and verifies the top-level repository changes. (Resin continues to use the old files and repository.) After everything is verified on the local filesystem, Resin switches to the new repository.

If at any point a servers stops, or the network fails, or a new file is corrupted in a partial transfer, Resin continues to use the old files. On recovery, Resin will verify and delete any partially copied files, and continue the repository update. Only the repository system itself knows that an update is in process; the rest of Resin continues to use the old repository files.

16.3.2 Repository tag system

Internally, the repository is organized by tags where each tag is an archive like a .war. The tag system enables versioning and archiving since each tag can point to an archive or two tags can point to the same archive.

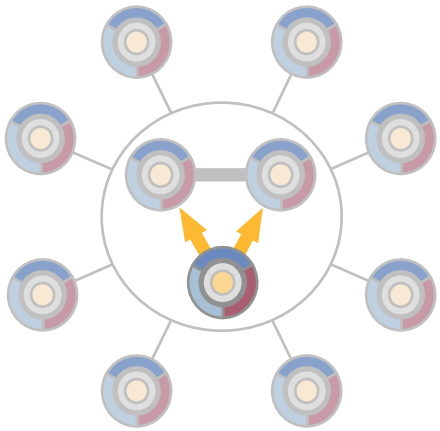
The current application for the "foo" web-app would have the tag production/webapp/default/foo . The tag points to a version of the archive, say the foo.war that was deployed on 2011-08-15 at 10:13:00. If you deploy a new foo.war, the same tag will point to the new foo.war that was deployed on 2011-08-16 13:43:15. The repository treats the two versions as entirely different archives and saves both of them.

The tag system lets you copy a current deployment to an archive tag or copy a preview staged application to the production application. You can copy the production/webapp/default/foo tag to archive/webapp/default/foo-20110815 , archiving it. If you're familiar with subversions tags and branches, this is a similar system.

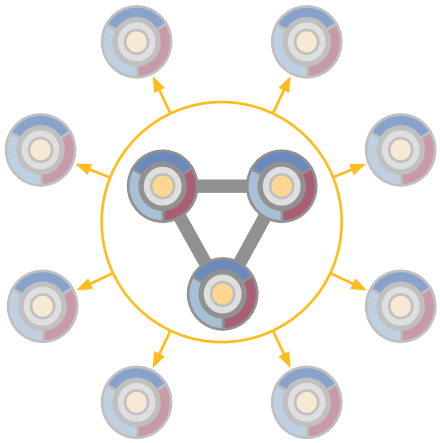
If you want to rollback to a previous version, you can copy the archived tag to the current production tag. Resin will run through the repository update system and ensure that all servers in the cloud see the updates.

16.3.3 Cloud Deployment

Deploying to a cloud extends the transactional repository to all the servers in a cluster. In Resin's replicated hub-and-spoke model, a deployment copies the archive first to all three servers in the triad. (If you have two servers, it will copy to the second server.) Since all three servers have a copy of the entire repository, your system keeps it reliability even if one server is down for maintenance and a second server restarts for an unexpected reason.



After all three servers in the hub have received and verified the deployment update, the triad hub can send the changes to all of the spoke servers.



If a spoke server restarts or a new spoke server is added to the cloud dynamically, it will contacts the hub for the most recent repository version. So even a new virtual-machine image can receive the most recent deployments without intervention.

Chapter 17

Deployment: Command-Line

17.1 resin.xml requirements

For security reasons, Resin's deployment must be enabled in the resin.xml. The default behavior is to disable deployment.

The configuration has four requirements: Start the RemoteAdminService to enable remote administration. Add an AdminAuthenticator to protect deployment with a password. Start the DeployService itself to enable remote deployment. A web-app-deploy to provide a location for the deployed .war files.

Example: minimal resin.xml for deployment

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="app-tier">
    <resin:AdminAuthenticator password-digest="none">
      <user name="foo" password="test"/>
    </resin:AdminAuthenticator>

    <resin:RemoteAdminService/>
    <resin:DeployService/>

    <server id="" port="6800">
       <http port="8080"/>
    </server>

    <host id="">
      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>
    </host>
  </cluster>
</resin>
```

17.2 command-line deployment

17.2.1 deploy for default host

The "deploy" command deploys a .war file to the default virtual host. Since the AdminAuthenticator requires a user and password, you'll need to pass those arguments as well.

Example: deploy hello.war

```
unix> bin/resin.sh deploy hello.war \
          -user foo -password test
```

17.2.2 undeploy for default host

The "undeploy" command removes a .war from the default virtual host. Since the AdminAuthenticator requires a user and password, you'll need to pass those arguments as well.

Example: undeploy hello

```
unix> bin/resin.sh undeploy hello \
          -user foo -password test
```

17.2.3 deploy with specified host

The "deploy" command allows a "-host" argument to specify a virtual host.

Example: deploy www.example.com hello.war

```
unix> bin/resin.sh deploy -host www.example.com hello.war \
          -user foo -password test
```

17.2.4 undeploy for www.example.com

The "undeploy" command removes a .war from the virtual host.

Example: undeploy www.example.com hello

```
unix> bin/resin.sh undeploy -host www.example.com hello \
          -user foo -password test
```

17.3 command-line deployment

Command line deployment capabilities were introduced in Resin 4.0.14. The set of commands allows deploying, undeploying, listing applications deployed on the server and controlling application lifecycle.

17.3.1 Synopsis of the provided commands and options

Table 17.1: commands

COMMAND	DESCRIPTION
deploy	deploys an application archive
undeploy	un-deploys an application specified by a context
deploy-list	lists all applications deployed on a server
deploy-copy	copies an application from one context to another
start-webapp	starts web application context
stop-webapp	stops web application context
restart-webapp	restarts web application context

Table 17.2: common options

ARGUMENT	MEANING	DEFAULT
-conf	configuration file	conf/resin.xml
-address	ip or host name of the server	taken from conf/resin.xml
-port	server http port	taken from conf/resin.xml
-user	user name used for authentication to the server	none, required
-password	password used for authentication to the server	none, required
-m	commit message	none, optional

17.3.1.1 deploying application

Deploying an application is done with a `deploy` command

```
bin/resin.sh [-conf <file>] deploy [options] <war-file>
```

deploying an application from a hello-world.war archive

```
unix> bin/resin.sh deploy -user admin -password secret /projects/hello-world/hello-world. ↩
    war

Deployed production/webapp/default/hello-world as hello-world.war to http://127.0.0.1:8080/ ↩
    hmtp
```

Table 17.3: deploy options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-name	name of the context to deploy to, defaults to war-file name	[/foo].war
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

17.3.1.2 listing deployed applications

Listing deployed applications is done with a `deploy-list` command

```
bin/resin.sh [-conf <file>] deploy-list [options]
```

.

```
unix> bin/resin.sh deploy-list -user admin -password secret

production/webapp/default/hello-world
```

17.3.1.3 copy application from context */hello-world* to context */foo*

Copying an applicaiton is done with a `deploy-copy` command

```
bin/resin.sh [-conf <file>] deploy-copy [options]
```

.

```
bin/resin.sh deploy-copy -user admin -password secret -source hello-world -target foo

copied production/webapp/default/hello-world to production/webapp/default/foo
```

Table 17.4: deploy-copy options

ARGUMENT	MEANING	DEFAULT
-source	context to copy application from	none
-source host	host to copy application from	default
-source-stage	source stage	production
-source-version	version of the source application formatted as <major.minor.micro.qualifier>	none
-target	context to copy application to	none
-target-host	host to copy an application to	default
-target-stage	target stage	production
-target-version	version application to use for a target, formatted as <major.minor.micro.qualifier>	none

17.3.1.4 undeploying application

Undeploying an application is done with an `undeploy` command

```
bin/resin.sh [-conf <file>] undeploy [options] <name>
```

.

```
unix> bin/resin.sh undeploy -user admin -password secret undeploy foo

Undeployed foo from http://127.0.0.1:8080/hmtp
```

Table 17.5: undeploy options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

17.3.1.5 starting application

Starting an application is done with a `start-webapp` command

```
unix> bin/resin.sh [-conf <file>] start-webapp [options] <name>
```

start web application deployed at context /foo

```
unix> bin/resin.sh start-webapp -user admin -password secret foo

'production/webapp/default/foo' is started
```

Table 17.6: start-webapp options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

17.3.1.6 stopping application

Stopping an application is done with an `stop-webapp` command

```
bin/resin.sh [-conf <file>] stop-webapp [options] <name>
```

stop web application deployed at context /foo

```
unix> bin/resin.sh stop-webapp -user admin -password secret foo

'production/webapp/default/foo' is stopped
```

Table 17.7: stop-webapp options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

17.3.1.7 restarting application

Restarting an application is done with an `restart-webapp` command

```
bin/resin.sh [-conf <file>] restart-webapp [options] <name>
```

stop web application deployed at context /foo

```
unix> bin/resin.sh restart-webapp -user admin -password secret foo

'production/webapp/default/foo' is restarted
```

Table 17.8: restart-webapp options

ARGUMENT	MEANING	DEFAULT
-host	virtual host to make application available on	default
-stage	specifies stage for staging an application	production
-version	version of application formatted as <major.minor.micro.qualifier>	none

Chapter 18

Health: Checks and Meters

18.1 Configuration

Health configuration is an extension of the standard Resin configuration file resin.xml. Because Resin uses ../admin/config-candi.xtp to create and update Java objects, each XML tag exactly matches either a Java class or a Java property. As a result, the HealthSystem JavaDoc and the JavaDoc of the various checks , actions , and predicates help to supplement the documentation as much as this reference.

18.1.1 health.xml

Resin version 4.0.16 and later includes health.xml as a standard Resin configuration file alongside resin.xml and app-default.xml .

health.xml is imported into resin.xml as a child of `<cluster>` or `<cluster-default>` .

Example: importing health.xml into resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">
  <cluster-default>
    ...
    <!--
       - Admin services
      -->
    <resin:DeployService/>

    <resin:if test="${resin.professional}">
      <resin:AdminServices/>
    </resin:if>

    <!--
       - Configuration for the health monitoring system
      -->
    <resin:if test="${resin.professional}">
      <resin:import path="${__DIR__}/health.xml" optional="true"/>
    </resin:if>
    ...
  </cluster-default>
</resin>
```

Example: simple health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthFatal/>
  </health:Restart>

</cluster>
```

18.1.2 health: namespace

health.xml introduces a new XML namespace, health: , defined by `xmlns:health="urn:java:com.caucho.health"` .

`health:` separates health objects from standard `resin:`

elements for clarity and performance. The packages references by

`health:` are: com.caucho.health com.caucho.health.check com.caucho.health.action com.caucho.health.predicate com.caucho.heal

18.1.3 Health check naming

18.1.3.1 ee: namespace

The ee: namespace is used for naming objects, for example

`ee:Named="someName"` , so that they may be referred to by name later in the configuration. This is sometimes necessary as some health conditions permit referencing a specific health check, as demonstrated in the following example.

Example: referencing named objects

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck ee:Named="pingJspCheck">
    <url>http://localhost:8080/test-ping.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:Restart>
    <health:IfHealthCritical healthCheck="${pingJspCheck}"/>
    <health:IfRechecked/>
  </health:Restart>

</cluster>
```

In this example, an instance of HttpStatusHealthCheck is named *pingJspCheck* and referred to by name in the IfHealthCritical criteria using an EL expression. The Restart action will only trigger if the health status is CRITICAL for this specific health check and no others.

18.1.3.2 Default names

All health checks classes are annotated with `@Named` , and therefore have a default name that corresponds to their bean name. For example

`<health:CpuHealthCheck/>` can be referred to by without the use of `ee:Named` .

Example: default health check name

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:CpuHealthCheck>
    <warning-threshold>95</warning-threshold>
  </health:CpuHealthCheck>

  <health:DumpThreads>
    <health:IfHealthWarning healthCheck="${cpuHealthCheck}"/>
  </health:DumpThreads>

</cluster>
```

18.1.3.3 Duplicate names

Duplicate health check names are not permitted. Resin will fail to startup due to invalid configuration in this case. This can be caused by configuring duplicate checks without using `ee:Named` , or by configuring more than one check with the same name. The following examples demonstrate both illegal cases.

Example: illegal unnamed duplicate checks

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck">
    <url>http://localhost:8080/test1.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:HttpStatusHealthCheck">
    <url>http://localhost:8080/test2.jsp</url>
  </health:HttpStatusHealthCheck>

</cluster>
```

In the preceding example, use of `ee:Named` is required.

Example: illegal duplicate names

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck" ee:Named="healthCheck">
    <url>http://localhost:8080/test1.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:CpuHealthCheck ee:Named="healthCheck">
    <warning-threshold>95</warning-threshold>
  </health:CpuHealthCheck>

</cluster>
```

In the preceding example, the health check names must be different, regardless of the type of check.

18.1.4 Default health configuration

If for any reason you are missing health.xml, for example you are upgrading from an older version of Resin and don't have the health.xml import in resin.xml, there's no need to worry. Resin creates some checks by default regardless of the presence of health.xml. Furthermore, Resin will detect if no checks are configured and setup default actions and conditions.

18.1.4.1 Standard health checks

The following health checks are considered critical to standard operation and thus will be created by Resin regardless of the presence of health.xml. If you wish to disabled any of these standard health checks, configure the check in health.xml and set the attribute

`enabled="false"` . #health:JvmDeadlockHealthCheck #health:MemoryTenuredHealthCheck #health:MemoryPermGenHealthCh
#health:CpuHealthCheck #health:TransactionHealthCheck #health:HealthSystemHealthCheck

18.1.4.2 Default actions

If any #Health checks are configured besides the standard checks mentioned above, Resin will assume the user is using health.xml and will not setup any #Health actions . If however health.xml is missing or empty, the following basic actions will be created.

```
<health:Restart>
  <health:IfHealthFatal/>
</health:Restart>
```

18.2 Health checks

Health checks are status monitors which are executed on a periodic basis by the health system to determine an individual health status. Health checks are designed to be simple; repeatedly evaluating the same data. The health system determines an overall Resin health status by aggregating the results of all the configured health checks.

18.2.1 Health status

Every time a health check executes it produces a HealthStatus and a message. The following is a list of all health statuses and their generally implied meaning.

Table 18.1: HealthStatus

NAME	ORDINAL VALUE	DESCRIPTION
UNKNOWN	0	Health check has not yet executed or failed to execute properly; status is inconclusive.
OK	1	Health check reported healthy status. This does not imply recovery.
WARNING	2	Health check reported warning threshold reached or critical is possible.
CRITICAL	3	Health check reported critical status; action should be taken.
FATAL	4	Health check reported fatal; restart expected.

The descriptions above should be understood to be entirely dependent on health action and predicate configuration. For example, a FATAL status does not imply a restart will occur unless `health:Restart` is configured with the `health:IfHealthFatal` predicate, as it is in the default health.xml.

18.2.2 System checks

System checks are health checks that can only exist once per JVM due to the nature of the data they sample. Most system checks are pre-configured in the default health.xml.

Note: System checks are singletons. Configuring duplicate system checks with different names will not result in the creation of duplicate system checks. The following is technically valid configuration, but results in configuring the same system check twice.

Example: duplicate system checks

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:CpuHealthCheck ee:Named="cpuCheck1">
    <warning-threshold>95</warning-threshold>
  </health:CpuHealthCheck>

  <health:CpuHealthCheck ee:Named="cpuCheck2">
    <warning-threshold>99</warning-threshold>
  </health:CpuHealthCheck>

</cluster>
```

In this example, `warning-threshold` will be set to 95 and then overrided to 99.

18.2.3 User checks

User checks are not pre-defined in health.xml; an administrator must configure them in health.xml as appropriate for an application. User checks are not singletons; the same check type can be configured in health.xml more than once provided they have different names.

Example: duplicate user checks

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <!-- Http status check 1 for database with email to database admin -->

  <health:HttpStatusHealthCheck ee:Named="databaseCheck">
    <url>http://localhost:8080/databaseCheck.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:SendMail>
    <to>database_team@yourdomain.com</to>
    <health:IfHealthCritical healthCheck="${databaseCheck}"/>
    <health:IfRechecked/>
  </health:SendMail>

  <!-- Http status check 2 for application with email to application admin -->

  <health:HttpStatusHealthCheck" ee:Named="appCheck">
```

```
    <url>http://localhost:8080/applicationTest.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:SendMail>
    <to>app_team@yourdomain.com</to>
    <health:IfHealthCritical healthCheck="${appCheck}"/>
    <health:IfRechecked/>
  </health:SendMail>

</cluster>
```

18.3 Health actions

Health actions perform a task, usually in response to specific conditions, or as remediation for a health check status. Like health checks, health actions are configured in health.xml and executed by the health system on a periodic basis. Health actions are usually accompanied by one or more conditions, or predicates , but this is not required. All actions have the potential to be executed once per period, determined by evaluation of associated conditions. A health action with no conditions will execute once per period.

18.4 Health conditions

Health condition, or predicates , qualify an action to execute based on a set of criteria. The action/condition pattern is intentionally similar to Resin's rewrite dispatch/condition pattern, so it should be familiar to some users. Health actions are evaluated every period. Conditions prevent the execution of an action unless all condition evaluate to true. A health action with no conditions will execute once per period. When more than one condition is present for an action, the default combining condion is #health:And .

18.4.1 Basic conditions

Basic conditions evaluate some general criteria and return true if the condition matches. Basic conditions do not evaluate the status of a health check. Instead they evaluate some general criteria like the time of day.

18.4.2 Combining conditions

General condition or health check conditions can be combined or negated using these conditions.

18.4.3 Health check conditions

All health check conditions evaluate some aspect of the results of a health check. All optionally accept the parameter health-check , which can reference a specific named health check. In absence of this parameter, overall aggregated Resin health will be used.

18.4.4 Lifecycle conditions

Lifecycle conditions evaluate the current state of Resin, qualifying actions to execute only during a Resin lifecycle state change.

18.5 Configuration

18.5.1 health.xml

Meters are configured as part of health.xml using ../admin/config-candi.xtp to create and update Java objects. Refer to ../admin/health-checking.xtp#Configuration for a full description of health.xml. Resin 4.0.17 and later includes a full compliemnt of pre-configured JMX meters in health.xml.

Example: importing health.xml into resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">
  <cluster-default>
    ...
    <!--
       - Admin services
      -->
    <resin:DeployService/>

    <resin:if test="${resin.professional}">
      <resin:AdminServices/>
    </resin:if>

    <!--
       - Configuration for the health monitoring system
      -->
    <resin:if test="${resin.professional}">
      <resin:import path="${__DIR__}/health.xml" optional="true"/>
    </resin:if>
    ...
  </cluster-default>
</resin>
```

Note: `<resin:AdminServices/>` (or more precisely just `<resin:StatsService/>`) is required to support health meters and graphing.

18.5.2 Meter names

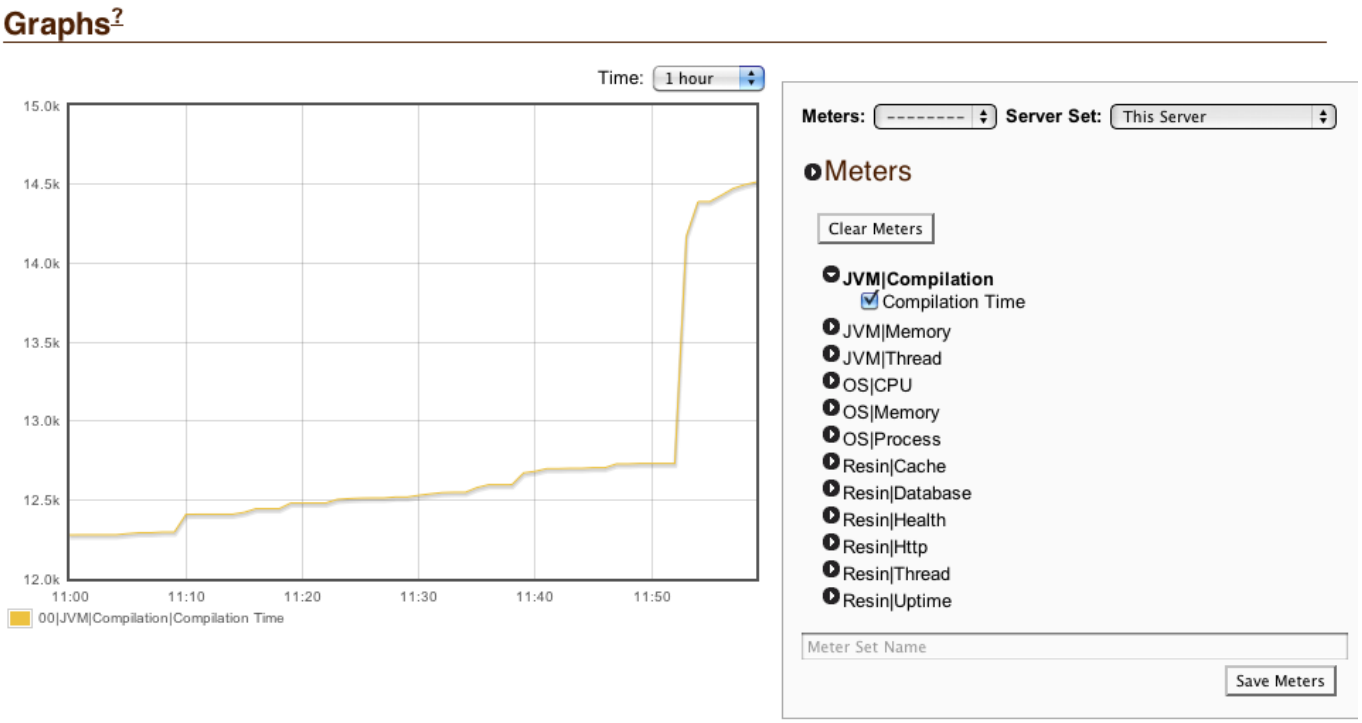
Health meters are named using a concatenation of keys separated by pipe (|) characters, loosely organized from least specific to most specific. Since meter statistics are shared between each member in a Resin cluster, Resin will automatically prefix each meter name with the cluster node index to insure the name is unique between cluster members.

The pipe character in the name provides a secondary benefit of helping to enhance the /resin-admin UI by categorizing meters into drill downs. Consider the following example.

Example: meter naming

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JmxDeltaMeter>
    <name>JVM|Compilation|Compilation Time</name>
    <object-name>java.lang:type=Compilation</object-name>
    <attribute>TotalCompilationTime</attribute>
  </health:JmxDeltaMeter>

</cluster>
```

In this example, `JVM|Compilation|Compilation Time` provides the base of the name. For cluster node index 0, Resin prefixes the name with `00|` . /resin-admin will then use the cluster index and first two keys to create drill downs to logically organized meters for display.



18.6 JMX meters

Virtually any local numeric JMX MBean attribute can be graphed using JMX meters.

18.7 Statistical Analysis

18.7.1 Detecting Anomalies

Meters alone are useful for manual inspection in resin-admin since every meter can be graphed. However Resin provides an extremely useful automatic analysis tool called AnomalyAnalyzer. AnomalyAnalyzer looks at the current meter value, checking for deviations from the average value. So unusual changes like a spike in blocked threads can be detected.

18.7.2 Reacting to Anomalies

The <health-event> attribute of AnomalyAnalyzer allows us to tie health actions to a detected anomaly by using the <health:IfHealthEven condition.

Chapter 19

Health: Report

19.1 Getting a Health Report

test

19.2 Getting a Health Report

The quickest method to get a health report is to use the command-line "pdf-report". The pdf-report will ask the server to generate a report immediately.

Example: generating a PDF report

```
unix> bin/httpd.sh pdf-report

generated /var/resin/log/default-Summary-20110921T1218.pdf
```

19.2.1 resin.xml automatic pdf report generation

PDFs can also be configured in the resin.xml to be generated weekly, or on events like a restart.

Example: PDF weekly summary generation

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster id="">

  <resin:import path="${__DIR__}/health.xml"/>

  <health:PdfReport>
     <path>${resin.root}/doc/admin/pdf-gen.php</path>
     <report>Summary</report>
     <snapshot/>
     <mailto>user@example.com</mailto>
     <profile-time>60s</profile-time>

     <health:IfCron value="0 0 * * 0"/>
   </health:PdfReport>

   ...
</cluster>
</resin>
```

The previous example generates weekly report by creating a snapshot (heap, threads, jmx, and profile), generating the PDF, and mailing the report to user@example.com .

The next example generates a PDF on a restart by the watchdog system.

Example: PDF watchdogreport

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster id="">

  <resin:import path="${__DIR__}/health.xml"/>

  <health:PdfReport>
     <path>${resin.root}/doc/admin/pdf-gen.php</path>
     <title>Restart</title>
     <watchdog/>
     <mailto>user@example.com</mailto>

     <health:OnRestart/>
   </health:PdfReport>

   ...
</cluster>
</resin>
```

19.3 Report Overview

Summary : The key JVM, OS and Resin versions. Meter Graphs : Graphs of statistics over the reported time. Heap Dump : The top heap memory usage. CPU Profile : A CPU profile of the system, showing the most active locations. Thread Dump : All the threads in Resin at the time the snapshot was taken. Log Report : The most recent "warning" level messages from java.util.logging. JMX Dump : Data from all the JMX mbeans at the time the snapshot was taken.

The reports are designed be used in several situations: bug report/support : for Resin's own support, it's much easier to debug a problem when we have a full snapshot to work from. restart/crach analysis : if the JVM crashes for some reason, the watchdog system report can help track down the cause. CPU and performance : if the JVM shows unexpected CPU use or sluggish performance, the report can help show the main causes. thread spikes and locking : the thread dump and CPU can be used to track down locked code.

19.4 Heap Dump

The heap dump gives a quick overview of the memory allocation of the system. It's generally useful as a check for any unusual allocations.

The heap dump is sorted by "self+desc", which is the objects own size plus its descendant size. A java.lang.String for example would include the char[] as part of its "self+desc".

The following is an example of a basic idle Resin heap dump.

Example: Heap Dump

```
 Class Name                                 self+desc  self  count

 byte[]                                      22.23M  22.23M   14741
 com.caucho.db.block.Block                   20.28M  134.9K    2410
 char[]                                      13.89M  13.89M  122606
 com.caucho.util.LruCache                     7.52M   30.4K     317
 java.lang.String                             7.15M   1.97M   61426
 ...
```

The first items, the byte[] and Block are primarily Resin's internal proxy cache and distributed cache database. Notice that the "self" for the Block is much smaller than its. "self+desc", because each Block has a large byte[] buffer.

Similarly, the String "self+desc" is much large than its "self" because it includes the char[] buffer.

19.4.1 ClassLoader Heap Dump

The Heap Dump section has a separate ClassLoader heap dump section which just displays the ClassLoader usage. You can use this report to check for class-based memory leaks.

19.5 Thread Dump

For CPU problems and stuck threads, the Thread Dump will show what each thread is doing in the system. You can use the report to see if many threads are piled up in an unusual location, like and unexpected lock, or track down a spinning thread.

The thread dump report merges threads which share the same stack trace. The merged threads will all be listed together, followed by their stack trace.

Blocked threads and the lock's owning thread are grouped together, so it's easier to see which thread is preventing many threads from continuing.

The following example shows a normal blocking situation. The JDK's SSL implementation only allows one thread to accept a connection at a time. All other threads will wait for the first thread. In this case, the thread named "http**://:8444-17" owns the SocksSocketImpl . Three threads are waiting in line for the lock: "http://**:8444-1" , "http**://:8444-1" , and "http://**:8444-1" .

Example: Thread Dump

```
http://*:8444-17
    java.net.PlainSocketImpl.socketAccept
    -- locked java.net.SocksSocketImpl@1199747469
    java.net.PlainSocketImpl.accept
    java.net.ServerSocket.implAccept
    ...
    com.caucho.env.thread.ResinThread.runTasks
    com.caucho.env.thread.ResinThread.run

http://*:8444-1
  waiting on java.net.SocksSocketImpl@4782b18d owned by [126] http://*:8444-17
http://*:8444-10
  waiting on java.net.SocksSocketImpl@4782b18d owned by [126] http://*:8444-17
http://*:8444-11
  waiting on java.net.SocksSocketImpl@4782b18d owned by [126] http://*:8444-17
    java.net.PlainSocketImpl.accept
    java.net.ServerSocket.implAccept
    com.sun.net.ssl.internal.ssl.SSLServerSocketImpl.accept
    ...
    com.caucho.env.thread.ResinThread.runTasks
    com.caucho.env.thread.ResinThread.run

...
```

19.6 CPU Profile

Because the CPU profile is calculated by repeated thread dumps, it's possible for a single stack trace to have more than 100% of the time when multiple threads are waiting at the same place.

You may need to skip the first set of waiting threads to see the profile traces you're interested in.

Example: CPU Profile

```
CPU Profile

Time:    60.1s
GC-Time: 0.303s
Ticks:   601
Sample-Period: 100
End: 2011-09-21 16:33

5000.00% 3005.00s  com.caucho.vfs.JniSocketImpl.nativeAccept()  RUNNABLE (JNI)
                   com.caucho.vfs.JniSocketImpl.nativeAccept()
                   com.caucho.vfs.JniSocketImpl.accept()
                   com.caucho.vfs.JniServerSocketImpl.accept()
                   com.caucho.network.listen.TcpSocketLinkListener.accept()
                   com.caucho.network.listen.TcpSocketLink.accept()
                   com.caucho.network.listen.TcpSocketLink.handleAcceptTask()

3564.23% 2142.10s  com.caucho.env.thread.ResinThread.waitForTask()   WAITING
                   sun.misc.Unsafe.park()
                   java.util.concurrent.locks.LockSupport.park()
                   com.caucho.env.thread.ResinThread.waitForTask()
                   com.caucho.env.thread.ResinThread.runTasks()
                   com.caucho.env.thread.ResinThread.run()

...
```

19.7 Logging

The most recent warning logs are reported as part of the heap dump.

Example: Warning Logs

```
Log(Warning)

2011-09-21 11:06:07 warning  WarningService: Resin restarting due to
                                    configuration change
```

19.8 JMX Dump

The JMX dump includes the full report of all the JMX MBeans in the system along with their values. Using this part of the report is somewhat specialized, either checking configured values against expectations or looking at statistics that aren't graphed as part of the metering system.

The JMX beans are sorted alphabetically.

Example: JMX Dump

```
JMX Dump

JMImplementation:type=MBeanServerDelegate
   ImplementationName     Resin-JMX
   ImplementationVendor   Caucho Technology
   ImplementationVersion  Resin-4.0.s110921
   MBeanServerId          Resin-JMX
   SpecificationName      Java Management Extensions
   SpecificationVendor    Sun Microsystems
   SpecificationVersion  1.4
```

```
com.sun.management:type=HotSpotDiagnostic
  ...
```

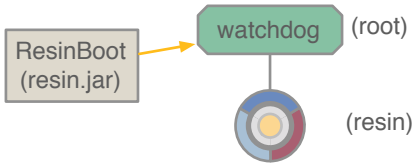
Chapter 20

Health: Watchdog

20.1 Overview

Because the watchdog runs quietly as a separate service, most of the time you won't need to pay attention to the watchdog at all. The standard configuration launches one watchdog per machine which monitors all the Resin JVMs on that matchine, so most sites will not need to change any watchdog configuration.

The main management tasks where you might need to pay attention to the watchdog is shutting it down if something is severely wrong with the machine, and checking the watchdog logs for Resin restart events.



The watchdog automatically restarts Resin if the Resin JVM ever crashes or exits. So if you want to stop Resin, you need to tell the watchdog to stop the instance, or you can stop the watchdog entirely. The watchdog is typically controlled by the resin.jar main program (`ResinBoot`), which has commands to start, stop, and restart Resin instances as well as reporting the watchdog status.

While most users will use the watchdog automatically with no extra configuration, ISPs and larger and complicated sites can create a specialised watchdog.xml with a `<watchdog-manager>` tag to control the watchdog at a much finer level. The `<watchdog-manager>` lets an ISP run the watchdog under its own control, and specify exactly the command-line parameters for their users' Resin instances, including the ability to create secure chroot instances for their users. Typically, the watchdog will run as root, while the user instances will run with their respective user ids.

20.2 command-line

The watchdog is controlled on the command-line using resin.jar's main class, `ResinBoot` . The major operations are: start, stop, restart, shutdown and status.

20.2.1 console

The "console" command starts a new Resin instance in a console window for development. The standard output of the Resin instance will appear in the console window.

Example: watchdog console

```
resin-4.0.x> bin/resin.sh -conf conf/test.conf -server a console
...
```

20.2.2 start

The "start" command starts a new Resin instance with the given server id.

`ResinBoot` will first try to contact the watchdog on the current machine, and start a new watchdog if necessary. The server id must be unique for all servers defined in the resin.xml.

Example: watchdog start

```
resin-4.0.x> bin/resin.sh -conf conf/test.conf -server a start

Resin/4.0.x started -server 'a' for watchdog at 127.0.0.1:6700
```

20.2.3 start-with-foreground

The "start-with-foreground" command starts a new Resin instance with the given server id. The command replicates "start" command and adds new behaviour: `ResinBoot` remains running after the command completes. This new command is designed to work with `launchd`

which is used to start daemon services on Mac OSX system. The server id must be unique for all servers defined in the resin.xml.

Example: watchdog start-with-foreground

```
resin-4.0.x> bin/resin.sh -conf conf/test.conf -server a start-with-foreground

Resin/4.0.x started -server 'a' for watchdog at 127.0.0.1:6700 with foreground
```

20.2.4 stop

The "stop" command stops the Resin instance with the given server id. If the stopped instances is the last one managed by the watchdog, the watchdog will automatically exit. If no `-server` is specified, the watchdog defaults to `-server ""` .

Example: watchdog stop

```
resin-4.0.x> bin/resin.sh stop

Resin/4.0.x started -server '' for watchdog at 127.0.0.1:6600
```

20.2.5 status

The "status" command summarizes the current Resin instances managed by the watchdog service.

Example: watchdog status

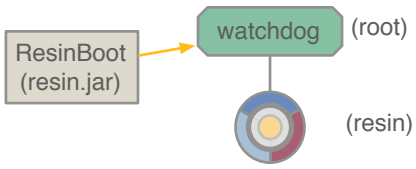
```
resin-4.0.x> bin/resin.sh status

Resin/4.0.x status for watchdog at 127.0.0.1:6600

server '' : active
  password: missing
  user: ferg
  root: /home/test/resin/
  conf: /etc/resin/resin.xml
```

20.3 Watchdog Health System

The watchdog system is responsible for monitoring the health of a Resin instance, restarting it if necessary, and reporting on the error conditions on any restart. start - the watchdog is responsible for keeping Resin running. security - on Unix systems the watchdog can bind to port 80 as root while Resin runs as a non-root user. report - the watchdog is responsible for collecting reports on the reasons for Resin restarts and providing that information to the health reporting system. restart - the watchdog works with the health system to allow graceful restarts if anomalies are detected in the Resin JVM.

20.4 Single Resin instance

This example shows a single-server site listening to the standard HTTP port 80 and running the server as the "resin" user. In this example, the watchdog typically runs as root so it can bind to the protected port 80, while the Resin instance runs as "resin" for security.



Since this configuration uses the default, the watchdog listens to port 6600 for commands.

Example: /etc/resin/resin.xml

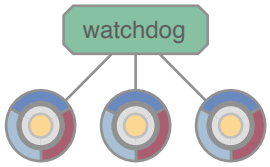
```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

<cluster id="">

  <server id="app-a" address="127.0.0.1">
    <user-name>resin</user-name>
    <group-name>resin</group-name>

    <http port="80"/>
  </server>

  <resin:import path="${__DIR__}/app-default.xml"/>

  <host id="">
    <web-app id="" path="/var/resin/htdocs"/>
  </host>

</cluster>
</resin>
```

20.5 Single machine load balance with shared watchdog

When running multiple instances of Resin on the same machine, one watchdog-manager typically handles all the instances. The server id will select which instance to start or stop.

In this example, there is one web-tier server acting as a load-balancer and two app-tier servers handling the backend data, all on a single machine. A site might want multiple app-tier servers for more reliable maintenance and upgrades. While one server is down, traffic can be handled by a second server.

The example uses default watchdog configuration from the standard resin.xml file. The watchdog process and `ResinBoot` will both read the resin.xml file for the server configuration, so there's no explicit watchdog configuration necessary. The watchdog detects that multiple servers are running on the same machine and manages all of them automatically.



Example: /etc/resin/resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

<cluster id="app-tier">

  <server-default>
    <user-name>resin</user-name>
    <group-name>resin</group-name>
  </server-default>

  <server id="app-a" address="192.168.1.10" port="6810"/>
  <server id="app-b" address="192.168.1.10" port="6811"/>

  <host id="">
    <web-app id="" path="/var/resin/htdocs"/>
  </host>

</cluster>

<cluster id="web-tier">

  <server-default>
    <user-name>resin</user-name>
    <group-name>resin</group-name>
  </server-default>

  <server id="web-a" address="192.168.1.10" port="6800">
    <http port="80"/>
  </server>

  <host id="">

     <resin:LoadBalance regexp="" cluster="app-tier"/>

  </host>

</cluster>

</resin>
```

20.6 Single machine load balance with distinct watchdog

In some cases, it's best to let each Resin instance have its own watchdog, for example when multiple users are sharing the same machine. Each <server> block configures a separate <watchdog-port>. Because the watchdog will read the resin.xml and use the <server> block matching the `-server id` command-line argument, each watchdog will start with it's own port.

Example: /etc/resin/resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="app-tier">

  <server-default>
    <user-name>resin</user-name>
    <group-name>resin</group-name>
  </server-default>

  <server id="app-a" address="192.168.1.10" port="6810">
    <watchdog-port>6700</watchdog-port>

    <http port="8080"/>
  </server>

  <server id="app-b" address="192.168.1.10" port="6811">
    <watchdog-port>6701</watchdog-port>

    <http port="8081"/>
  </server>

  <host id="">
    <web-app id="" path="/var/resin/htdocs"/>
  </host>

</cluster>

</resin>
```

In the previous example, starting Resin with `-server app-a`

will start a watchdog at port 6700. Starting Resin with `-server app-b` will start the watchdog at port 6701.

Example: starting app-b with watchdog-port=6701

```
resin-4.0.x> bin/resin.sh -server app-b start
```

20.7 ISP watchdog management

In a situation like an ISP, you may wish to have a separate configuration file for the watchdog, which launches Resin instances for different users. In this case, you will want to make sure the watchdog.xml is not readable by the users, and make sure to set a management user (see security.xtp). Start and restart the user's Resin JVM Set JVM parameters and Java executable Set the Resin instance root-directory setuid user-name and group-name Set the resin.xml configuration (must be readable by the user) Open protected ports like port 80 Optional chroot for additional security

The watchdog will launch the Resin instance with the given user as a setuid. It will also open any necessary protected ports, e.g. port 80.

Example: /etc/resin/watchdog.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">
```

```
<resin:AdminAuthenticator>
  <user name="harry" password="MD5HASH=="/>
</resin:AdminAuthenticator>

<watchdog-manager>

  <watchdog-default>
    <jvm-arg>-Xmx256m</jvm-arg>
  </watchdog-default>

  <watchdog id="user_1">
    <user-name>user_1</user-name>
    <group-name>group_1</group-name>

    <resin-xml>/home/user_1/conf/resin.xml</resin-conf>
    <resin-root>/home/user_1/www</resin-root>

    <open-port address="192.168.1.10" port="80"/>
  </watchdog>

  ...

  <watchdog id="user_n">
    <user-name>user_n</user-name>
    <group-name>group_n</group-name>

    <resin-conf>/home/user_n/conf/resin.xml</resin-conf>
    <resin-root>/home/user_n/www</resin-root>

    <open-port address="192.168.1.240" port="80"/>
  </watchdog>

</watchdog-manager>

</resin>
```

20.8 Management/JMX

The watchdog publishes the watchdog instances to JMX with the JMX name "resin:type=Watchdog,name=a". With a JMX monitoring tool like jconsole, you can view and manage the watchdog instances.

Chapter 21

Installation

21.1 null

If you have not yet done so, we suggest you use the http-server.xtp option first.

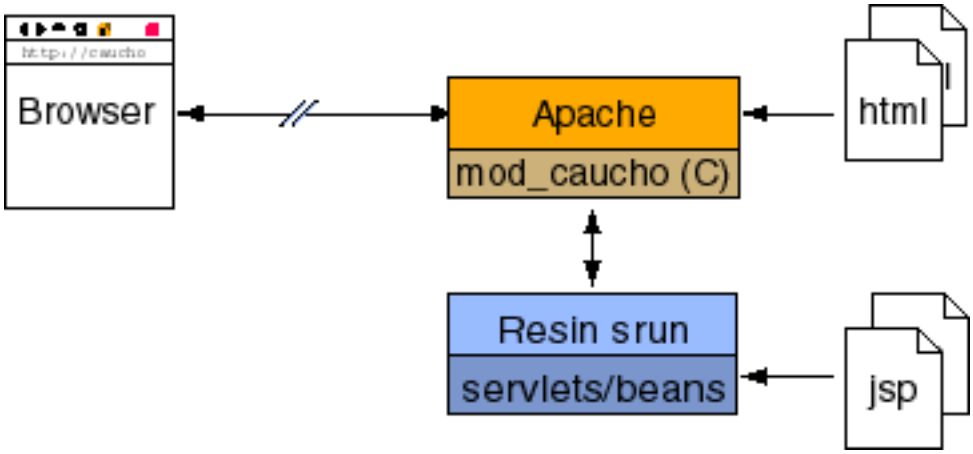
21.2 Before you integrate Resin with Apache

Before integrating Resin with Apache, it is valuable to configure Resin as a http-server.xtp , especially with more complicated setups such as those involving virtual hosts. Doing so isolates the steps and makes troubleshooting easier.

Many users find that the performance, flexibility, and features of Resin make Resin a desirable replacement for Apache.

21.3 How Resin integrates with Apache

When used with Apache, Resin serves JSPs and Servlets and Apache serves static content like html and images. Apache is a frontend server, it handles the request from the browser. Resin's `mod_caucho` plugin integrates with Apache, it dispatches requests for JSPs and Servlets to one or more backend Resin servers.



mod_caucho queries the backend server to distinguish the URLs going to Resin from the URLs handled by Apache. The backend server uses the <servlet-mapping> directives to decide which URLs to send. Also, any *.war file automatically gets all its URLs. Other URLs stay with Apache.

There's a more complete discussion of the URL dispatching in the starting-resin-apache-ref.xtp page.

21.4 Unix Installation

Resin needs Apache 1.3.x or greater and DSO support.

To configure Resin with Apache, you must follow the following steps: Compile Apache Compile mod_caucho.so Configure Apache Set up environment Configure resin.xml Restart Apache and start the backend Resin server

21.4.1 Compiling Apache

You need a version of Apache with DSO support enabled. Apache has full documentation at http://httpd.apache.org/docs/dso.html . To check if your apache has DSO support, you can check for mod_so.c in your your httpd.

checking apache httpd for mod_so.c

```
unix> /usr/local/apache/bin/httpd -l
Compiled-in modules:
  ...
  mod_so.c
  ...
```

Many distributions, e.g. Red Hat Linux, will have Apache preinstalled. However, because the standard distribution has files all over the place, some people prefer to recompile Apache from scratch.

Once you untar Apache, build it like:

```
unix> ./configure --prefix=/usr/local/apache --enable-module=so
unix> make
unix> make install
```

Solaris versions of Apache may need additional flags, otherwise you'll get some linking errors when trying to load Resin. You may need to refer to the Apache documentation if you get linking errors. Here's an example configuration on Solaris:

```
unix> ./configure --prefix=/usr/local/apache \
                  --enable-rule=SHARED_CORE \
                  --enable-rule=SHARED_CHAIN \
                  --enable-module=so \
                  --enable-module=most \
                  --enable-shared=max
```

21.4.2 Compiling mod_caucho.so

To compile and install mod_caucho on Unix, you'll need to run Resin's `configure` and then make. This step will create mod_caucho.so and put it in the Apache module directory. Usually, mod_caucho.so will end up in /usr/local/apache/libexec/-mod_caucho.so.

If you know where your apxs executable is, you can use --with-apxs. apxs is a little Perl script that the Apache configuration makes. It lets modules like Resin know how all the Apache directories are configured. It is generally in /usr/local/apache/bin/apxs or /usr/sbin/apxs. It's usually easiest to use --with-apxs so you don't need to worry where all the Apache directories are.

```
unix> ./configure --with-apxs=/usr/local/apache/bin/apxs
unix> make
```

Even if you don't know where apxs is, the configure script can often find it:

```
unix> ./configure --with-apxs
unix> make
```

As an alternative to --with-apxs, if you've compiled Apache yourself, or if you have a simple configuration, you can generally just point to the Apache directory:

```
unix> ./configure --with-apache=/usr/local/apache
unix> make
unix> make install
```

The previous --with-apxs or --with-apache should cover most configurations. For some unusual configurations, you can have finer control over each directory with the following arguments to ./configure. In general, you should use --with-apache or --with-apxs, but the other variables are there if you know what you're doing.

FLAG	DESCRIPTION
--with-apache=dir	The Apache root directory.
--with-apxs=apxs	Pointer to the Apache extension script
--with-apache-include=dir	The Apache include directory
--with-apache-libexec=dir	The Apache module directory
--with-apache-conf=httpd.conf	The Apache config file

21.4.3 Configure the Environment

If you don't already have Java installed, you'll need to download a JDK and set some environment variables.

Here's a typical environment that you might put in ~/.profile or /etc/profile

```
# Java Location
JAVA_HOME=/usr/java
export JAVA_HOME

# Resin location (optional).  Usually Resin can figure this out.
RESIN_HOME=/usr/local/share/resin
export RESIN_HOME

# If you're using additional class libraries, you'll need to put them
# in the classpath.
CLASSPATH=
```

21.5 Windows Installation

The `setup.exe` program installs the mod_caucho.dll plugin for any Apache it finds, and modifies the Apache httpd.conf file.

The httpd.conf file is also easily modified manually:

httpd.conf

```
LoadModule caucho_module \
    <installdir>/resin-pro-4.0.17/win32/apache-2.2/mod_caucho.dll

ResinConfigServer localhost 6800
<Location /caucho-status>
  SetHandler caucho-status
</Location>
```

21.6 Configuring resin.xml

The communication between `mod_caucho` and the backend Resin server takes place using a server port.

The `resin.xml` for the backend server contains a `server` to enable the port. The default `resin.xml`

has an server listener on port 6800.

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">

  ...


  <cluster id="app-tier">

    ...

    <server id="" address="127.0.0.1" port="6800"/>

    ...
```

The resin.xml and the layout of your webapps should match the layout that Apache expects. The mapping of urls to filesystem locations should be consistent between Apache and the backend Resin server.

The default resin.xml looks in

`resin-4.0.x/webapps/ROOT`

for JSP files and

`resin-4.0.x/webapps/ROOT/WEB-INF/classes`

for servlets and java source files. To tell Resin to use Apache's document area, you configure an explicit web-app with the appropriate document-directory:

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">

  ...

  <server>

    ...

    <host id="">

      <web-app id='/' document-directory="/usr/local/apache/htdocs"/>

    </host>

    ...

  </server>
</resin>
```

21.7 Starting the app-tier Resin server

Now you need to start the app-tier Resin server. Starting Resin is the same with Apache or standalone. See the http-server.xtp#deploy page for a detailed description.

```
unix> $RESIN_HOME/bin/resin.sh start

unix> bin/resin.sh start

win> resin.exe
```

```
May 6, 2011 3:06:05 PM com.caucho.boot.WatchdogChildTask run
INFO: WatchdogChild[] starting
May 6, 2011 3:06:05 PM com.caucho.boot.WatchdogChildProcess run
WARNING: Watchdog starting Resin[]
Resin Professional 4.0.17 (built Fri, 15 Apr 2011 06:35:56 PDT)
Copyright(c) 1998-2010 Caucho Technology.  All rights reserved.

  current.license -- 1 Resin server Caucho

Starting Resin Professional on Fri, 06 May 2011 15:06:06 -0400 (EDT)

[11-05-06 15:06:07.824] {main} Proxy Cache disk-size=1024M memory-size=64M
[11-05-06 15:06:08.179] {main}
[11-05-06 15:06:08.179] {main} Mac OS X 10.6.7 x86_64
[11-05-06 15:06:08.179] {main} Java(TM) SE Runtime Environment 1.6.0_24-b07-334-10M3326,  ↩
    MacRoman, en
[11-05-06 15:06:08.179] {main} Java HotSpot(TM) 64-Bit Server VM 19.1-b02-334, 64, mixed  ↩
    mode, Apple Inc.
[11-05-06 15:06:08.179] {main}
[11-05-06 15:06:08.179] {main} user.name  = caucho
[11-05-06 15:06:08.472] {main}
[11-05-06 15:06:08.479] {main} server listening to localhost:6800
[11-05-06 15:06:08.555] {main}
[11-05-06 15:06:08.873] {main}
[11-05-06 15:06:08.874] {main} resin.home = /Users/caucho/resin-pro-4.0.17/
[11-05-06 15:06:08.878] {main} resin.root = /Users/caucho/resin-pro-4.0.17/
[11-05-06 15:06:08.879] {main} resin.conf = /Users/caucho/resin-pro-4.0.17/conf/resin.xml
[11-05-06 15:06:08.889] {main}
[11-05-06 15:06:08.889] {main} server    = 127.0.0.1:6800 (app-tier:default)
[11-05-06 15:06:08.899] {main} stage      = production
[11-05-06 15:06:09.526] {main} WebApp[production/webapp/default/resin-admin] active
[11-05-06 15:06:10.245] {main} WebApp[production/webapp/default/resin-doc] active
[11-05-06 15:06:10.445] {main} WebApp[production/webapp/default/ROOT] active
[11-05-06 15:06:10.446] {main} Host[production/host/default] active
[11-05-06 15:06:10.447] {main} ProServer[id=default,cluster=app-tier] active
[11-05-06 15:06:10.448] {main}   JNI: file, nio keepalive (max=9984), socket
[11-05-06 15:06:10.448] {main}
[11-05-06 15:06:10.449] {main}
[11-05-06 15:06:10.450] {main} http listening to *:8080
[11-05-06 15:06:11.023] {main} https listening to *:8443
[11-05-06 15:06:11.092] {main}
[11-05-06 15:06:11.160] {main} ProResin[id=default] started in 4222ms
```

Resin will print every port it's listening to. In the above example, Resin has an http listener on port 8080 and an server listener on port 6800 (using its custom *hmux* protocol). mod_caucho establishes connections to Resin using port 6800, and a web browser can connect using port 8080. Usually the 8080 port will be unused, because web browsers will make requests to Apache, these requests get dispatched to Resin as needed by mod_caucho. A Resin configured http listener on port 8080 is a useful debugging tool, it allows you to bypass Apache and make a request straight to Resin.

The following snippet shows the < http-server-ref.xtp#http > and < server > configuration for the above example.

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <server-default>
      <http address="*" port="8080"/>
    </server-default>

    <server id="" address="192.168.2.10" port="6800"/>

    ...
  </cluster>
```

```
</resin>
```

21.8 Testing the servlet engine

Create a test file */usr/local/apache/htdocs/test.jsp*

```
2 + 2 = <%= 2 + 2 %>
```

Browse http://localhost/test.jsp again. You should now get

```
2 + 2 = 4
```

21.9 Configuring Apache httpd.conf

The installation process above automatically changes the httpd.conf file. You can also configure the httpd.conf file manually, or modify the default configuration created by the installation process.

Unix - httpd.conf

```
LoadModule caucho_module libexec/mod_caucho.so

ResinConfigServer localhost 6800
<Location /caucho-status>
  SetHandler caucho-status
</Location>
```

Windows - httpd.conf

```
LoadModule caucho_module \
    <installdir>/resin-pro-4.0.17/win32/apache-2.2/mod_caucho.dll

ResinConfigServer localhost 6800
<Location /caucho-status>
  SetHandler caucho-status
</Location>
```

The `ResinConfigServer` is used to tell mod_caucho how to contact the backend Resin server. The backend Resin server tell's mod_caucho which urls should be dispatched.

APACHE COMMAND	MEANING
ResinConfigServer host port	Specifies the Resin JVM at host:port as a configuration server.

21.9.1 caucho-status

`caucho-status` is optional and probably should be avoided in a production site. It lets you ask the Caucho Apache module about it's configuration, and the status of the backend server(s), valuable for debugging.

After any change to httpd.conf, restart Apache. Now browse http://localhost/caucho-status.

21.9.2 Manual configuration of dispatching

You can also dispatch to Resin directly from the httpd.conf. Instead of relying on the ResinConfigServer directive to determine which url's to dispatch to the backend server, Apache handler's are used to specify the url's to dispatch.

```
CauchoHost 127.0.0.1 6800

<Location /foo/*>
  SetHandler caucho-request
</Location>
```

APACHE COMMAND	MEANING
CauchoHost host port	Alternative to ResinConfigServer , adds the Resin JVM with an server port at host:port as a backend server.
CauchoBackup host port	Alternative to ResinConfigServer , adds the Resin JVM with a server port at host:port as a backup backend server.

APACHE HANDLER	MEANING
caucho-status	Handler to display /caucho-status
caucho-request	Dispatch a request to Resin

Requests dispatched directly from the Apache httpd.conf will not appear in /caucho-status.

21.10 Virtual Hosts

The http-virtual-hosts.xtp topic describes virtual hosts in detail. If you're using a single JVM, you only need to configure the resin.xml.

httpd.conf

```
LoadModule caucho_module libexec/mod_caucho.so

ResinConfigServer 192.168.0.1 6800
<Location /caucho-status>
  SetHandler caucho-status
</Location>
```

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="app-tier">

  <server id="" address="192.168.0.1" port="6800"/>

  <host id='www.gryffindor.com'>
    <host-alias>gryffindor.com</host-alias>
     ...
  </host>

  <host id='www.slytherin.com'>
    <host-alias>slytherin.com</host-alias>
     ...
  </host>
</cluster>
</resin>
```

21.10.1 Virtual Host per JVM

If you want a different JVM for each virtual host, your httpd.conf can specify a different server port for each host.

httpd.conf

```
<VirtualHost gryffindor.com>
ServerName gryffindor.com
ServerAlias www.gryffindor.com
ResinConfigServer 192.168.0.1 6800
</VirtualHost>

<VirtualHost slytherin.com>
ServerName slytherin.com
ServerAlias www.slytherin.com
ResinConfigServer 192.168.0.1 6801
</VirtualHost>
```

gryffindor.conf

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <server id="" address="192.168.0.1" port="6800"/>

  <host id="">
     ...
  </host>
</cluster>
</resin>
```

slytherin.conf

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster>

  <server id="" address="192.168.0.1" port="6801"/>

  <host id="">
     ...
  </host>
</cluster>
</resin>
```

```
$ bin/resin.sh -pid gryffindor.pid -conf conf/gryffindor.conf start
$ bin/resin.sh -pid slytherin.pid -conf conf/slytherin.conf start

...

$ bin/resin.sh -pid gryffindor.pid stop
```

21.11 Load Balancing

The clustering-overview.xtp section provides an introduction to the concepts of load balancing.

mod_caucho recognizes cluster configurations for load balancing. Requests are distributed to all machines in the cluster, all requests in a session will go to the same host, and if one host goes down, Resin will send the request to the next available machine. Optional backup machines only receive requests if all of the primaries are down.

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <server id="a" address="192.168.0.11" port="6800" index="1"/>
    <server id="b" address="192.168.0.11" port="6801" index="2"
```

```
              backup="true"/>
    <server id="c" address="192.168.0.12" port="6800" index="3"/>
    <server id="d" address="192.168.0.12" port="6801" index="4"
              backup="true"/>
    ...
  </cluster>
</resin>
```

mod_caucho only needs to know about one of the backend servers. It will query that backend server, and learn about all of the other members of the cluster.

```
ResinConfigServer 192.168.0.11 6800
```

mod_caucho keeps a local cache of the configuration information, so if the backend server becomes unavailable then the cached configuration will be used until the backend server becomes available again.

The httpd.conf file can also specify more than one backend server, when mod_caucho checks for configuration updates, it will check each in turn, and only if none of them are available will it use the local cached copy.

```
ResinConfigServer 192.168.0.11 6800
ResinConfigServer 192.168.0.12 6801
```

21.11.1 Manual configuration of load balanced dispatching

Manual dispatching in httpd.conf can also specify the backend hosts and the backend backup hosts, as an alternative to using `ResinConfigServer` .

```
CauchoHost 192.168.0.11 6800
CauchoBackup 192.168.0.11 6801
CauchoHost 192.168.0.12 6800
CauchoBackup 192.168.0.12 6801

<Location /foo/*>
  SetHandler caucho-request
</Location>
```

21.11.2 Manual configuration of location based dispatching

```
<Location /applicationA/*>
  ResinConfigServer 192.168.0.11 6800
</Location>

<Location /applicationB/*>
  ResinConfigServer 192.168.0.12 6800
</Location>
```

21.12 Troubleshooting

First, check your configuration with Resin standalone.sh. In other words, add a <http port=*8080*/> and check port 8080. Check http://localhost/caucho-status. That will tell if mod_caucho has properly connected to the backend Resin server. Each server should be green and the mappings should match your resin.xml. If caucho-status fails entirely, the problem is in the mod_caucho installation and the Apache httpd.conf. If caucho-status shows the wrong mappings, there's something wrong with the resin.xml or the pointer to the backend server in httpd.conf. If caucho-status shows a red servlet runner, then Resin hasn't properly started. If you get a "cannot connect to servlet engine", caucho-status will show red, and Resin hasn't started properly. If Resin doesn't start properly, you should look at the logs in resin-4.0.x/log. You should start resin.sh -verbose or resin.exe -verbose to get more

information. If Resin never shows a "hmux listening to *:6800" line, it's not listening for connections from mod_caucho. You'll need to add a <server> line. If you get Resin's "file not found", the Apache configuration is good but the resin.xml probably points to the wrong directories.

21.13 null

If you have not yet done so, we suggest you research the http-server.xtp option first.

Note: isapi_srun.dll should only be used with versions of IIS that do not provide integration with .NET.

21.14 Before you integrate Resin with IIS

Starting with version 4.0.7 Resin provides an IIS handler built using ASP.NET technologies. The handler is distributed with Resin in a dll named `Resin.IIS.Handler.dll` . The dll delivers an implementation of ASP.NET's

`System.Web.IHttpHandler`

that facilitates streaming content from Resin via IIS. This can be useful in a number of configurations where the main requirement is having IIS serving as a front-end to the clients (browsers).

In configurations that don't exhibit the aforementioned requriement a more powerful configuration of homogeneous Resin environment is recommended. Many users find that the performance, flexibility, and features of Resin make Resin a desirable replacement for IIS.

Before integrating Resin with IIS, it is valuable to configure Resin as a http-server.xtp , especially with more complicated setups such as those involving virtual hosts. Doing so isolates the steps and makes troubleshooting easier.

21.15 IIS Prerequisites

Resin IIS Handler relies on functionality provided by ASP.NET framework. The handler may also be used with earlier versions of IIS provided that ASP.NET framework is installed with the following Application Development Features enabled: .NET Extensibility ASP ASP.NET ISAPI Extensions ISAPI Filters

It's also recommended that IIS Management Console and IIS Management Scripts and Tools features are enabled. IIS Scripts come with a command line IIS configuration utility `appcmd.exe` . Once the feature is installed,

`appcmd.exe`

utility will be placed into %SystemRoot%\system32\inetsrv directory. Note: the directory won't be added to %PATH% automatically.

21.16 How Resin Integrates with IIS

In the deployment scenarios made available by the IIS Handler, Resin can serve application completely or partially. Capability to serve application partially may be especially useful in scenarios where applicaton uses a mix of .NET and Java technologies. Such situations may arise when porting an application between platforms.

In all deployment schemes Resin IIS Handler needs to be added to the

`handlers`

section in ASP.NET web application configuration file located in the root of the context, where the context may be a Site Context or Web Application Context.

The Diagram below depicts a deployment with Resin IIS Handler configured to handle all requests to

`*.jsp`

pages, while all static content is served by IIS.

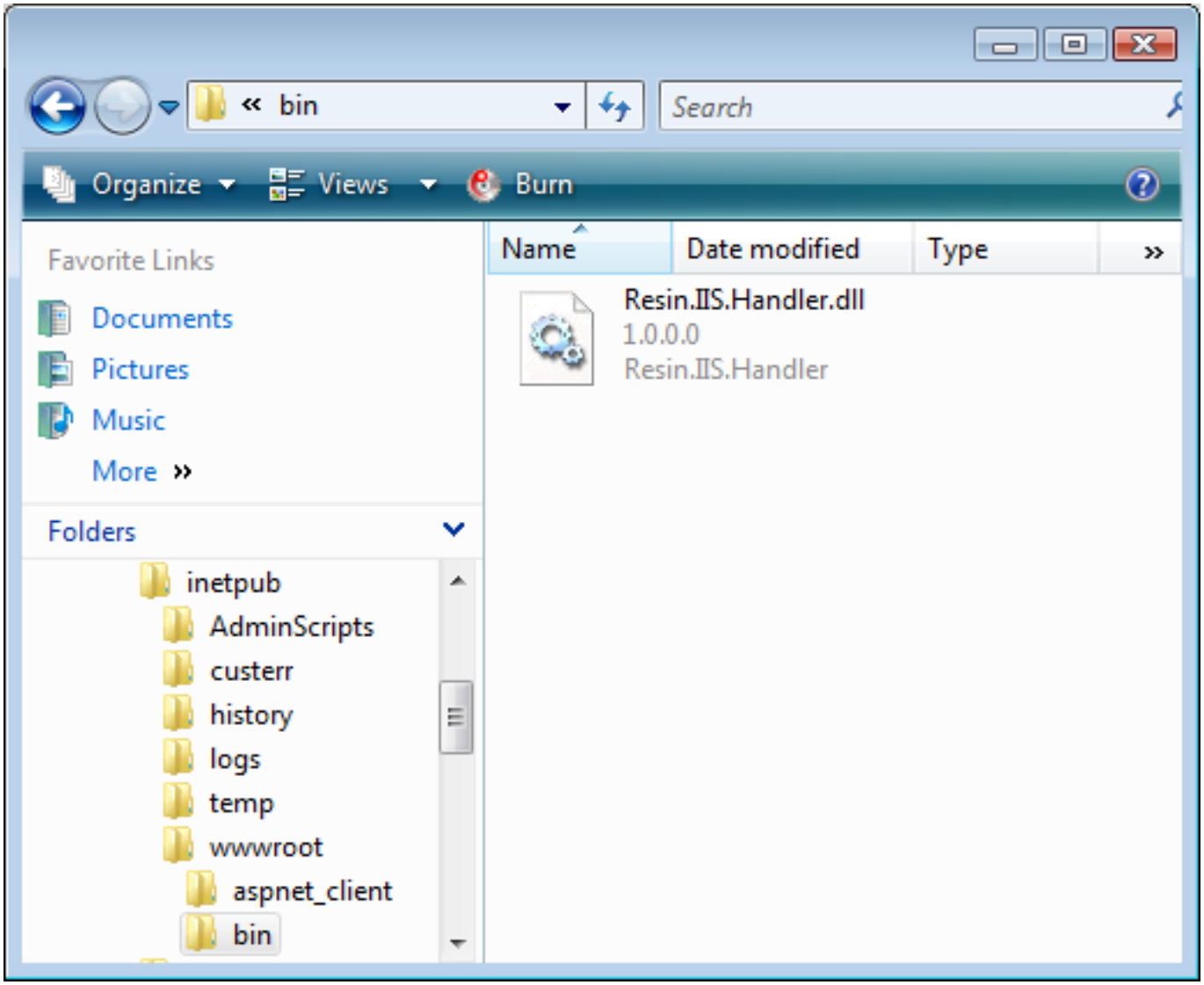
alt="IIS_srun.gif"

Next section explains how to install and configure Resin IIS Handler to handle variuos scenarios.

21.17 Installing and Configuring Resin IIS Handler

Installing Resin IIS Pugin into IIS requires following the rules defined for ASP.NET extensions, which prescribe location of the binaries and configuration steps including: Copying Resin.IIS.Handler.dll to web application's bin directory Registering the handler in ASP.NET web application config file

The bin directory is located at the root of the context which can be a web site context or a web application context. In both cases bin directory needs to be created if one does not exist already. E.g. after deploying the dll to handle requests at the default web site level the web site context contents tree should looks like the following:



If the permissions don't allow the dll file to be copied into the needed location higher trust level needs to be requested from system's administrator.

Once the dll is copied into the directory it is ready for registering it with the context. Registration can be done manually, by adding a handler to web application context configuration file. The web application configuration file in ASP.NET is named web.config and is placed in the root of the context. E.g. for default site the configuration file will be located in c:\inetpub\wwwroot directory. If the file isn't there it can be created manually or, if appcmd utility is used, automatically.

Successful registration of Resin IIS Handler manually or using appcmd should produce a file that will look as the following simplified but complete configuration file.

web.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="ResinHandlerFactory"  <!--Name used by resin handler. Can be changed.-->
           path="*"  <!-- Requests path to invoke the handler on-->
           verb="*"  <!-- HTTP Method. possible values: 'GET', 'POST', 'HEAD' or  ↩
               combination 'GET,POST' -->
           type="Caucho.IIS.ResinHandlerFactory"  <!-- Resin Handler Factory that supplies  ↩
               one reusable handler instance-->
           preCondition="integratedMode"/>
    </handlers>
  </system.webServer>
  <appSettings>
    <add key="resin.servers" value="127.0.0.1:6800"/>  <!-- Address and port to Resin  ↩
        server(s)-->
    <add key="resin.log-level" value="Information"/> <!-- logging level used by Resin IIS  ↩
        Handler. Use 'None' for silent operation-->
  </appSettings>
</configuration>
```

Configuration given above register Resin IIS Handler to handle requests coming at any url by specifying a wild card for

`path`

attribute: path="*"

In order to have Resin IIS Handler serve only jsp files the path attribute needs to be given a value of "*.jsp". The values for attributes follow the rules and conventions defined for IIS and are limited in their matching capabilities.

Resin IIS Handler communicates with Resin using HMUX (Resin's internal protocol) with the connection made to the port defined in Resin's configuration file. Matching Resin configuration for the example IIS configuration given above must bind Resin HMUX listener to port 6800.

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
       ...
  <cluster id="app-tier">
       ...
    <server id="" address="127.0.0.1" port="6800"/>
       ...
```

Port 6800 is the default HMUX port used by Resin.

Resin IIS Handler accepts the following configuration parameters:

NAME	DESCRIPTION	DEFAULT
resin.servers	Space separated list of ip:port pairs to backend Resin servers e.g. *127.0.0.1:6800 127.0.0.1:6801*	127.0.0.1:6800
resin.log-level	Logging level: Information	Warning
Error	None	Error
resin.session-cookie	Session Cookie Name	JSESSIONID
resin.ssl-session-cookie	Secure Session Cookie Name	SSLJSESSIONID
resin.sticky-sessions	Use sticky sessions for distributing requests	true

NAME	DESCRIPTION	DEFAULT
resin.session-url-prefix	prefix used to identifiy sessionid in urls	;jsessionid=
resin.load-balance-connect-timeout	Timeout used with connect to backend	5 sec.
resin.load-balance-idle-time	Maximum time to keep sockets to backend open	5 sec.
resin.load-balance-recover-time	Retry period on failed backend server	15 sec.
resin.socket-timeout	How long to wait for a read or write operation to backend server to complete	65 sec.
resin.caucho-status	Enables or disables /caucho-status request.	true

In order to configure Resin IIS Handler to communicate to a set of Resin Pro Servers resin.servers configuration parameter needs be an enumeration of ip:port pairs:

web.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="ResinHandlerFactory"  <!--Name used by resin handler. Can be changed.-->
           path="*"  <!-- Requests path to invoke the handler on-->
           verb="*"  <!-- HTTP Method. possible values: 'GET', 'POST', 'HEAD' or  ↩
               combination 'GET,POST' -->
           type="Caucho.IIS.ResinHandlerFactory"  <!-- Resin Handler Factory that supplies  ↩
               one reusable handler instance-->
           preCondition="integratedMode"/>
    </handlers>
  </system.webServer>
  <appSettings>
    <add key="resin.servers" value="127.0.0.1:6800 127.0.0.2:6800 127.0.0.1:6801  ↩
        127.0.0.2:6802"/>  <!-- Address and port to Resin server(s)-->
    <add key="resin.log-level" value="Information"/> <!-- logging level used by Resin IIS  ↩
        Handler. Use 'None' for silent operation-->
  </appSettings>
</configuration>
```

21.18 Configuring using appcmd.exe configuration utility

Appcmd.exe offers a command line interface to configuring IIS server. Below are the examples showing how to use the utility to configure various aspects of Resin IIS Handler.

Assuming that we are working with the Default Web Site:

Registering the Resin IIS Handler

```
        # Remove Resin IIS Hanlder if one is registered
        %APPCMD% set config "Default Web Site" /section:handlers /-[name=' ↩
            ResinHandlerFactory']

        # Add Resin IIS Handler
        %APPCMD% set config "Default Web Site" /section:handlers /+[name=' ↩
            ResinHandlerFactory',path='*',\
        verb='*',type='Caucho.IIS.ResinHandlerFactory',preCondition='integratedMode']

        # Note: The above command needs to be issued in one line. The \ character
        # at the last position of the first line is used to denote that command
```

```
        # continues on next line


        # Note: %APPCMD% should be pointing to the location of appcmd.exe utility.
        #       default location is c:\Windows\System32\inetsrv\appcmd.exe
```

Specifying parameters using appcmd.exe:

Resin IIS Hanlder parameters

```
        # Remove resin.servers parameter
        %APPCMD% set config "Default Web Site" /section:appSettings /-[key='resin.servers']
        # Set resin.servers parameter
        %APPCMD% set config "Default Web Site" /section:appSettings /+[key='resin.servers', ↩
            value='192.168.0.1:6800']

        # Remove resin.log-level parameter
        %APPCMD% set config "Default Web Site" /section:appSettings /-[key='resin.log-level ↩
            ']
        # Set resin.log-level parameter to 'Information'
        %APPCMD% set config "Default Web Site" /section:appSettings /+[key='resin.log-level ↩
            ',value='Information']
```

21.19 Servicing a mixed technologies application

Using Resin IIS Handler it's possible to deploy an application that employes both ASP.NET and JSP technologies. In such a configuration requests for Java EE based resources needs to be forwarded to Resin while the rest can be served by IIS.

E.g. Assuming an applicaiton consists of index.jsp and an index.aspx page in application residing at path c:\temp\a. To be able to serve such an application from both Resin and IIS it needs to be deployed at both.

Deploying application /a to Resin

```
        ...
        <cluster id="app-tier">
          ...
          <server id="" port="6800"/>
          ...
          <web-app id="/a" root-directory="c:/temp/a"/>
        </cluster>
```

Deploying application */a* to IIS can be done with IIS Manager, by executing *Add Applicaton* action from the *Site's* context menu. Once the *Add Application* dialog opens field Alias needs to be supplied value of *a*, field Physical path – value of *c:\temp\a*

Once this configuration is complete copy the Resin.IIS.Handler.dll into the *c:\temp\a\bin* directory and create web.conf file with settings configured to match **.jsp* requests and forward them to an instance of Resin.

web.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<system.webServer>
  <handlers>
    <add name="ResinHandlerFactory" path="*.jsp" verb="*" type="Caucho.IIS. ↩
        ResinHandlerFactory" preCondition="integratedMode"/>
  </handlers>
</system.webServer>
<appSettings>
  <add key="resin.servers" value="127.0.0.1:6800"/>
  <add key="resin.log-level" value="Information"/>
</appSettings>
</configuration>
```

21.20 Tracing & Logging with Resin IIS Handler

21.20.1 Tracing

When Resin IIS Handler is compiled with TRACE option enabled it uses

`System.Diagnostics.Trace` class to output detailed information for every request. This can be useful in debugging errors. Tracing information will go into a file configured in a tracing listener.

Configuring Resin IIS Handler for tracing

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.diagnostics>
    <trace autoflush="true" indentsize="0">
      <listeners>
         <add name="resin-trace" initializeData="C:\temp\resin-iis-trace.log" type="System. ↩
            Diagnostics.TextWriterTraceListener" />
      </listeners>
    </trace>
  </system.diagnostics>
</configuration>
```

Note, that Resin IIS Handler in distribution compiled with no TRACE option.

21.20.2 Logging

Resin IIS Handler uses standart .NET loggin mechanism to output information on its opeartion. Upon startup, the handler attempts to initialize Logging to write to *Application* log using *Resin IIS Handler* log source. If the log source can not be created automatically due to permissions or other issues, the source can be created manually be adding a key *Resin IIS Handler* to log sources collection in the registry at

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\Resin IIS Handle
```

Alternatively, copy the contents of the following text box into a file with .reg extension and double click on it to import into registry.

Resin IIS Handler Logging Source

```
Windows Registry Editor Version 5.00

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\Resin IIS Handler
```

21.21 Starting the app-tier Resin server

Now you need to start the app-tier Resin server. Starting Resin is the same with IIS or standalone. See the http-server.xtp#deploy page for a detailed description.

```
        win> resin.exe console
```

```
        Resin 4.0.7 (built Mon Aug 4 09:26:44 PDT 2006)
        Copyright(c) 1998-2006 Caucho Technology. All rights reserved.

        Starting Resin on Mon, 04 Aug 2006 09:43:39 -0700 (PDT)
        [09:43:40.664] Loaded Socket JNI library.
        [09:43:40.664] http listening to *:8080
        [09:43:40.664] ServletServer[] starting
```

```
        [09:43:40.879] hmux listening to *:6800
        [09:43:41.073] Host[] starting
        [09:43:41.446] Application[http://localhost:8080/resin-doc] starting
        [09:43:41.496] Application[http://localhost:8080] starting
```

Resin will print every port it's listening to. In the above example, Resin has an http listener on port 8080 and an server listener on port 6800 (using its custom *HMUX* protocol). Resin IIS Handler establishes connections to Resin using port 6800, and a web browser can connect using port 8080. Usually the 8080 port will be unused, because web browsers will make requests to IIS, these requests get dispatched to Resin as needed by Resin IIS Handler. A Resin configured http listener on port 8080 is a useful debugging tool, it allows you to bypass IIS and make a request straight to Resin.

21.22 Resin IIS Handler Status

`caucho-status`

is optional and probably should be avoided in a production site. It lets you ask the Caucho IIS module about it's configuration, and the status of the backend server(s), valuable for debugging. To see caucho-status point your browser url to http://iisserver/-caucho-status.

Note, that in a configuration where handler mapping prevents handler from being invoked on /caucho-status request a 404 will be returned. In such configuration /caucho-status may still be available if there is an extension mapping present for the handler e.g. ***.jsp*** *or .xtp* – simply add the extension to status request http://iisserver/caucho-status.jsp or http://iisserver/cauchos-status.xtp

21.23 unix layout

Sample Debian layout

```
/var/resin/             # Resin root-directory
  app-inf/              # custom app-tier cluster configuration and jars
  doc/                  # Resin documentation
  endorsed/             # Special overriding jar directory (uncommon)
  project-jars/         # Resin extensions to third-party projects
  resin-data/           # Resin working directory
  resin-inf/            # custom Resin-wide configuration and jars
  watchdog-data/        # Watchdog working directory
  webapp-jars/          # jars included for every web-app
  webapps/              # Application deployment

/var/log/resin/         # Resin runtime logs
  watchdog-manager.log  # log for the watchdog
  jvm-app-0.log         # log for the server named "app-0"

/etc/resin/             # Resin configuration
  resin.properties      # configuration properties
  resin.xml             # main configuration file
  app-default.xml       # documentation for web-app default
  cluster-default.xml   # common configuration for all clusters
  health.xml            # health system rules, meters, and actions
  keys/                 # openssl keys
  licenses/             # Resin licenses

/etc/init.d/resin       # Unix init startup service

/usr/bin/resinctl       # Resin command-line script

/usr/share/resin/       # resin-home
  bin/                  # Resin startup scripts
  lib/                  # Resin jars
```

```
  libexec64/           # Resin jni binaries
```

21.24 resin-data

The resin-data directory contains Resin's internal data, including deployed webapps, the health system data, and distributed caches.

If Resin's internal data becomes corrupted, you can remove the resin-data directory to reset the state. If you have multiple servers in a triad hub, the reset server will restore its values from the other servers.

When Resin is deployed in a Unix environment, the resin-data will typically be in /var/resin/resin-data.

example resin-data

```
resin-data/
  app-0/          # each named server gets its own section
    .git/         # web-app deployment .git repository
    distcache/    # distributed cache directory
      data.db     # distcache value data
      mnode.db    # distcache key/value binding
    log/          # health system log
      log_data.db # log entries
      log_name.db # log names
    stat_data.db  # health statistics data
    stat_name.db  # health statistics names
    tmp/          # temp swap directory
      temp_file
    xa.log.a      # XA log
```

Chapter 22

Logging

22.1 java.util.logging

22.1.1 Overview

Resin uses the JDK standard java.util.logging for all its internal logging and offers flexible configuration for the logging format and the logging level. The logging configuration has two parts: the set of log handlers, and the logger level.

A log handler tells Resin where to send logging output. Resin includes file-based log handlers, mail handlers, syslog handlers, and any custom logging handler following the JDK standard. The log handler is configured with a name and a level which must both match for the log to be output.

Example: file-based log-handler in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

  <log-handler name="com.foo" level="all"
               path="${resin.root}/log/foo.log"
               timestamp="[%y-%m-%d %H:%M:%S.%s] {%{thread}} "/>
  ...

</resin>
```

The <logger> configures the logging level for a named logger. Because a <logger> will generally have several log-handlers, both the logger level and the log-handler level must match for the log to output. Since the logger and log-handler names are hierarchical, a "com.foo" <logger> will enable "com.foo.bar".

Example: logging at the fine level in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

  <logger name="com.foo" level="fine"/>
  <logger name="com.foo.bar" level="finest"/>
  ...

</resin>
```

22.1.2 Log names

The JDK logging api uses a hierarchical naming scheme. Typically the name is aligned with a java class name. When you specify a name, all logging requests that use a name that starts with the name you have specified are matched. For example: `<logger name="example.hogwarts" ...>`

matches a logging request for both "example.hogwarts.System" and "example.hogwarts.gryffindor.System"

Resin's own logging is based on Resin's source class names. The following are useful log names in Resin:

Table 22.1: Resin log names

NAME	MEANING
""	Debug everything
com.caucho.ejb	EJB handling
com.caucho.jsp	Debug jsp
com.caucho.java	Java compilation
com.caucho.server.port	TCP port debugging and threading
com.caucho.server.http	HTTP-related debugging
com.caucho.server.webapp	web-app related debugging
com.caucho.server.cache	Cache related debugging
com.caucho.sql	Database pooling
com.caucho.transaction	Transaction handling

22.1.3 Log levels

The logger level enables logs for a given debugging granularity. A "severe" level only shows logs which threaten the stability of the server, and a "fine" level shows debugging information intended for application/library users.

The logger levels match the values defined by the JDK java.util.logging.Level .

Table 22.2: Logging Level values

NAME	API	SUGGESTED USE
off		turn off logging
severe	log.severe("...")	a major failure which prevents normal program execution, e.g. a web-app failing to start or a server restart
warning	log.warning("...")	a serious issue, likely causing incorrect behavior, like a 500 response code to a browser
info	log.info("...")	major lifecycle events, like a web-app starting
config	log.config("...")	detailed configuration logging
fine	log.fine("...")	debugging at a user level, i.e. for someone not familiar with the source code being debugged
finer	log.finer("...")	detailed debugging for a developer of the code being debugged
finest	log.finest("...")	events not normally debugged, e.g. expected exceptions logged to avoid completely swallowing, or Hessian or XML protocol parsing
all		all messages should be logged

22.1.4 <log-handler>

Configure a log handler for the JDK java.util.logging.* API. java.util.logging has two steps: configure a set of log handlers, and configure the levels for each logger. The <log-handler> creates a destination for logs, sets a minimum logging level for the handler, and attaches the handler to a logging name.

In addition to configuring custom handlers, <log-handler> has the most common configuration build-in: logging to a rotating file. Most of the configuration attributes are used for the rotating file and are shared with the other logging configuration.

22.1.5 log-handler timestamp

The timestamp for log tags is a format string which can contain percent codes which are substituted with time and date values.

CODE	MEANING
%a	day of week (short)
%A	day of week (verbose)
%b	day of month (short)
%B	day of month (verbose)
%c	Java locale date
%d	day of month (two-digit)
%H	24-hour (two-digit)
%I	12-hour (two-digit)
%j	day of year (three-digit)
%m	month (two-digit)
%M	minutes
%p	am/pm
%S	seconds
%s	milliseconds
%W	week in year (three-digit)
%w	day of week (one-digit)
%y	year (two-digit)
%Y	year (four-digit)
%Z	time zone (name)
%z	time zone (+/-0800)
	Current thread name
	Current logging level
	Current class-loader environment

Example: typical timestamp for the log tag

```
<resin xmlns="http://caucho.com/ns/resin">

  <log-handler name='' path='stderr:' timestamp="[%H:%M:%S.%s] {%{thread}}"/>

  ...
</resin>
```

```
[22:50:11.648] WebApp[/doc] starting
[22:50:11.698] http listening to *:8080
[22:50:11.828] hmux listening to *:6800
```

22.1.6 log-handler archiving

The following example is a standard log handler writing to a rollover file. Because the handler's level is "all", the <logger> configuration will set the actual logging level.

Example: logging to a rollover file

```
<web-app xmlns="http://caucho.com/ns/resin">

  <log-handler name="" level="all"
       timestamp="[%Y/%m/%d %H:%M:%S.%s] {%{thread}} "/>

  <logger name="com.caucho" level="info"/>

</web-app>
```

The default archive format is

```
path  + ".%Y%m%d"    if rollover-period >= 1 day.
path  + ".%Y%m%d.%H" if rollover-period < 1 day.
```

For example, to log everything to standard error use:

Example: logging everything to System.err

```
<resin xmlns="http://caucho.com/ns/resin">
  <log-handler name='' level='all' path='stderr:' timestamp="[%H:%M:%S.%s]"/>
  ...
</resin>
```

A useful technique is to enable full debug logging to track down a problem:

debug logging

```
<resin>
  ...
  <log-handler name='' level='finer' path='log/debug.log'
       timestamp="[%H:%M:%S.%s]"
       rollover-period='1h' rollover-count='1'/>
  ...
</resin>
```

22.1.7 log-handler EL formatting

The format for Resin's log-handler tags specifies a format string for each log message. `format` recognizes EL-expressions. The EL variable log is a com.caucho.log.ELFormatter.ELFormatterLogRecord object.

Example: log format string

```
<log-handler name='' level='all' path='stderr:' timestamp="[%H:%M:%S.%s]"
     format=" ${log.level} ${log.name} ${log.message}"/>
```

Table 22.3: log EL variable *log* is a LogRecord

ACCESSOR	VALUE
${log.level}	The level of the log record
${log.name}	The source loggers name
${log.shortName}	A shorter version of the source loggers name, "Foo" instead of "com.hogwarts.Foo"
${log.message}	The message, with no formatting or localization
${log.millis}	event time in milliseconds since 1970
${log.sourceClassName}	Get the name of the class that issued the logging request (may not be available at runtime)
${log.sourceMethodName}	Get the name of the method that issued the logging request (may not be available at runtime)

Table 22.3: (continued)

ACCESSOR	VALUE
${log.threadID}	Get an int identifier of the thread where the logging request originated
${log.thrown}	Get any java.lang.Throwable associated with the logging request
	The name of the current thread.
	The servlet request value.
	The servlet session.
${cookie[*JSESSIONID*]}	The value of a request cookie.

You can also use the config-el.xtp in your format string:

log format string using an Environment EL variable.

```
<host ...>

  <web-app>
    <log name='' level='all' path='log/debug.log' timestamp="[%H:%M:%S.%s]"
         format=" [ ${app.contextPath} ] ${log.message}"/>

    ...
  </web-app>

  ...

</host>
```

```
[14:55:10.189] [/foo] 'null' returning JNDI java:
       model for EnvironmentClassLoader[web-app:http://localhost:8080/foo]
[14:55:10.189] [/foo] JNDI lookup 'java:comp/env/caucho/auth'
       exception javax.naming.NameNotFoundException: java:comp/env/caucho/auth
[14:55:10.199] [/foo] Application[http://localhost:8080/foo] starting
```

The config-el.xtp#fmt.sprintf function can space pad the values and make the results look a little nicer:

fmt.sprintf() in log format string

```
<log name='' level='all' path='stderr:' timestamp="[%H:%M:%S.%s]"
     format=" ${fmt.sprintf('%-7s %45s %s',log.level,log.loggerName,log.message)}">
```

```
[14:28:08.137] INFO com.caucho.vfs.QJniServerSocket Loaded Socket JNI library.
[14:28:08.137] INFO com.caucho.server.port.Port http listening to *:8080
[14:28:08.137] INFO com.caucho.server.resin.ServletServer ServletServer[] starting
[14:28:08.307] INFO com.caucho.server.port.Port hmux listening to localhost:6802
[14:28:08.437] INFO com.caucho.server.host.Host Host[] starting
```

config-el.xtp#fmt.sprintf and config-el.xtp#fmt.timestamp can be used to produce CSV files:

CSV log files

```
<log name='' level='all' path='log/debug.csv' timestamp=""
     format="${fmt.sprintf('%vs,%d,%d,%vs,%vs',fmt.timestamp('%Y-%m-%d %H:%M:%S.%s'),
               log.threadID, log.level.intLevel(), log.loggerName, log.message)}"/>
```

```
"2003-11-17 14:46:14.529",10,800,"com.caucho.vfs.QJniServerSocket",
            "Loaded Socket JNI library."
"2003-11-17 14:46:14.549",10,800,"com.caucho.server.port.Port",
            "http listening to *:8080"
"2003-11-17 14:46:14.549",10,800,"com.caucho.server.resin.ServletServer",
            "ServletServer[] starting"
"2003-11-17 14:46:14.719",10,800,"com.caucho.server.port.Port",
            "hmux listening to localhost:6802"
"2003-11-17 14:46:14.850",10,800,"com.caucho.server.host.Host",
            "Host[] starting"
"2003-11-17 14:46:15.100",10,800,"com.caucho.server.webapp.Application",
            "Application[http://localhost:8080/freelistbm] starting"
```

22.1.8 Logger: Application logging

You can take advantage of the JDK's logging facility to add logging to your application. Choosing a good logging name and levels are important for troubleshooting and debugging your code. Logging to much can be almost as confusing as logging too little.

The logging name should be the full class name of the class you're instrumenting. Although other schemes are possible, the class name is more maintainable.

The logging level should be consistent across your application. For Resin, we use the following level conventions:

Example: logging at finer

```
import java.util.logging.Logger;
import java.util.logging.Level;

public class Foo {
  private static final Logger log
    = Logger.getLogger(Foo.class.getName());

  ...
  void doFoo(String bar)
  {
    // check for log level if your logging call does anything more
    // than pass parameters
    if (log.isLoggable(Level.FINER))
        log.finer(this + "doFoo(" + bar + ")");

    ...

    log.info(...);

    try {
        ...
    } catch (ExpectedException ex) {
      log.log(Level.FINEST, "expected exception", ex);
    }
  }
  ...
}
```

22.1.9 Custom and library log handlers

Custom handlers and log handlers from libraries can be configured with Resin's logging system, using the CanDI XML configuration syntax. The custom handler is a child of <log-handler> and configured with any argument or setters necessary. Resin will install the handler just like one of its own handlers.

Example: JDK's FileHandler

```
<web-app xmlns="http://caucho.com/ns/resin"
   xmlns:jdk-logging="urn:java.util.logging">

  <log-handler name="com.foo" level="info">
    <jdk-logging:FileHandler>
      <new>
        <value>/tmp/test.out</value>
      </new>
    </jdk-logging:FileHandler>
  </logger>

</web-app>
```

Example: MyHandler.java

```
package com.foo.demo;

import java.util.logging.*;

public class MyHandler extends Handler
{
  @Override
  public void publish(LogRecord record)
  {
    System.out.println(getFormatter().format(record));
  }

  @Override
  public void flush();
  {
  }

  @Override
  public void close();
  {
  }
}
```

22.1.10 Custom log formatting

The formatting of a log message can be customized just like the log handler. The Formatter is a java.util.logging interface which Resin's logging understands and can be configured with <log-handler>.

Sites may wish to change the formatting of log messages to gather information more appropriate for the site. The formatter can be custom-configured just like the handlers.

Example: custom formatter configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:mypkg="urn:java:com.mycom.mypkg">

  <log-handler name="com.foo" level="warning" path="WEB-INF/log.log">
    <formatter><mypkg:MyFormatter/></formatter>
  </log-handler>

</web-app>
```

Example: MyFormatter.java

```
package com.mycom.mypkg;

import java.util.logging.*;

public class MyFormatter extends Formatter
{
  @Override
  public String format(LogRecord record)
  {
    return "[" + record.getLevel() + "] " + record.getMessage();
  }
}
```

22.1.11 Resin Builtin Log Handlers

Resin provides a number of predefined custom log handlers for common logging patterns, including sending messages to JMS, HMTP, and the syslog service. Creating your own custom handler is also straightforward.

22.1.11.1 BamLogHandler (4.0.5)

The BAM handler publishes the log message to a BAM agent. The agent can be a custom HMTP service to process log messages. The `BamHandler` needs a JID (Jabber id) as the address of the target service.

Example: BAM handler configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.resin">

  <logger name="com.foo">
    <resin:BamLogHandler level="warning">
       <to>test@localhost</to>
    </resin:BamLogHandler>
  </logger>

</web-app>
```

22.1.11.2 EventLogHandler

The event handler publishes a `LogEvent` to the CanDI event system. Any CanDI component with an `@Observes` method for

`LogEvent` will receive the notifications. The log handler classname is `com.caucho.log.EventLogHandler` .

Example: event handler configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
   xmlns:resin="urn:java:com.caucho.resin">

  <logger name="com.foo">
    <resin:EventLogHandler level="warning"/>
  </logger>

</web-app>
```

22.1.11.3 JmsLogHandler

The JMS handler publishes the log message to a JMS queue.

Example: JMS handler configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
    xmlns:ee="urn:java:ee"
    xmlns:resin="urn:java:com.caucho.resin">

  <resin:MemoryQueue ee:Named="myQueue"/>

  <logger name="com.foo">
    <resin:JmsLogHandler level="warning">
       <target>${myQueue}</target>
   </resin:JmsLogHandler>
  </logger>

</web-app>
```

22.1.11.4 MailLogHandler (4.0.5)

The Mail handler sends log messages to an email address. To keep the number of mails down, the handler will concatenate messages and only send them after a period of time.

Table 22.4: MailLogHandler attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
to	mail address	required
delay-time	time to wait before sending first mail	1m
mail-interval-min	minimum time between mail messages	1h
properties	javamail properties in property file format	

Example: Mail handler configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.resin">

  <logger name="">
    <resin:MailLogHandler level="warning">
       <to>admin@foo.com</to>
       <properties>
         mail.smtp.host=127.0.0.1
         mail.smtp.port=25
       </properties>
    </resin:MailLogHandler>
  </logger>

</web-app>
```

22.1.11.5 SyslogLogHandler

On Unix systems, the SyslogLogHandler lets you log messages to syslog.

Example: syslog configuration

```
<resin xmlns="http://caucho.com/ns/resin"
      xmlns:resin="urn:java:com.caucho.resin">

<logger name="">
  <resin:SyslogLogHandler level="warning">
    <facility>daemon</facility>
    <severity>notice</severity>
  </resin:SyslogLogHandler>
</logger>

</resin>
```

The possible values for facility are user, mail, daemon, auth, lpr, news, uucp, cron, authpriv, ftp, local0, local1, local2, local3, local4, local5, local6, local7. The default is daemon .

The possible values for severity are emerg, alert, crit, err, warning, notice, info, debug. The default is info .

See also `man 3 syslog ' and `man syslog.conf '.

22.2 Log rotation and archiving

Log rotation are a way stop your log files from getting too large, and to archive log files on a weekly or daily basis. When a rollover is triggered, the existing log file is renamed and a new file is started. Logs can be rotated by size or by time.

The same log rotation mechanism in Resin is used for JDK logging, HTTP access logging, and standard output logging.

22.2.1 Size based rollover

A size based rollover is triggered when the size of the file reaches a certain amount. The default Resin behaviour for log's is to rollover when the file size reaches 1mb.

`rollover-size` is used to specify the maximum size, and can be in bytes (50000), kilobytes (128kb), or megabytes (10mb). A value of `-1`

disables size based rollovers.

22.2.2 Time based rollover

A time based rollover is triggered when a certain period of time has passed since the last rollover. The default Resin behaviour is to perform no time based rollover, unless rollover-size has been disabled with a value of -1 in which case the default time period is 1 month.

`rollover-period` is used to specify the time period, and can be in days (15D), weeks (2W), months (1M), or hours (1h).

22.2.3 Archive files

When a rollover is triggered, the log file is renamed (archived) and a new log file is started.

`archive-format` is used to specify the name of the archive file. It can contain regular characters, config-el.xtp , and % codes that capture the current date and time. The % codes are the same as the ones used for `timestamp`

(see #timestamp .

The default behaviour depends on the value of rollover-period. If rollover-period is greater than one day, or is not being used because rollover-size has been specified, the archive filename is the original path with `.%Y%m%d` appended. If rollover-period is less than one day, the archive filename is the original path with `.%Y%m%d.%H` appended.

22.2.4 Disabling rollovers

To completely disable rollovers, set the `rollover-size` to such a high number that it will never occur:

disable log rollovers

```
  <stdout-log path="log//stdout.log" rollover-size="1024mb"/>
```

22.2.5 Compression

Rollover log files can be compressed with gzip or zip. The extension of the archive-format determines the compression.

```
<log name="" level="warning" path='log/error.log'
     archive-format="%Y-%m-%d.error.log.gz"
     rollover-period="1D"/>

<access-log path="log/access.log"
            archive-format="access-%Y%m%d.log.gz"
            rollover-period="1D"/>
```

22.3 Standard Output Redirection

22.3.1 stdout-log

Configure the destination for System.out .

Usage of the stdout-log overrides a previous usage. For example, specifying stdout-log as a child of a web-app causes a redirection of System.out for that web application only, and will override the System.out location in the enclosing host .

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-format	the format for the archive filename when a rollover occurs, see #rollover .	see below
path	Output path for the stream, see #path .	required
path-format	Selects a format for generating path names. The syntax is the same as for archive-format	optional
rollover-count	maximum number of rollover files before the oldest ones get overwritten. See #rollover .	none
rollover-period	how often to rollover the log. Specify in days (15D), weeks (2W), months (1M), or hours (1h). See #rollover .	none
rollover-size	maximum size of the file before a rollover occurs, in bytes (50000), kb (128kb), or megabytes (10mb). See #rollover .	1mb
timestamp	a timestamp #timestamp to use at the beginning of each log line.	no timestamp

The default archive format is path + ".%Y%m%d" or path + ".%Y%m%d.%H" if rollover-period < 1 day.

The following example configures System.out for a host . Unless a web-app overrides with it's own `stdout-log` , all web-apps in the host will write to the same output file.

```
...
<host id='foo.com'>
```

```
  <stdout-log path='/var/log/foo/stdout.log'
              rollover-period='1W'/>
  ...
</host>
...
```

22.3.2 stderr-log

Configure the destination for System.err .

Usage of the stderr-log overrides a previous usage. For example, specifying stderr-log as a child of a web-app causes a redirection of System.err for that web application only, and will override the System.err location in the enclosing host .

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Output path for the stream, see #path .	required
path-format	Selects a format for generating path names. The syntax is the same as for archive-format	optional
timestamp	a timestamp #timestamp to use at the beginning of each log line.	no timestamp
rollover-count	maximum number of rollover files before the oldest ones get overwritten. See #rollover .	none
rollover-period	how often to rollover the log. Specify in days (15D), weeks (2W), months (1M), or hours (1h). See #rollover .	none
rollover-size	maximum size of the file before a rollover occurs, in bytes (50000), kb (128kb), or megabytes (10mb). See #rollover .	1mb
archive-format	the format for the archive filename when a rollover occurs, see #rollover .	see below

The default archive format is path + ".%Y%m%d" or path + ".%Y%m%d.%H" if rollover-period < 1 day.

The following example configures System.err for a host . Unless a web-app overrides with it's own `stderr-log` , all web-apps in the host will write to the same output file.

```
...
<host id='foo.com'>
  <stderr-log path='/var/log/foo/stderr.log'
              rollover-period='1W'/>
  ...
</host>
...
```

22.4 Log Paths

path is used to configure a destination for the messages. Typically,

`access-log` , `stdout-log` , and `stderr-log` are configured to go to files, and `log` is configured to go to a file or to stderr or stdout so that they show up on the console screen.

PATH	RESULT
filesystem path	output log entries to a file

PATH	RESULT
stdout:	output log entries to stdout
stderr:	output log entries to stderr

Log messages to stdout

```
<log name="" level="all" path="stdout:"/>
```

You can use the config-el.xtp as part of your filesystem path:

Filesystem path using Environment EL variables

```
<log name="" level="all"
     path="log/debug-${server.id}.log"
     rollover-period="1h" rollover-count="1"/>
```

Chapter 23

Scheduled Tasks

23.1 <resin:ScheduledTask>

<resin:ScheduledTask> schedules a job to be executed at specific times or after specific delays. The times can be specified by a cron syntax or by a simple delay parameter. The job can be either a `Runnable`

bean, a method specified by an EL expression, or a URL.

When specified as an Java Injection bean, the bean task has full IoC capabilities, including injection, @TransactionAttribute aspects, interception and @Observes.

Table 23.1: <resin:ScheduledTask> Attributes

ATTRIBUTE	DESCRIPTION
cron	a cron-style scheduling description
delay	a simple delay-based execution
mbean-name	optional MBean name for JMX registration
method	EL expression for a method to be invoked as the task
name	optional IoC name for registering the task
period	how often the task should be invoked in simple mode
task	alternate task assignment for predefined beans

23.1.1 Java Injection bean job configuration

The most common and flexible job configuration uses standard IoC bean-style configuration. The bean must implement `Runnable` . The <task> element specifies the bean, using standard Java injection syntax as described in candi.xtp configuration.

Example: 5min cron bean task

```
<web-app xmlns="http://caucho.com/ns/resin"
      xmlns:resin="urn:java:com.caucho.resin">

  <resin:ScheduledTask>
    <cron>*/5</cron>

    <qa:MyTask xmlns:qa="urn:java:com.caucho.resin"/>
  </ScheduledTask>

</web-app>
```

23.1.2 task reference job configuration

The task bean can also be passed to the <scheduled-task> using a Resin-IoC EL reference. The name of the task bean would be defined previously, either in a <bean> or <component> or picked up by classpath scanning. Like the bean-style job configuration, the reference bean must implement `Runnable` .

Example: midnight cron bean task

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:ScheduledTask task="#{taskBean}">
    <cron>0 0 *</cron>
  </resin:ScheduledTask>

</web-app>
```

23.1.3 method reference job configuration

<scheduled-task> can execute a method on a defined bean as the scheduler's task. The method is specified using EL reference syntax. At each trigger time, <scheduled-task> will invoke the EL method expression.

In the following example, the task invokes `myMethod()`

on the myBean singleton every 1 hour.

Example: 1h period method task

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.resin"
  xmlns:qa="urn:java:qa">

  <qa:MyBean>
    <Named>myBean</Named>
  </qa:MyBean>

  <resin:ScheduledTask method="#{myBean.myMethod}">
    <resin:delay>10m</resin:delay>
    <resin:period>1h</resin:period>
  </resin:ScheduledTask>

</web-app>
```

23.1.4 url job configuration

In a <web-app>, the <scheduled-task> can invoke a servlet URL at the trigger times. The task uses the servlet `RequestDispatcher`

and forwards to the specified URL. The URL is relative to the <web-app> which contains the <scheduled-task.

Example: sunday cron url task

```
<web-app xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.config">

  <resin:ScheduledTask url="/cron.php">
    <resin:cron>0 15 * * 0</resin:cron>
  </resin:ScheduledTask>

</web-app>
```

23.1.5 cron trigger syntax

Some ascii art from the http://en.wikipedia.org/wiki/Crontab

```
# +---------------- minute (0 - 59)
# |  +------------- hour (0 - 23)
# |  |  +---------- day of month (1 - 31)
# |  |  |  +------- month (1 - 12)
# |  |  |  |  +---- day of week (0 - 6) (Sunday=0 or 7)
# |  |  |  |  |
  *  *  *  *  *
```

Table 23.2: cron patterns

PATTERN	DESCRIPTION
*	matches all time periods
15	matches the specific time, e.g. 15 for minutes
15,45	matches a list of times, e.g. every :15 and :45
*/5	matches every n times, e.g. every 5 minutes
1-5	matches a range of times, e.g. mon, tue, wed, thu, fri (1-5)

Each field specifies a range of times to be executed. The patterns allowed are:

Table 23.3: example ranges

RANGE	EXPLANATION (USING MINUTES AS EXAMPLE)
*	run every minute
*/5	run every 5 minutes
0,5,50	run at :00, :05, :50 every hour
0-4	run at :00, :01, :02, :03, :04
0-30/2	run every 2 minutes for the first half hour

The minutes field is always required, and the hours, days, and months fields are optional.

Table 23.4: example times

RANGE	EXPLANATION
0 */3	run every 3 hours
15 2 *	run every day at 0215 local time
0 0 */3	run every third day at midnight
15 0 * * 6	run every Saturday at 0015

Chapter 24

Security

24.1 Overview

Resin includes a comprehensive security framework for application authentication, authorization and transport level SSL based security. Authentication capabilities include built-in support for security data stored in XML files, the database, JAAS, LDAP or properties files, HTTP basic authentication, form based authentication and password digests. The authorization features include traditional role based security as well as robust conditionals based-on cookies, HTTP headers, locale, IP address and the like. The security framework also supports single sign-on shared across multiple web applications.

This document covers Resin's authentication and authorization capabilities while Resin's SSL support is described in detail security-ssl.xtp .

The basic structure of the Resin security framework can be demonstrated through a simple example:

WEB-INF/resin-web.xml Basic Security Framework Example

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  <!-- The authentication method -->
  <resin:BasicLogin/>

  <!-- Authorization -->
  <resin:Allow url-pattern="/foo/*">
    <resin:IfUserInRole role="user"/>
  </resin:Allow>

  <!-- Authentication provider -->
  <resin:XmlAuthenticator password-digest="none">
    <resin:user name="Harry Potter" password="quidditch" group="user,gryffindor"/>
    <resin:user name="Draco Malfoy" password="pureblood" group="user,slytherin"/>
  </resin:XmlAuthenticator>

</web-app>
```

Most usages of the Resin security framework will follow the general outline of the example above. In the example, the <resin:Allow> tag enables authorization whereas the <resin:IfUserInRole> applies role-based authorization. Users actually enter login/authentication information through HTTP basic authentication in the example above. The authentication information that the user enters is checked against an authentication provider. In the example, the authentication information, including user name, password and groups, are stored in XML (the passwords above are simple text, but they need not and should not be). Note, authentication can sometimes be entirely unnecessary, especially while using conditional authorization rules not specific to a user - such as a condition allowing or denying a set of IP addresses from accessing a URL.

24.2 Authenticators

Authentication is the process of verifying that a user is who they say they are. The most common way of verifying the identity of a user is through user name and password. As demonstrated in the example, Resin uses authenticators to verify user credentials. Authenticators look for authentication data matching a login in a back-end resource such as a database or LDAP directory.

The following are the authenticators Resin currently supports:

Table 24.1: Authenticators

NAME	DESCRIPTION
http://caucho.com/resin-javadoc/com/caucho/security/-DatabaseAuthenticator.html	This authenticator works with authentication information stored in a relational database and uses JDBC.
http://caucho.com/resin-javadoc/com/caucho/security/-JaasAuthenticator.html	This authenticator can be used to plugin-in any Java authentication service (JAAS) module into Resin, including the JAAS modules built into the Sun JDK.
http://caucho.com/resin-javadoc/com/caucho/security/-LdapAuthenticator.html	This can be used with authentication data stored in LDAP and uses JNDI under the hood.
http://caucho.com/resin-javadoc/com/caucho/security/-PropertiesAuthenticator.html	This authenticator can use credentials stored in properties files.
http://caucho.com/resin-javadoc/com/caucho/security/-XmlAuthenticator.html	Uses data stored in XML (either in-line or in a separate file).
http://caucho.com/resin-javadoc/com/caucho/security/-AbstractAuthenticator.html	This is an abstract class you can use to create your own Resin custom authenticator.

Each authenticator is described in detail security-authenticators.xtp , including example code that you could use as a starting point for your application. The built-in authenticators should satisfy a large number of common cases, but you can easily create your own custom authenticator when needed.

Resin supports single sign-on in the form of authenticators at the server or virtual host level shared across multiple web applications. This described in detail with authenticators if this is functionality you need for your application.

24.3 Securing Resin Administration

The Resin security framework and the authenticators above are intended for application security. However, Resin resources such as `/resin-admin` as well as the Resin clustered managemnt and deployment administrative features also needs to be secured. Resin internally uses the security framework to secure these resources.

The http://caucho.com/resin-javadoc/com/caucho/security/AdminAuthenticator.html tag is used to secure all Resin resources such as /resin-admin. The admin authenticator is defined only once in the resin.xml file. The authenticator uses the exact same syntax as the XmlAuthenticator.

Resin's top-level <resin:AdminAuthenticator> tag is essentially a static, XML-based authentication context. The authenticator is automatically shared for all hosts and web-apps, so simple sites can even use this authenticator configuration for their site-wide authentication.

Here is a basic example of the Resin admin authenticator:

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:AdminAuthenticator>
    <user name="admin" password="MD5HASH=="/>
```

```
    ...
  </resin:AdminAuthenticator>
  ...
</resin>
```

24.4 Securing Passwords with Digests

Passwords in clear-text form are a major security vulnerability. Such passwords can be stolen during transmission or storage and used malicicously (such as in order to gain unauthorized access to back-end resources). The transmission vulnerability is caused by the fact that passwords are sent across the network to the server from the browser in plain text when HTTP basic authentication or form-based authentication is used. This vulnerability can be addressed by either using HTTP digest authentication (covered security-authentication-method.xtp) or by using transport layer SSL security (covered security-ssl.xtp).

You can secure passwords in storage by using the password digest feature built-into Resin authenticators (see the `password-digest` attribute). You can use the password-digest attribute to specify that the authenticator should use passwords in a secure fashion. When this feature is enabled, the authenticator will store the password in digest instead of clear-text form. When the authenticator receives a clear-text password, it will digest it before comparing it to a stored password for a match.

A digest of a clear-text password is calculated when it is passed through a one-way hashing function that consistently produces another series of characters,

`digestPassword = digester(username + ":" + realm + ":" + cleartextPassword)` . The function is "one-way" because the digestPassword cannot be used to practically reverse-engineer the original password.

Resin's authenticators use "MD5-base64" and a realm "resin" to digest passwords by default. MD5 indicates that the MD5 hashing algorithm is used. base64 is an encoding format to apply to the binary result of MD5. You can create an MD5/Base64 digest yourself with a simple PHP script like this:

Calculating a Digest Using PHP

```
<?php

  $username = "harry";
  $password = "quidditch";
  $realm = "resin";

  echo base64_encode(md5("$username:$realm:$password", true));

?>
```

The following are some examples of passwords digested by Resin:

USERNAME	REALM	PASSWORD	DIGEST
root	resin	changeme	j/qGVP4C0T7UixSpKJpTdw==
harry	resin	quidditch	uTOZTGaB6pooMDvqvl2Lbg==
hpotter	resin	quidditch	x8i6aM+zOwDqqKPRO/vkxg==
filch	resin	mrsnorris	KmZIq2RKXAHV4BaoNHfupQ==
pince	resin	quietplease	Txpd1jQc/xwhISIqodEjfw==
snape	resin	potion	I7HdZr7CTM6hZLlSd2o+CA==
mcgonagall	resin	quidditch	4slsTREVeTo0sv5hGkZWag==
dmalfoy	resin	pureblood	yI2uN1l97Rv5E6mdRnDFwQ==
lmalfoy	resin	myself	sj/yhtU1h4LZPw7/Uy9IVA==

In the above examples the digest of "harry/quidditch" is different than the digest of "hpotter/quidditch" because even though the password is the same, the username has changed. The Resin digest is calculated with

`digest(username + ":" + realm + ":" + password)` , so if the username changes the resulting digest is different.

24.4.1 Calculating a Digest

While using password digests with Resin authenticators, it may occationally be necessary to calculate digests yourself. You can do this in a number of different ways. You could use the PHP script example above. The `/resin-admin` page includes a form to easily generate the MD5 hash. You can also use the http://caucho.com/resin-javadoc/com/caucho/server/security/-PasswordDigest.html class to generate the digest programmatically. The following is an example of using this class:

Calculating a Digest - Java example

```
  import com.caucho.security.PasswordDigest;
  ...
  String username = ...;
  String password = ...;
  String realm = "resin";

  PasswordDigest passwordDigest = PasswordDigest();

  String digest = passwordDigest.getPasswordDigest(username, password, realm);
```

Unix users can quickly calculate a digest with this script:

```
echo -n "user:resin:password" | openssl dgst -md5 -binary | uuencode -m -
```

24.4.2 Disabling the Use of password-digest

Using password digests is so important that all Resin authenticators use it by default. Although it is really not advised, Resin's authenticators can be also be configured to use passwords that are not in digest form. You can do this by specifying

`password-digest="none"` as in the example below:

Disabling the Use of password-digest

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:XmlAuthenticator</type>
    <resin:password-digest>none</resin:password-digest>
    <resin:user name="harry" password="quidditch" group="user"/>
  </resin:XmlAuthenticator>
  ...
</web-app>
```

This technique can come in handy for development, testing, etc where password security is not critical.

24.4.3 Setting Password Digest Realm

The realm for Resin authenticators such as the DatabaseAuthenticator and the XmlAuthenticator defaults to "resin". However, if you want, you can explicitly specify the realm to be used for digesting like this:

Specifying a Realm

```
<web-app xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:DatabaseAuthenticator>
    <resin:password-digest-realm>hogwarts</resin:password-digest-realm>
    ...
  </resin:DatabaseAuthenticator>
  ...
</web-app>
```

24.5 Single Sign-on

Single sign-on refers to allowing for a single login for more than one context, for example, logging into all web-apps in a server at once without having to re-enter authentication information. Resin allows single sign-on by allowing you to place an authenticator at the host or server/cluster level instead of at the web-app level. The shared authenticator then applies to all the web applications under the host or server. Once you authenticate, the login will last for all the web-apps in that environment/scope.

For example, to configure an XML authenticator for all the web-apps in foo.com, you might configure as follows:

Single Sign-on for foo.com

```
<resin xmlns="http://caucho.com/ns/resin"
      xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="app-tier">
    <http port="8080"/>

    <host id="foo.com">
      <root-directory>/opt/foo.com</root-directory>

      <!-- Shared across the host -->
      <resin:XmlAuthenticator">
        <!-- password: quidditch -->
        <resin:user name="harry" password="uTOZTGaB6pooMDvqvl2LBu" group="user,gryffindor ↩
            "/>

        <!-- password: pureblood -->
        <resin:user name="dmalfoy" password="yI2uN1l97Rv5E6mdRnDFDB" group="user,slytherin ↩
            "/>
      </resin:XmlAuthenticator>

      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>
    </host>
  </cluster>
</resin>
```

Any .war in the webapps directory will share the same login across the host. Note, you will still need to implement a deploy-ref.xtp#login-config for each web-app in addition to login managers/authorization.

The value of reuse-session-id must be

`true` for single sign-on.

24.5.1 Single Sign-on for Virtual Hosts

The basis for establishing client identity is the standard JSESSIONID session cookie. If single sign-on is desired for virtual hosts, Resin must be configured to notify the browser of the proper domain name for the cookie so that the same JSESSIONID cookie is submitted by the browser to each virtual host.

In this case, an authenticator is placed at the cluster level so that it is common to all virtual hosts. The deploy-ref.xtp#session-config is placed in a deploy-ref.xtp#web-app-default at the cluster level so that it is applied as the default for all webapps in all virtual hosts.

The following example shows how you can configure single sign-on for two different sub-domains:

Single Sign-on for gryffindor.hogwarts.com and slytherin.hogwarts.com

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="app-tier>
```

```
    <http port="8080"/>

    <!-- Shared across all hosts -->
    <resin:XmlAuthenticator">
      <user name="Harry" password="..."/>
    </resin:XmlAuthenticator>

    <web-app-default>
      <session-config>
        <cookie-domain>.hogwarts.com</cookie-domain>
      </session-config>
    </web-app-default>

    <host id="gryffindor.hogwarts.com">
      ...
    </host>

    <host id="slytherin.hogwarts.com">
      ...
    </host>
  </cluster>
</resin>
```

Because of the way that browsers are restricted by the HTTP specification from submitting cookies to servers, it is not possible to have a single sign-on for virtual hosts that do not share some portion of their domain name. For example, "gryffindor.com" and "slytherin.com" cannot share a common authentication.

24.6 Login Managers

Authenticators manage how authentication data is stored, how the user-provided login information is matched to the stored authentication information and how an authenticated principal is constructed. A login manager, on the other hand, controls how the login information is actually collected. HTTP basic authentication is the simplest authentication method (the variety that causes a login/password prompt to appear on the browser when you access a URL). The following are the login managers Resin currently supports:

Table 24.2: Login Managers

NAME	DESCRIPTION
http://caucho.com/resin-javadoc/com/caucho/security/-BasicLogin.html	HTTP basic authentication.
http://caucho.com/resin-javadoc/com/caucho/security/-DigestLogin.html	HTTP digest authentication.
http://caucho.com/resin-javadoc/com/caucho/security/-FormLogin.html	Form-based authentication.
http://caucho.com/resin-javadoc/com/caucho/security/-AbstractLogin.html	Abstract class for custom login managers.

Each login manager is described in detail security-authentication-method.xtp , including example code that you could use as a starting point for your application. The built-in login managers should satisfy a large number of common cases, but you can easily create your own custom login manager when needed.

24.7 Authorization

Authorization is the process of verifying that an authenticated user has the appropriate privileges to access a secure resource. The typical authorization process verifies that a user has the right set of permissions to access a URL or is assigned to the correct role/group.

Resin has a very robust set of built-in authorization rules including conditionals, role-based security and combining/chaining rules. Authorization rules are based on a basic URL pattern based top level <resin:Allow>/<resin:Deny> tag set. Any conditionals, if applicable, are nested within these top level tags:

Table 24.3: Basic Authorization

NAME	DESCRIPTION
http://caucho.com/resin-javadoc/com/caucho/security/-Allow.html	Allows access to a URL pattern.
http://caucho.com/resin-javadoc/com/caucho/security/-Deny.html	Denies access to a URL pattern.

The allow/deny high-level directives can be qualified through a set of conditionals that include the most common case of role-based security (shown in the initial example):

Table 24.4: Basic Conditions

NAME	DESCRIPTION
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfAuthType.html	Checks for the authentication type, request.getAuthType().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfCookie.html	Checks for the presence of a named HTTP cookie from request.getCookies().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfCron.html	Matches if the current time is in an active range configured by cron-style times.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfFileExists.html	Matches if the URL corresponds to an actual file.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfHeader.html	Tests for a HTTP header and value match.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfLocale.html	Tests for a Locale match from the HTTP request.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfLocalPort.html	Compares the local port of the request, request.getLocalPort().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfMethod.html	Compares the HTTP method, request.getMethod().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfNetwork.html	Compares the remote IP address to a network pattern like 192.168/16.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfQueryParam.html	Tests for a HTTP query parameger, request.getParameter().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfRemoteAddr.html	Tests against the remote IP address, request.getRemoteAddr().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfRemoteUser.html	Tests against the remote user, request.getRemoteUser().
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfSecure.html	True for SSL requests, i.e. if request.isSecure() is true.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-IfUserInRole.html	Tests is the user is in the role.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-RequestPredicate.html	Interface for custom request predicates.

These conditionals can also be combined/chained as needed using the following tags:

Table 24.5: Combining Conditions

NAME	DESCRIPTION
http://caucho.com/resin-javadoc/com/caucho/rewrite/-And.html	Matches if all children match.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-Or.html	Matches if any children match.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-Not.html	Matches if the child does not match.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-NotAnd.html	Matches if any child does not match.
http://caucho.com/resin-javadoc/com/caucho/rewrite/-NotOr.html	Matches if all the children do not match.

Each authorization tag is described in detail security-authorization.xtp , including example code that you could use as a starting point for your application as well as common usage patterns. The built-in rules should satisfy a large number of common cases, but you can easily create your own custom predicate when needed.

24.8 What SSL provides

SSL provides two kinds of protection, encryption and server authentication .

24.8.1 Encryption

A set of bytes used to encrypt data and verify signatures . The key is public because it can be made available without a loss of security. The public key can only be used for encryption; it cannot decrypt anything. A public key always has a corresponding private key .

SSL provides encryption of the data traffic betweeen a client and a server. When the traffic is encrypted, an interception of that traffic will not reveal the contents because they have been encrypted - it will be unusable nonsense. A set of bytes used to decrypt data and generate signatures . The key is private because it must be kept secret or there will be a loss of security. The private key is used for decryption of data that has been encrypted with the corresponding public key .

SSL uses public key cryptography. Public key cryptography is based upon a pair of keys, the public key and the private key. The public key is used to encrypt the data. Only the corresponding private key can successfully decrypt the data.

For example, when a browser connects to Resin, Resin provides the browser a public key. The browser uses the public key to encrypt the data, and Resin uses the private key to decrypt the data. For this reason, it is important that you never allow anyone access to the private key, if the private key is obtained by someone then they can use it to decrypt the data traffic.

Encryption is arguably the more important of the security meausures that SSL provides.

24.8.2 Server Authentication

A combination of a private key , identity information (such as company name), and a signature generated by a signing authority . private key .

SSL also provides the ability for a client to verify the identity of a server. This is used to protect against identity theft, where for example a malicious person imitates your server or redirects client traffic to a different server while pretending to be you.

A company that is trusted to sign certificates. Browsers include certificates of signing authorities that they trust.

Server authentication uses the signature aspect of public key cryptography. The private key is used to sign messages, and the public key is used to verify the signature. With SSL, the validity of signatures depends upon signing authorities. Signing authorites (also called certificate authorities) are companies who have generated public keys that are included with browser software. The browser knows it can trust the signing authority, and the signing authority signs your SSL certificate, putting its stamp of approval on the information in your certificate.

Another name for signing authority . A company that is trusted to sign certificates. Browsers include certificates of signing authorities that they trust.

For example, after you generate your public and private key, you then generate a signing request and send it to a signing authority. This signing request contains information about your identity, this identity information is confirmed by the signing authority and ultimately displayed to the user of the browser. The signing authority validates the identity information you have provided and uses their private key to sign, and then returns a certificate to you. This certificate contains the identity information and your public key, verified by the signing authority, and is provided to the browser. Since the browser has the public key of the signing authority, it can recognize the signature and know that the identity information has been provided by someone that can be trusted.

24.9 OpenSSL

OpenSSL is the same SSL implementation that Apache's mod_ssl uses. Since OpenSSL uses the same certificate as Apache, you can get signed certificates using the same method as for Apache's mod_ssl or following the OpenSSL instructions.

24.9.1 Linking to the OpenSSL Libraries on Unix

On Unix systems, Resin's libexec/libresinssl.so JNI library supports SSL using the http://www.openssl.org libraries. Although the ./configure script will detect many configurations, you can specify the openssl location directly:

```
resin> ./configure --with-openssl=/usr/local/ssl
```

24.9.2 Obtaining the OpenSSL Libraries on Windows

On Windows systems, the resinssl.dll includes JNI code to use OpenSSL libraries (it was in resin.dll in versions before 3.0). All you need to do is to obtain an OpenSSL binary distribution and install it.

Resin on Windows 32 is compiled against the Win32 binary, you can obtain an installation package http://www.slproweb.com .

Resin on Windows 64 is compiled against a Win64 binary, you can obtain an installation package http://www.deanlee.cn?p=60&cp=1 .

Once you have run the installation package, you can copy the necessary dll libraries into `$RESIN_HOME` :

Copying the Windows SSL libraries into $RESIN_HOME

```
C:\> cd %RESIN_HOME%
C:\resin-4.0.x> copy "C:\Program Files\GnuWin32\bin\libssl32.dll" .\libssl32.dll
C:\resin-4.0.x> copy "C:\Program Files\GnuWin32\bin\libeay32.dll" .\libeay32.dll
```

24.9.3 Preparing to use OpenSSL for making keys

You can make a `keys/` subdirectory of $RESIN_HOME to do your work from and as a place to store your generated keys.

$RESIN_HOME/keys

```
unix> cd $RESIN_HOME
unix> mkdir keys
unix> cd keys

win> cd %RESIN_HOME%
win> mkdir keys
win> cd keys
```

Using OpenSSL requires a configuration file. Unix users might find the default configuration file in `/usr/ssl/openssl.cnf` or `/usr/share/ssl/openssl.cnf` . Windows users may not have received one with their package.

Either way, it can be valuable to make your own `openssl.cnf` that is used just for generating the keys to use with Resin. You can use the following as a template for a file `$RESIN_HOME/keys/openssl.cnf` . You may want to fill in the `_default` values so you don't have to type them in every time.

$RESIN_HOME/keys/openssl.cnf

```
[ req ]
 default_bits            = 1024
 distinguished_name      = req_distinguished_name

[ req_distinguished_name ]
 C                      = 2 letter Country Code, for example US
 C_default              =
 ST                     = State or Province
 ST_default             =
 L                      = City
 L_default              =
 O                      = Organization Name
 O_default              =
 OU                     = Organizational Unit Name, for example 'Marketing'
 OU_default             =
 CN                     = your domain name, for example www.hogwarts.com
 CN_default             =
 emailAddress           = an email address
 emailAddress_default   =
```

24.9.4 Creating a private key

Create a private key for the server. You will be asked for a password - don't forget it! You will need this password anytime you want to do anything with this private key. But don't pick something you need to keep secret, you will need to put this password in the Resin configuration file.

creating the private key gryffindor.key

```
unix> openssl genrsa -des3 -out gryffindor.key 1024
win>  "C:\Program Files\GnuWin32\bin\openssl.exe" \
         genrsa -des3 -out gryffindor.key 1024
```

24.9.5 Creating a certificate

OpenSSL works by having a signed public key that corresponds to your private key. This signed public key is called a certificate . A certificate is what is sent to the browser.

You can create a self-signed certificate, or get a certificate that is signed by a certificate signer (CA).

24.9.5.1 Creating a self-signed certificate

You can create a certificate that is self-signed, which is good for testing or for saving you money. Since it is self-signed, browsers will not recognize the signature and will pop up a warning to browser users. Other than this warning, self-signed certificates work well. The browser cannot confirm that the server is who it says it is, but the data between the browser and the client is still encrypted.

creating a self-signed certificate gryffindor.crt

```
unix> openssl req -config ./openssl.cnf -new -key gryffindor.key \
        -x509 -out gryffindor.crt
win> "C:\Program Files\GnuWin32\bin\openssl.exe" req -config ./openssl.cnf \
         -new -key gryffindor.key -x509 -out gryffindor.crt
```

You will be asked to provide some information about the identity of your server, such as the name of your Organization etc. Common Name (CN) is your domain name, like: "www.gryffindor.com".

24.9.5.2 Creating a certificate request

To get a certificate that is signed by a CA, first you generate a certificate signing request (CSR).

creating a certificate request gryffindor.csr

```
unix> openssl req -new -config ./openssl.cnf -key gryffindor.key \
      -out gryffindor.csr
win> "C:\Program Files\GnuWin32\bin\openssl.exe" req -new \
      -config ./openssl.cnf  -key gryffindor.key -out gryffindor.csr
```

You will be asked to provide some information about the identity of your server, such as the name of your Organization etc. Common Name (CN) is your domain name, like: "www.gryffindor.com".

Send the CSR to a certificate signer (CA). You'll use the CA's instructions for Apache because the certificates are identical. Some commercial signers include: http://digitalid.verisign.com/server/apacheNotice.htm http://www.thawte.com/certs/server/-request.html

You'll receive a gryffindor.crt file.

Most browsers are configured to recognize the signature of signing authorities. Since they recognize the signature, they will not pop up a warning message the way they will with self-signed certificates. The browser can confirm that the server is who it says it is, and the data between the browser and the client is encrypted.

24.9.6 resin.xml - Configuring Resin to use your private key and certificate

The OpenSSL configuration has two tags certificate-file and certificate-key-file . These correspond exactly to mod_ssl's SSLCertificateFile and SSLCertificateKeyFile. So you can use the same certificates (and documentation) from mod_ssl for Resin.

The full set of parameters is in the port configuration.

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="http-tier">

  <server id="a" address="192.168.1.12">
    <http port="443">
      <openssl>
        <certificate-file>keys/gryffindor.crt</certificate-file>
        <certificate-key-file>keys/gryffindor.key</certificate-key-file>
        <password>my-password</password>
      </openssl>
   </http>
```

```
  </server>

  ...
</resin>
```

The default resin configuration allows you to setup open-ssl in resin.properties.

Setting up open ssl in resin.properties

```
# OpenSSL certificate configuration
openssl_file : key/gryffindor.crt
openssl_key : keys/gryffindor.key
openssl_password : my-password
```

24.9.7 Testing SSL with the browser

A quick test is the following JSP.

```
Secure? <%= request.isSecure() %>
```

24.9.8 Testing with openssl to test the server

The openssl tool can be used as a client, showing some interesting information about the conversation between the client and the server:

```
unix$ openssl s_client -connect www.some.host:443 -prexit
```

24.9.9 Certificate Chains

A certificate chain is used when the signing authority is not an authority trusted by the browser. In this case, the signing authority uses a certificate which is in turn signed by a trusted authority, giving a chain of

`[your certificate] ←– signed by – [untrusted signer] ←– signed by – [trusted signer]` .

The Resin config parameter certificate-chain-file is used to specify a certificate chain. It is used to reference a file that is a concatenation of: your certificate file the intermediate (untrusted) certificate the root (trusted) certificate.

The certificates must be in that order, and must be in PEM format.

24.9.9.1 Example certificate chain for Instant SSL

http://instantssl.com is a signing authority that is untrusted by most browsers. Comodo has their certificate signed by GTECyberTrust.

Comodo gives you three certificates:

`your_domain.crt` (signed by Comodo)

`ComodoSecurityServicesCA.crt` (signed by GTE CyberTrust)

`GTECyberTrustRoot.crt` (universally known root)

In addition to this, you have your key, `your_domain.key` . The contents of the file referred to by certificate-chain-file is a concatenation of the three certificates, in the correct order.

Creating a certificate chain file

```
$ cat your_domain.crt ComodoSecurityServicesCA.crt GTECyberTrustRoot.crt > chain.txt
```

resin.xml using a certificate chain file

```
<http port="443">
  <openssl>
    <certificate-key-file>keys/your_domain.key</certificate-key-file>
    <certificate-file>keys/your_domain.crt</certificate-file>
    <certificate-chain-file>keys/chain.txt</certificate-chain-file>
    <password>test123</password>
  </openssl>
</http>
```

24.10 JSSE

We recommend avoiding JSSE if possible. It is slower than using Resin's OpenSSL support and does not appear to be as stable as Apache or IIS (or Netscape/Zeus) for SSL support. In addition, JSSE is far more complicated to configure. While we've never received any problems with Resin using OpenSSL, or SSL from Apache or IIS, JSSE issues are fairly frequent.

24.10.1 Install JSSE from Sun

This section gives a quick guide to installing a test SSL configuration using Sun's JSSE. It avoids as many complications as possible and uses Sun's keytool to create a server certificate.

Resin's SSL support is provided by Sun's http://java.sun.com/products/jsse . Because of export restrictions, patents, etc, you'll need to download the JSSE distribution from Sun or get a commercial JSSE implementation.

More complete JSSE installation instructions for JSSE are at http://java.sun.com/products/jsse/install.html . First download Sun's http://java.sun.com/products/jsse . Uncompress and extract the downloaded file. Install the JSSE jar files: jsse.jar, jnet.jar, and jcert.jar. You can either put them into the CLASSPATH or you can put them into $JAVA_HOME/jre/lib/ext. Since you will use "keytool" with the new jars, you need to make them visible to keytool. Just adding them to resin/lib is not enough. Register the JSSE provider (com.sun.net.ssl.internal.ssl.Provider). Modify $JAVA_HOME/jre/lib/security/java.security so it contains something like:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

Adding the JSSE provider allows "keytool" to create a key using the RSA algorithm.

24.10.2 Create a test server certificate

The server certificate is the core of SSL. It will identify your server and contain the secret key to make encryption work. Sun's keytool A self-signed certificate using open_ssl A test certificate from Thawte A production certificate from one of the certificate authorities (Verisign, Thawte, etc)

In this case, we're using Sun's keytool to generate the server certificate. Here's how:

```
resin1.2.b2> mkdir keys
resin1.2.b2> keytool -genkey -keyalg RSA -keystore keys/server.keystore
Enter keystore password:  changeit
What is your first and last name?
  [Unknown]:  www.caucho.com
What is the name of your organizational unit?
  [Unknown]:  Resin Engineering
What is the name of your organization?
  [Unknown]:  Caucho Technology, Inc.
What is the name of your City or Locality?
  [Unknown]:  San Francisco
What is the name of your State or Province?
```

```
  [Unknown]:  California
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=www.caucho.com, OU=Resin Engineering,
  O="Caucho Technology, Inc.", L=San Francisco, ST=California, C=US> correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):  changeit
```

Currently, the key password and the keystore password must be the same.

24.10.3 resin.xml

The Resin SSL configuration extends the http configuration with a few new elements.

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="">

    <server-default>

    <http port="8443">
     <jsse-ssl>
       <key-store-type>jks</key-store-type>
       <key-store-file>keys/server.keystore</key-store-file>
       <password>changeit</password>
     </jsse-ssl>
    </http>

    </server-default>
    ...

  </cluster>
</resin>
```

24.10.4 Testing JSSE

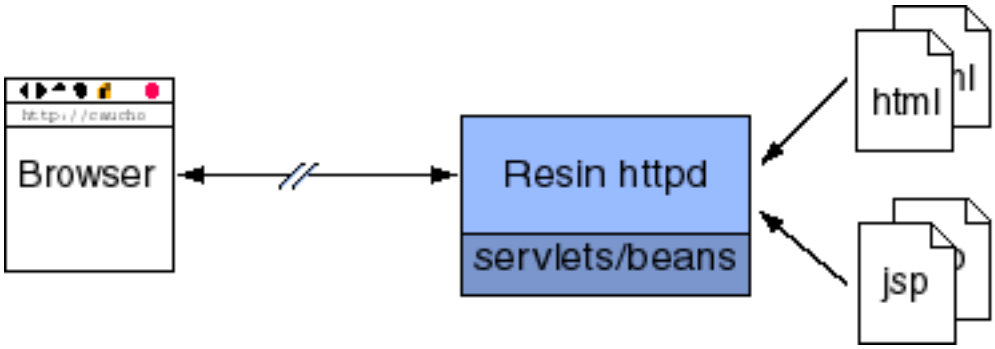
With the above configuration, you can test SSL with https://localhost:8443. A quick test is the following JSP.

```
Secure? <%= request.isSecure() %>
```

Chapter 25

Web Server: HTTP Server

25.1 HTTP Server Overview



The server listens at port 8080 in the default configuration and can be changed to the HTTP port 80 during deployment.

25.2 Unix, Linux, and Mac OS X

25.2.1 Getting Started

The following steps will start Resin for development: Install JDK 1.6 or later and link /usr/java to the Java home or define the environment variable JAVA_HOME `.tar -vzxf resin-4.0.x.tar.gz`

`cd resin-4.0.x`

`./configure --prefix=pwd` some starting-resin-command-line.xtp are available `make`

`make install` Execute `bin/resin.sh console` Or run `java -jar lib/resin.jar console` Browse to http://localhost:8080

Successful Foreground Startup Output

```
unix> bin/resin.sh console
May 6, 2011 3:06:05 PM com.caucho.boot.WatchdogChildTask run
INFO: WatchdogChild[] starting
May 6, 2011 3:06:05 PM com.caucho.boot.WatchdogChildProcess run
WARNING: Watchdog starting Resin[]
Resin Professional 4.0.17 (built Fri, 15 Apr 2011 06:35:56 PDT)
Copyright(c) 1998-2010 Caucho Technology.  All rights reserved.

  current.license -- 1 Resin server Caucho
```

```
Starting Resin Professional on Fri, 06 May 2011 15:06:06 -0400 (EDT)

[11-05-06 15:06:07.824] {main} Proxy Cache disk-size=1024M memory-size=64M
[11-05-06 15:06:08.179] {main}
[11-05-06 15:06:08.179] {main} Mac OS X 10.6.7 x86_64
[11-05-06 15:06:08.179] {main} Java(TM) SE Runtime Environment 1.6.0_24-b07-334-10M3326,  ↩
    MacRoman, en
[11-05-06 15:06:08.179] {main} Java HotSpot(TM) 64-Bit Server VM 19.1-b02-334, 64, mixed  ↩
    mode, Apple Inc.
[11-05-06 15:06:08.179] {main}
[11-05-06 15:06:08.179] {main} user.name  = caucho
[11-05-06 15:06:08.472] {main}
[11-05-06 15:06:08.479] {main} server listening to localhost:6800
[11-05-06 15:06:08.555] {main}
[11-05-06 15:06:08.873] {main}
[11-05-06 15:06:08.874] {main} resin.home = /Users/caucho/resin-pro-4.0.17/
[11-05-06 15:06:08.878] {main} resin.root = /Users/caucho/resin-pro-4.0.17/
[11-05-06 15:06:08.879] {main} resin.conf = /Users/caucho/resin-pro-4.0.17/conf/resin.xml
[11-05-06 15:06:08.889] {main}
[11-05-06 15:06:08.889] {main} server    = 127.0.0.1:6800 (app-tier:default)
[11-05-06 15:06:08.899] {main} stage      = production
[11-05-06 15:06:09.526] {main} WebApp[production/webapp/default/resin-admin] active
[11-05-06 15:06:10.245] {main} WebApp[production/webapp/default/resin-doc] active
[11-05-06 15:06:10.445] {main} WebApp[production/webapp/default/ROOT] active
[11-05-06 15:06:10.446] {main} Host[production/host/default] active
[11-05-06 15:06:10.447] {main} ProServer[id=default,cluster=app-tier] active
[11-05-06 15:06:10.448] {main}   JNI: file, nio keepalive (max=9984), socket
[11-05-06 15:06:10.448] {main}
[11-05-06 15:06:10.449] {main}
[11-05-06 15:06:10.450] {main} http listening to *:8080
[11-05-06 15:06:11.023] {main} https listening to *:8443
[11-05-06 15:06:11.092] {main}
[11-05-06 15:06:11.160] {main} ProResin[id=default] started in 4222ms
```

25.2.2 Deployment Directories

When deploying, it's a good idea to create a bit of structure to make Resin and website upgrades easier and more maintainable. Create a user to run Resin (e.g. resin or another non-root user) Link /usr/local/share/resin to the current Resin directory. This is $RESIN_HOME . Create a deployment root, e.g. /var/resin , owned by the resin user. This is $RESIN_ROOT . Put the modified resin.xml in /etc/resin/resin.conf Put the site documents in /var/resin/webapps/ROOT . Put any .war files in /var/resin/webapps . Put any virtual hosts in /var/resin/hosts/www.foo.com . Refer to http-virtual-hosts.xtp for more information. Output logs will appear in /var/resin/log . Create a startup script and configure the server to start it when the machine reboots.

25.2.3 Startup Script

If you installed using the .deb package or ran " `sudo make install` ", the installer created a file named /etc/init.d/resin which contains a standard Unix init.d startup file. This file will start Resin when invoked as:

```
/etc/init.d/resin start
```

Use the tools that came with your system to execute the script on startup.

Or you can create your own startup script which will start and stop the #Running Resin , and will pass any #command-line . The script might typically do a number of things: Configure the location of Java in JAVA_HOME Configure the location of Resin in RESIN_HOME Configure your web site directory in RESIN_ROOT Select a server and pid file if you have multiple Resin servers. Start and stop the #Running Resin .

The start script might look like:

Example start.sh script

```
#!/bin/sh

JAVA_HOME=/usr/java
RESIN_HOME=/usr/local/share/resin
RESIN_ROOT=/var/resin

java=$JAVA_HOME/bin/java

export JAVA_HOME
export RESIN_HOME
export RESIN_ROOT

$java -jar $RESIN_HOME/lib/resin.jar \
      -root-directory $RESIN_ROOT \
      -conf /etc/resin/resin.xml \
      -server a \
       $*
```

This script would be called as `./start.sh start` to start and `./start.sh stop` to stop.

The `-server` argument is only necessary if you have multiple servers (JVM instances) either on different machines or the same machine. The clustering-overview.xtp and deploy-ref.xtp#session-config pages describe when you might use `-server` .

25.3 Windows

25.3.1 Getting Started

Install JDK 1.6 or later. Check that the environment variable JAVA_HOME is set to the JDK location, e.g. `c:\java\jdk1.6.0_14` Unzip resin-4.0.x.zip Define the environment variable RESIN_HOME to the location of Resin, for example `c:\resin-4.0.x` Execute `resin.exe` Browse to http://localhost:8080

Starting on Win32

```
C:\resin-pro-4.0.17>resin.exe
May 6, 2011 4:41:28 PM com.caucho.boot.WatchdogChildTask run
INFO: WatchdogChild[] starting
May 6, 2011 4:41:28 PM com.caucho.boot.WatchdogChildProcess run
WARNING: Watchdog starting Resin[]
Resin Professional 4.0.17 (built Fri, 15 Apr 2011 06:35:56 PDT)
Copyright(c) 1998-2010 Caucho Technology.  All rights reserved.

  current.license -- 1 Resin server Caucho

Starting Resin Professional on Fri, 06 May 2011 16:41:29 -0400 (EDT)

[11-05-06 16:41:29.895] {main} Proxy Cache disk-size=1024M memory-size=64M
[11-05-06 16:41:30.083] {main}
[11-05-06 16:41:30.083] {main} Windows Vista 6.0 amd64
[11-05-06 16:41:30.083] {main} Java(TM) SE Runtime Environment 1.6.0_23-b05, Cp1252, en
[11-05-06 16:41:30.084] {main} Java HotSpot(TM) 64-Bit Server VM 19.0-b09, 64, mixed mode,  ↩
    Sun Microsystems Inc.
[11-05-06 16:41:30.084] {main}
[11-05-06 16:41:30.084] {main} user.name  = caucho
[11-05-06 16:41:30.327] {main}
[11-05-06 16:41:30.329] {main} server listening to 127.0.0.1:6800
[11-05-06 16:41:30.337] {main}
[11-05-06 16:41:30.604] {main}
[11-05-06 16:41:30.604] {main} resin.home = C:\resin-pro-4.0.17
```

```
[11-05-06 16:41:30.605] {main} resin.root = C:\resin-pro-4.0.17
[11-05-06 16:41:30.605] {main} resin.conf = C:\resin-pro-4.0.17\conf\resin.xml
[11-05-06 16:41:30.605] {main}
[11-05-06 16:41:30.605] {main} server    = 127.0.0.1:6800 (app-tier:default)
[11-05-06 16:41:30.605] {main} stage      = production
[11-05-06 16:41:31.037] {main} WebApp[production/webapp/default/resin-admin] active
[11-05-06 16:41:31.573] {main} WebApp[production/webapp/default/resin-doc] active
[11-05-06 16:41:31.648] {main} WebApp[production/webapp/default/ROOT] active
[11-05-06 16:41:31.648] {main} Host[production/host/default] active
[11-05-06 16:41:31.649] {main} ProServer[id=default,cluster=app-tier] active
[11-05-06 16:41:31.649] {main}   JNI keepalive: not available on Windows
[11-05-06 16:41:31.649] {main}   JNI: file, socket
[11-05-06 16:41:31.649] {main}
[11-05-06 16:41:31.649] {main}
[11-05-06 16:41:31.650] {main} http listening to *:8080
[11-05-06 16:41:32.022] {main} https listening to *:8443
[11-05-06 16:41:32.038] {main}
[11-05-06 16:41:32.057] {main} ProResin[id=default] started in 2774ms
```

25.3.2 Deploying as a Windows Service

To install the service run setup.exe

You will either need to reboot the machine or start the service from the Control Panel/Services panel to start the server. On a machine reboot, Windows will automatically start the web server.

You can also start and stop the service from the command-line:

```
C:\> net start resin
...
C:\> net stop resin
```

25.4 Running Resin

25.4.1 Processes Overview

Resin runs as multiple processes that begin with the following JVM command:

```
unix> bin/resin.sh /usr/local/share/resin/lib/resin.jar \
          -conf /etc/resin/resin.xml \
          start
```

The -jar argument tells java to run the Main-Class defined in resin.jar's manifest. The -conf argument specifies the path to your Resin configuration file. Lastly, Resin accepts start , stop , and restart arguments which are passed to the #The Watchdog Process . An additional command-line option, -server is used in clustering-overview.xtp .

JDK 1.6 includes a http://download.oracle.com/javase/6/docs/technotes/tools/share/jps.html command which will show the pids of any java processes.

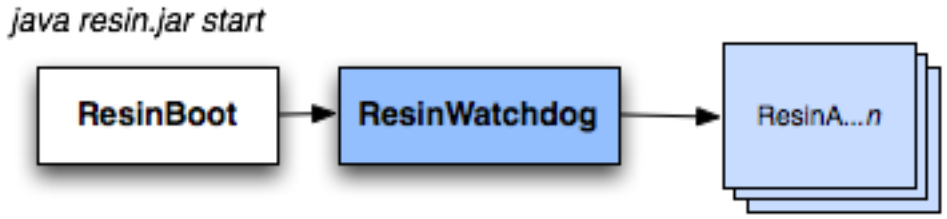
Example jps Process List

```
unix> jps
2098 Jps
2064 ResinWatchdogManager
2097 Resin
```

When running as a daemon (eg, `resin.sh start`) ResinWatchdogManager is the watchdog and Resin is the actual Resin instance. When running Resin as a foreground process, the process list displays resin.jar , which acts as the watchdog.

The first process that starts is the actual startup program,

`java -jar resin.jar` . It passes command-line arguments to the second process, the ResinWatchdogManager. This watchdog process takes care of starting the actual Resin process(es). ResinWatchdogManager monitors the state of Resin and restarts it if necessary, improving reliability.



25.4.2 The Watchdog Process

The ResinWatchdogManager is the parent process, providing automatic restarting Resin in cases of failure, and providing a single point of control for the start , stop and restart of all Resin processes. It is responsible for launching Resin with the correct JVM arguments and environment options such as starting Resin as the specified user, e.g. for < user-name > on unix.

ResinWatchdogManager watches Resin via a Socket connection. When the watchdog socket closes, Resin shuts itself down gracefully. The watchdog closes the socket on a stop or restart or if the watchdog itself is killed. If Resin exits for any reason, the watchdog will automatically start a new Resin process. This socket connection approach avoids the need for any signals or actual killing of Resin from the watchdog, and also makes it easy to stop all the Resins if necessary by just killing the watchdog.

The ResinWatchdogManager doesn't actually kill Resin or even check Resin's status, it just checks to see if Resin is alive or not. So if the JVM were to completely lock up, the watchdog would still think Resin was okay and would take no action.

25.4.3 Resin Processes

If Resin detects a major error (like running out of memory) or if the resin.xml changes, it will exit and the watchdog would start a new Resin instance. Reasons a Resin instance might exit include: resin.xml changes out of memory error detected deadlocks segv and other severe errors

Because the watchdog is always managing Resin processes, if you ever need to stop Resin with kill , you must kill the watchdog. Just killing the Resin process results in the watchdog restarting it automatically.

25.4.4 Logging

The watchdog will log to log/watchdog-manager.log . The Resin standard out/err is log/jvm-servername.log . ResinWatchdog-Manager is responsible for creating both of these log files, so jvm-servername.log is not really under the control of the Resin instance. This makes it somewhat more reliable in case of JVM deadlocks, etc.

Chapter 26

Web Server: URL Rewriting and Dispatch

26.1 Dispatch Rules

Resin's dispatching is based on a list of dispatch rules configured in the resin-web.xml or the resin.xml configuration files. Each rule has a regular expression matching request URLs. The first dispatch rule that matches takes control of the request. For example, a < resin:Redirect > sends a HTTP redirect, and a < resin:Dispatch > dispatches the request as normal.

Each matching rule can rewrite the URL using a target attribute. The target uses regexp replacement syntax like Perl's rewrite or sed. The following rule flips the first two segments around, so /foo/bar would become /bar/foo.

Example: redirect flipping

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="^/([^/]+)/([^/]+)" target="/$2/$1"/>

</web-app>
```

26.2 Conditions

Some dispatches might depend on request attributes like the security attribute. The http://caucho.com/resin-javadoc/com/caucho/-rewrite/IfSecure.html tag checks if the request is an SSL request, i.e. if request.isSecure() is true. For non-SSL requests, the following <resin:Forbidden> applies.

The rewrite conditions can all be used as security conditions, e.g. for <resin:Allow> or <resin:Deny>.

Example: dispatch on header

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:Forbidden regexp="^/secret">
    <resin:IfSecure value="false"/>
  </resin:Forbidden>

</web-app>
```

26.2.1 Basic Conditions

Basic conditions check the request and return true if the condition matches. Conditions can check on authentication (IsUserIn-Role), the remote IP (IfNetwork), check for SSL (IfSecure), and check for activation time (IfCron) or if a file exists (IfFileExists).

26.2.2 Combining Conditions

Conditions can be combined using <resin:And>, <resin:Not>, etc. tags.

26.3 Filter Actions

The rewrite capability can also add standard predefined filters to modify the output, e.g. setting a response header. Filters can use conditions as restrictions, just like the dispatch rules.

Example: SetHeader

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:SetHeader regexp="^/secret" name="Foo" value="bar"/>

</web-app>
```

26.3.1 Servlet Filters

Standard servlet filters can also be invoked as an action to the Dispatch target. Your filter is created using Java Injection syntax and will be applied if the Dispatch rule matches.

Example: dispatching with a custom filter

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:Dispatch regexp="^/test">
    <mypkg:MyFilter xmlns:my="urn:java:com.foo.mypkg">
      <mypkg:my-param>my-value</mypkg:my-param>
    </mypkg:MyFilter>
  </resin:Dispatch>

</web-app>
```

26.4 Logging and Debugging

Logging for the name `com.caucho.server.rewrite`

at the "finer" level reveals successful matches. At the "finest" level both successful and unsuccessful matches are logged.

Example: Logging example

```
<logger name="com.caucho.server.rewrite" level="finest"/>
```

```
[1998/05/08 02:51:31.000] forward ^/foo: '/baz/test.jsp'  no match
[1998/05/08 02:51:31.000] forward ^/bar: '/baz/test.jsp'  no match
[1998/05/08 02:51:31.000] forward ^/baz: '/baz/test.jsp'  -->  '/hogwarts/test.jsp'
```

26.5 Examples

26.5.1 Redirect http:// requests to https:// requests

The desired behaviour is to redirect plain connections to SSL connections.

Desired behaviour

```
  http://anything.com/anything.html
    redirect => https://anything.com/anything.html
```

Example: resin.xml configuration

```
<resin xmlns="http://caucho.com/ns/resin"
    xmlns:resin="urn:java:com.caucho.resin">

  <cluster ...>
  <host ...>
    ...
    <resin:Redirect regexp="^" target="https://${host.name}">
      <resin:IfSecure value="false"/>
    </resin:Redirect>
    ...
  </host>
</resin>
```

26.5.2 Make an alias for a web-app

The desired behaviour is to make it so that a web-app will match more than one url pattern. For example, a web-app is deployed in

`webapps/physics` and available at

`http://hostname/physics/` , the desired behaviour is to allow a request to `http://hostname/classroom/physics` to end up at the

`/physics` web-app.

Desired behaviour

```
    http://hostname/classroom/physics
      forward => http://hostname/physics

    http://hostname/classroom/physics/anything
      forward => http://hostname/physics/anything
```

The rewrite-dispatch tag is used at the <host> level. If it was placed in a <web-app> then it would be too late to forward to a different web-app because Resin would have already resolved the web-app.

Example: resin.xml configuration

```
<resin xmlns="http://caucho.com/ns/resin"
  xmlns:resin="urn:java:com.caucho.resin">

<cluster id="">
<host id="">

  <resin:Forward regexp="^/classroom/physics" target="/physics"/>
```

26.5.3 Forward based on host name

The desired behaviour is to forward requests internally based on the host name.

Desired behaviour

```
    http://gryffindor.hogwarts.com/anything.html
      forward => /gryffindor/*

    http://slytherin.hogwarts.com/anything.html
      forward => /slytherin/anything.html

    http://hogwarts.com/anything.html
      forward => /anything.html
```

The rewrite-dispatch tag is used at the <cluster> level. If it was placed in the <host> or the <web-app> then it would be too late to forward to a different host because Resin would have already resolved the host.

Example: resin.xml Configuration

```
<resin xmlns="http://caucho.com/ns/resin"
    xmlns:resin="urn:java:com.caucho.resin">

  <cluster>
      ...
    <resin:Forward regexp="http://gryffindor\.[^/]+" target="/gryffindor/"/>
    <resin:Forward regexp="http://slytherin\.[^/]+" target="/slytherin/"/>
   ...
</cluster>
</resin>
```

26.6 Tag listing by function

Table 26.1: Dispatch rules

NAME	DESCRIPTION
<resin:Dispatch>	Normal servlet dispatching with optional target rewriting.
<resin:FastCgiProxy>	Proxies the request to a backend server using FastCGI as a proxy protocol.
<resin:Forbidden>	Send a HTTP forbidden response.
<resin:Forward>	Forwards to the new URL using RequestDispatcher.forward with the target URL.
<resin:HttpProxy>	Proxies the request to a backend server using HTTP as a proxy protocol.
<resin:LoadBalance>	Load balance to a cluster of backend Resin servers.
<resin:MovedPermanently>	Send a HTTP redirect to a new URL specified by target .
<resin:Redirect>	Send a HTTP redirect to a new URL specified by target .
<resin:SendError>	Send a HTTP error response.
AbstractTargetDispatchRule	Base class for custom dispatch rules.

Table 26.2: Basic conditions

NAME	DESCRIPTION
<resin:IfAuthType>	Checks for the authentication type, request.getAuthType().

Table 26.2: (continued)

NAME	DESCRIPTION
<resin:IfCookie>	Checks for the presence of a named HTTP cookie from request.getCookies().
<resin:IfCron>	Matches if the current time is in an active range configured by cron-style times.
<resin:IfFileExists>	Matches if the URL corresponds to an actual file.
<resin:IfHeader>	Tests for a HTTP header and value match.
<resin:IfLocale>	Tests for a Locale match from the HTTP request.
<resin:IfLocalPort>	Compares the local port of the request, request.getLocalPort().
<resin:IfMethod>	Compares the HTTP method, request.getMethod().
<resin:IfNetwork>	Compares the remote IP address to a network pattern like 192.168/16.
<resin:IfQueryParam>	Tests for a HTTP query parameger, request.getParameter().
<resin:IfRemoteUser>	Tests against the remote user, request.getRemoteUser().
<resin:IfSecure>	True for SSL requests, i.e. if request.isSecure() is true.
<resin:IfUserInRole>	Tests is the user is in the servlet security role.
RequestPredicate	Interface for custom request predicates.

Table 26.3: Combining conditions

NAME	DESCRIPTION
<resin:And>	Matches if all children match.
<resin:Or>	Matches if any children match.
<resin:Not>	Matches if the child does not match.
<resin:NotAnd>	Matches if any child does not match.
<resin:NotOr>	Matches if all the children do not match.

Table 26.4: Rewrite filters

NAME	DESCRIPTION
<resin:SetHeader>	Sets a response header.
<resin:SetRequestSecure>	Marks the request as secure.
<resin:SetVary>	Sets the Vary header.
<mypkg:MyFilter>	Servlet filters.

Chapter 27

Web Server: Virtual Hosts

27.1 Overview

Virtual hosts are multiple internet domains served by the same Resin server. Because one JVM handles all the domains, its more memory and processing efficient, as well as sharing IP addresses. With Resin, adding virtual hosts can as easy as creating a directory like `/var/resin/hosts/foo.com` and setting up the DNS name. Explicit virtual host is also possible to match existing layouts, like matching a `/var/resin/htdocs` configuration when migrating a PHP mediawiki or wordpress site to use Quercus for security and performance.

The virtual host will contain one or more deploy-ref.xtp to serve the host's contents. Simple sites will use a fixed root webapp, like the Apache-style

`/var/resin/htdocs` . More complicated sites can use a

`webapps` -style directory.

Each virtual host belongs to a Resin <cluster>, even if the cluster has only a single server.

For example, a Resin server might manage both the

`www.gryffindor.com` and `www.slytherin.com` domains, storing the content in separate directories (/var/resin/-gryffindor and /var/resin/slytherin), and using a single IP address for both domains. In this scenario, both www.gryffindor.com and www.slytherin.com are registered with the standard domain name service registry as having the IP address `192.168.0.13` . When a user types in the url

`http://www.gryffindor.com/hello.jsp` in their browser, the browser will send the HTTP request to the IP address

`192.168.0.13` and send an additional HTTP header for the gryffindor host, "Host: www.gryffindor.com". When Resin receives the request it will grab the host header, and dispatch the request to the configured virtual host.

Example: HTTP request headers

```
C: GET /test.jsp HTTP/1.1
C: Host: www.gryffindor.com
C:
```

host name host aliases optional host.xml root directory web-applications configuration environment logging

27.2 Dynamic virtual hosts

Resin can deploy virtual hosts automatically by scanning a host deployment directory for virtual host content. Each sub-directory in the `hosts` directory will cause Resin to create a new virtual host. To customize the configuration, you can add a `host.xml` in the host's root directory for shared databases, beans or security, or to add http-virtual-hosts-ref.xtp#host-alias names.

You can add hosts dynamically to a running server just by creating a new host directory. Resin periodically scans the `hosts`

directory looking for directory changes. When it detects a new host directory, it will automatically start serving files from the new virtual hosts.

If you add a `default` directory in the `hosts`, Resin will use it to serve all unknown virtual hosts. The default host is handy for simple servers with only a single virtual host and for sites where the virtual host is handled in software, like Drupal. If the

`default` directory is missing, Resin will return `404 Not Found` for any unknown virtual hosts.

Example: virtual host directory structure

```
/var/resin/hosts/www.gryffindor.com/
                                 host.xml
                                 log/access.log
                                 webapps/ROOT/index.jsp
                                 webapps/ROOT/WEB-INF/resin-web.xml

/var/resin/hosts/www.slytherin.com/
                                host.xml
                                log/access.log
                                webapps/ROOT/index.php
                                webapps/ROOT/WEB-INF/resin-web.xml

/var/resin/hosts/default/
                      host.xml
                      log/access.log
                      webapps/ROOT/index.php
                      webapps/ROOT/WEB-INF/resin-web.xml
```

27.2.1 host-aliasing for dynamic hosts

Often, the same virtual host will respond to multiple names, like

`www.slytherin.com` and `slytherin.com`. One name is the primary name and the others are aliases. In Resin, the primary name is configured by the http-virtual-hosts-ref.xtp#host-name tag and aliases are configured by http-virtual-hosts-ref.xtp#host-alias . In a dynamic host configuration, the directory name is used as the `host-name`

by default, and aliases are declared in the `host.xml`.

Example: www.slytherin.com/host.xml

```
<host xmlns="http://caucho.com/ns/resin">
  <host-name>www.slytherin.com</host-name>
  <host-alias>slytherin.com</host-alias>
  <host-alias>quidditch.slytherin.com</host-alias>
</host>
```

Since the `host.xml` is shared for all web-applications in the host, you can also use it to configure shared resources like security logins, shared databases, and shared resources.

27.2.2 host-deploy configuration

The http-virtual-hosts-ref.xtp#host-deploy tag configures the dynamic virtual hosting specifying the directory where Resin should scan for virtual hosts. Because Resin does not automatically add default configuration, you will need to also add configuration for the

`host.xml`, `app-default.xml` and

`web-app-deploy`. Although it's a bit more verbose, the no-default rule makes Resin more secure and debuggable. If an item like a <web-app> is missing, Resin will return `404 Not Found` for security. Because all configuration is explicit, it's ultimately traceable to the `resin.xml` which makes debugging more reliable.

Shared host configuration goes in the http-virtual-hosts-ref.xtp#host-default tag. In this case, we've added an optional `host.xml` for configuration, an access log in `log/access.log` and a standard

`webapps` directory. The standard servlets and file handling come from the `app-default.xml` file. If you omit either the app-default.xml or the webapps, you will see `404 Not Found` for any requests.

The example below is a complete, working `resin.xml` listening to HTTP at port 8080. The clustering-overview.xtp consists of a single server. It includes a < development-mode-error-page > to help debugging the configuration. Many sites will omit the error-page to hide configuration details in case an error occurs on a live site.

Example: /etc/resin/resin.xml host-deploy configuration

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">
<cluster id="app-tier">
  <server id="app-a" address="192.168.1.13" port="6800">
    <http port="8080"/>
  </server>

  <development-mode-error-page/>

  <resin:import path="${__DIR__}/app-default.xml"/>

  <host-default>
    <resin:import path="host.xml" optional="true"/>

    <access-log path="log/access.log"/>

    <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>
  </host-default>

  <host-deploy path="hosts">
</host-deploy>

</cluster>
</resin>
```

Any directory created in `${resin.root}/hosts` will now become a virtual host. You can also place a .jar file in `${resin.root}`, it is expanded to become a virtual host.

```
${resin.root}/hosts/www.gryffindor.com/
${resin.root}/hosts/www.gryffindor.com/webapps/ROOT/index.jsp
${resin.root}/hosts/www.gryffindor.com/webapps/foo/index.jsp

${resin.root}/hosts/www.slytherin.com.jar
```

Jar libraries and class files that are shared amongst all webapps in the host can be placed in `lib` and `classes` subdirectories of the host:

```
${resin.root}/hosts/www.gryffindor.com/lib/mysql-connector-java-3.1.0-alpha-bin.jar
${resin.root}/hosts/www.gryffindor.com/classes/example/CustomAuthenticator.java
```

More information is available in the configuration documentation for < http-virtual-hosts-ref.xtp#host-deploy > and < http-virtual-hosts-ref.xtp#host-default >.

27.3 Explicit Virtual Hosting

In a more structured site, you can take complete control of the virtual host configuration and configure each virtual host explicitly. Existing sites wanting to upgrade to Resin or sites with extra security needs may prefer to configure each <host> in the resin.xml.

For example, a PHP Drupal site evaluating Quercus to improve performance and security might use the explicit <host> to point to the existing `/var/resin/htdocs` directory.

In the explicit configuration, each virtual host has its own http-virtual-hosts-ref.xtp#host block. At the very least, each host will define the id specifying the host name and a root web-app. A root-directory is often used to provide a host specific root for logfiles.

As with the dynamic hosting, servlets and web-apps must be configured either in a <host-default> or explicitly. If they are missing, Resin will return a `404 Not Found` for security. The host id="" is the default host and will serve any request that doesn't match other hosts. If you don't have a default host, Resin will return a

`404 Not Found` for any unknown host.

The following sample configuration defines an explicit virtual hosts www.slytherin.com and a default host, each with its own root directory, access-log and a single explicit deploy-ref.xtp in the `htdocs`

directory. The default virtual host is configured just like a typical Apache configuration, so it can be used to upgrade an Apache/PHP site to use Quercus for security and performance.

Example: /etc/resin/resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
        xmlns:resin="urn:java:com.caucho.resin">

<cluster id="app-tier">
  <server id="app-a" address="192.168.1.10" port="6800">
    <http port="8080"/>
  </server>

  <development-mode-error-page/>

  <resin:import path="${__DIR__}/app-default.xml"/>

  <host id="">
    <root-directory>/var/resin</root-directory>

    <access-log path="logs/access.log"/>

    <web-app id="" root-directory="htdocs"/>
  </host>

  <host id="www.slytherin.com">
    <host-alias>slytherin.com</host-alias>

    <root-directory>/var/slytherin</root-directory>

    <access-log path="logs/access.log"/>

    <web-app id="" root-directory="htdocs"/>
  </host>

</cluster>
</resin>
```

Browsing http://gryffindor.caucho.com/test.php will look for /var/resin/htdocs/test.php.

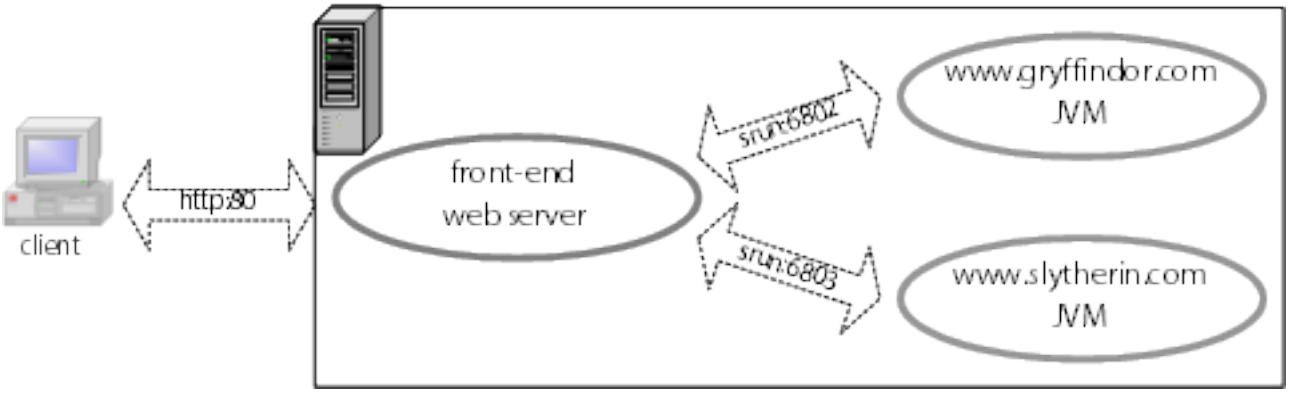
Browsing http://slytherin.caucho.com/test.php will look for /var/slytherin/htdocs/test.php.

27.4 Server per virtual host

In some ISP setups, it may make sense to assign a server for each virtual host. The isolation of web-apps may not be sufficient; each host needs a separate JVM. In this configuration, each <host> belongs to its own <cluster> and has a dedicated <server>. Normally, this configuration will operate using load-balancing, so the load-balance server will dispatch requests as appropriate.

For further security restrictions, see the health-watchdog.xtp section. ISPs can also use the watchdog to assign different <username> values for each host and can even create chroot directories for each JVM.

A front-end web server receives all requests, and is configured to dispatch to back-end Resin server that correspond to the host name.



27.4.1 Back-end JVMs

Each host is placed in its own <cluster> with a dedicated <server>. Since the server listens to a TCP port for load-balancing and clustering messages, each server on the maching needs a different server port.

In this example, the virtual hosts `www.gryffindor.com` and

`www.slytherin.com` each get their own server. The backend clusters have their own virtual host. The frontend load-balancer dispatches the clustering-overview.xtp tags to the backend.

This example is split into two blocks to emphasize the frontend and backend. Typically, they will both actually be in the same resin.xml to ensure consistency.

Example: /etc/resin/resin.xml for backend

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

  <cluster-default>
    <resin:import path="${resin.home}/conf/app-default.xml"/>

    <host-default>
      <web-app-deploy path="webapps"
           expand-preserve-fileset="WEB-INF/work/**"/>
    </host-default>
  </cluster-default>

  <cluster id="gryffindor>
    <server id="gryffindor" host="localhost" port="6800"/>

    <host id="www.gryffindor.com">

      <root-directory>/var/resin/gryffindor</root-directory>

    </host>
  </cluster>

  <cluster id="slytherin">
    <server id="slytherin" host="localhost" port="6801"/>

    <host id="www.slytherin.com">

      <root-directory>/var/resin/slytherin</root-directory>
```

```
    </host>
  </cluster>

  <cluster id="web-tier">
    <!-- see below -->
    ...
  </cluster>

</resin>
```

Each back-end server is started separately:

Example: starting backend servers

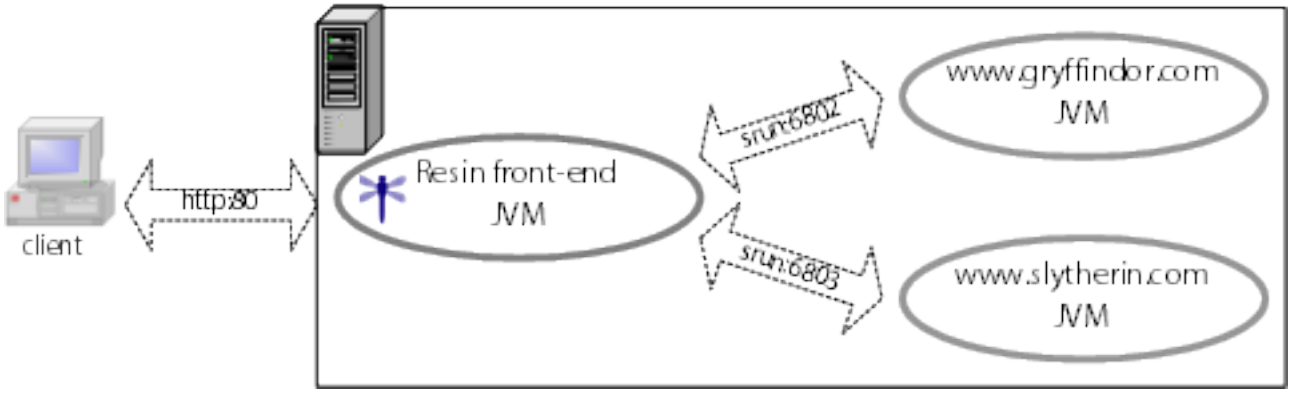
```
unix> bin/resin.sh -server gryffindor start
unix> bin/resin.sh -server slytherin start
```

Example: stopping backend servers

```
unix> bin/resin.sh -server gryffindor stop
unix> bin/resin.sh -server slytherin stop
```

27.4.2 Resin web-tier load balancer

The host-specific back-end servers are ready to receive requests on their server ports. A third Resin server can be used as the front-end load-balancer. It receives all requests and dispatches to the back-end servers.



The Resin web server is configured using http-rewrite.xtp with a clustering-overview.xtp directive to dispatch to the back-end server. A cluster is defined for each back-end host, so that the <load-balance> knows how to find them.

Example: /etc/resin/resin.xml for front-end web server

```
<resin xmlns="http://caucho.com/ns/resin"
     xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="web-tier">
    <server-default>
      <http port="80"/>
    </server-default>

    <server id="web" address="192.168.2.1" port="6800"/>

    <host id="gryffindor.com">
      <web-app id="/">

        <resin:LoadBalance regexp="" cluster="gryffindor"/>
```

```
      </web-app>
    </host>

    <host id="slytherin.com">
      <web-app id="/">

        <resin:LoadBalance regexp="" cluster="slytherin"/>

      </web-app>
    </host>
  </cluster>

  <cluster id="gryffindor">
    <server id="gryffindor" address="192.168.2.2" port="6800"/>

    <host id="www.gryffindor.com">
      ...
    </host>
  </cluster>

  <cluster id="slytherin">
    <server id="slytherin" address="192.168.2.2" port="6801"/>

    ...
  </cluster>
</resin>
```

27.4.2.1 Starting the servers on Unix

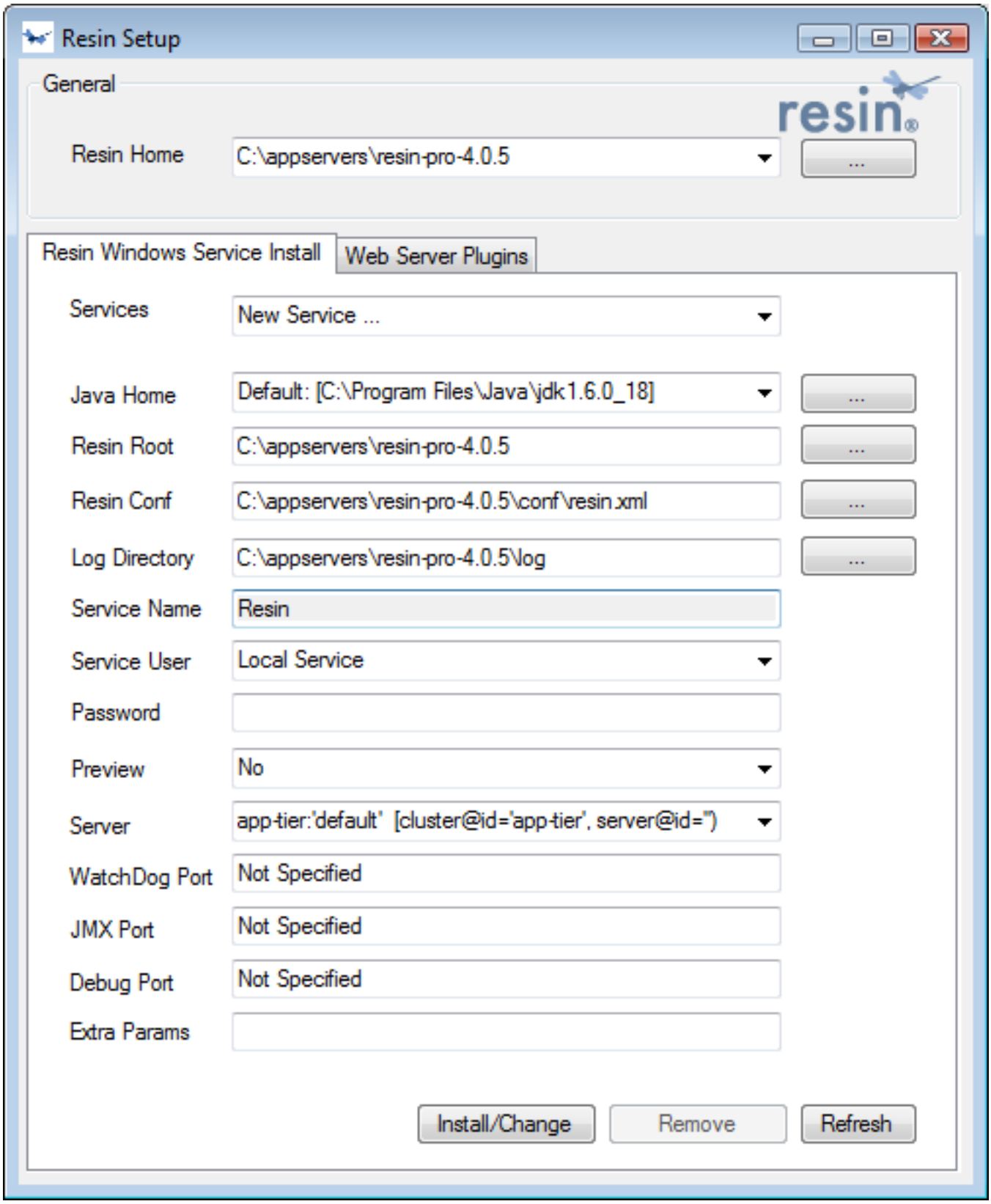
The front-end server JVM is started similar to the back-end JVMs:

Example: starting the load balancer

```
unix> bin/resin.sh -server web -conf conf/resin.xml start
...
unix> bin/resin.sh -server web -conf conf/resin.xml stop
```

27.4.2.2 Starting the servers on Windows

With Windows, each JVM is installed as a service. Service is installed using setup.exe graphical utility. It's possible to install a number of Resin services each using a unique name. The name will need to be supplied into *Service Name* field.



You will either need to reboot the machine or start the service from the Control Panel/Services panel to start the server. On a machine reboot, NT will automatically start the service.

There is a bug in many JDKs which cause the JDK to exit when the administrator logs out. JDK 1.4 and later can avoid that bug if the JDK is started with -Xrs.

27.5 Configuration tasks

27.5.1 host naming

The virtual host name can be configured by an explicit http-virtual-hosts-ref.xtp#host-name , a http-virtual-hosts-ref.xtp#host-alias , a http-virtual-hosts-ref.xtp#host-alias-regexp , by the http-virtual-hosts-ref.xtp#host tag or implicitly by the http-virtual-hosts-ref.xtp#host-deploy . For explicit configuration styles, the host name and alias configuration will generally be in the resin.xml. For dynamic configuration, the host aliases will typically be in an included `host.xml` inside the host directory.

The default host catches all unmatches hosts. Simpler sites will use the default host for all requests, while security-conscious sites may remove the default host entirely. If the default host is not configured, Resin will return a `404 Not Found` .

27.5.2 host.xml

The `host.xml` is an optional file where virtual hosts can put host-common configuration. The `host.xml`

is a good place for shared resources like authentication, database pools or host-wide config-candi.xtp . It's also a location for the <host-alias> in a dynamic hosting configuration.

The `host.xml` is configured in a <host-deploy> or <host-default> by adding a <resin:import> tag specifying the host.xml name and location. Because the <host-default> applies the <resin:import> to every virtual host, it becomes a common system-wide configuration file.

27.5.3 web-applications

Hosts must define deploy-ref.xtp in order to serve files, servlets, or PHP pages. If the host is missing all webapps, Resin will return `404 Not Found` for all requests make to the host.

Both explicit deploy-ref.xtp#web-app and dynamic deploy-ref.xtp#web-app-deploy tags are used to configure webapps. The explicit style is generally used for Apache-style configuration, while the dynamic style is generally used for Java app-server .war configuration.

Remember, Resin's default servlets like the file, JSP, and PHP servlets also need to be defined before they're used. So all Resin configuration files need to have a <resin:import> of the `conf/app-default.xml`

configuration file either in the <cluster> or in a shared <cluster-default>. If the app-default.xml is missing, Resin will not serve static files, JSP, or PHP, and will not even look in the WEB-INF for resin-web.xml, classes, or lib.

27.6 IP-Based Virtual Hosting

While Resin's virtual hosting is primarily aimed at named-based virtual hosts, it's possible to run Resin with IP-Based virtual hosts.

With IP virtual hosting, each <http> block is configured with the virtual host name. This configuration will override any virtual host supplied by the browser.

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="web-tier">
  <server id="a">

    <http address="192.168.0.1" port="80"
          virtual-host="slytherin.caucho.com"/>

    <http address="192.168.0.2" port="80"
          virtual-host="gryffindor.caucho.com"/>

  </server>
```

```
  ...

  <host id="slytherin.caucho.com">
    ...
  </host>
</cluster>
</resin>
```

27.7 Internationalization

Resin's virtual hosting understands host names encoded using rfc3490 (Internationalizing Domain Names in Applications). This support should be transparent. Just specify the virtual host as usual, and Resin will translate the brower's encoded host name the unicode string.

Support, of course, depends on the browser. http://devedge.netscape.com/viewsource/2003/idn/ supports the encoding.

27.8 Virtual Hosts with Apache or IIS

A common configuration uses virtual hosts with Apache or IIS. As usual, Apache or IIS will pass matching requests to Resin.

27.8.1 Apache

The Resin JVM configuration with Apache is identical to the standalone configuration. That similarity makes it easy to debug the Apache configuration by retreating to Resin standalone if needed.

The ServerName directive in Apache with the UseCanonicalName can be used to select a canonical name for the virtual host virtual hosting work. When Apache passes the request to Resin, it tells Resin the ServerName . Without the ServerName , Apache will use the "Host:" header in the HTTP request to select which host to serve.

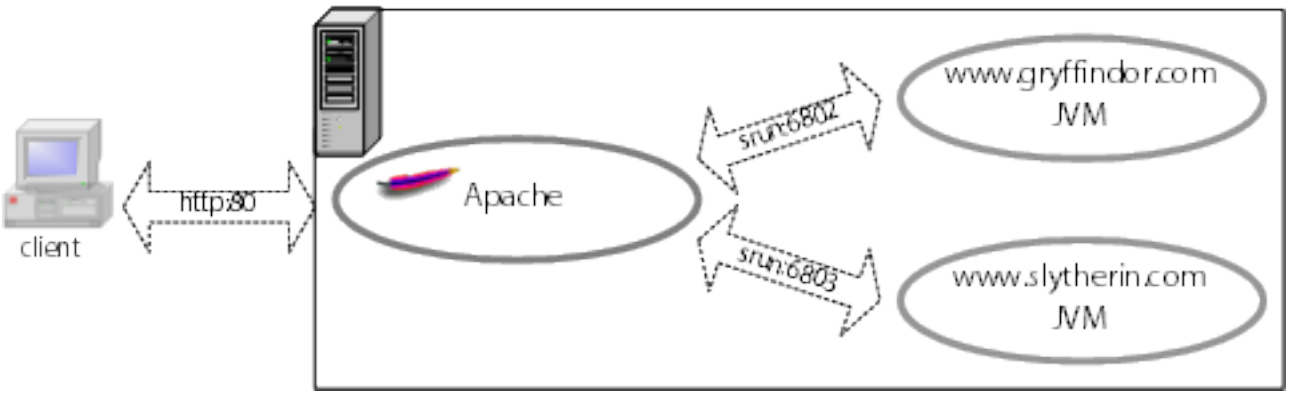
httpd.conf

```
LoadModule caucho_module /usr/local/apache/libexec/mod_caucho.so

ResinConfigServer localhost 6802

UseCanonicalName on

<VirtualHost 127.0.0.1>
  ServerName gryffindor.caucho.com
</VirtualHost>

<VirtualHost 192.168.0.1>
  ServerName slytherin.caucho.com
</VirtualHost>
```

You'll the LoadModule must appear before the ResinConfigServer for Apache to properly understand the ResinConfigServer command. If they're missing, Apache will send an error.

27.8.2 Apache front-end

The host-specific back-end JVMs are ready to receive requests on their server ports. Apache is the front-end server, and is configured to dispatch to the appropriate back-end Resin JVM for the host:



httpd.conf

```
UseCanonicalName on

<VirtualHost 127.0.0.1>
  ServerName gryffindor.caucho.com
  ResinConfigServer 192.168.0.10 6800
</VirtualHost>

<VirtualHost 192.168.0.1>
  ServerName slytherin.caucho.com
  ResinConfigServer 192.168.0.11 6800
</VirtualHost>
```

When you restart the Apache web server, you can look at http://gryffindor/caucho-status and http://slytherin/caucho-status to check your configuration. Check that each virtual host is using the server address and port that you expect.

27.9 Testing virtual hosts

During development and testing, it is often inconvenient or impossible to use real virtual host names that are registered as internet sites, and resolve to an internet-available IP address. OS-level features on the test client machine can be used to map a virtual host name to an IP address.

For example, developers often run the Resin server and the test client (usually a browser) on the same machine. The OS is configured to map the "www.gryffindor.com" and "www.slytherin.com" names to "127.0.0.1", pointing these host names back to computer that the client is running on.

Unix users edit the file `/etc/hosts` :

/etc/hosts

```
127.0.0.1       localhost

127.0.0.1       www.gryffindor.com
127.0.0.1       www.slytherin.com
```

Windows user edit the file `C:\WINDOWS\SYSTEM32\DRIVERS\ETC\HOSTS` :

C:\WINDOWS\SYSTEM32\DRIVERS\ETC\HOSTS

```
127.0.0.1       localhost

127.0.0.1       www.gryffindor.com
127.0.0.1       www.slytherin.com
```

27.10 Deployment

27.10.1 Overriding web-app-deploy configuration

The web-app-deploy can override configuration for an expanded war with a matching <web-app> inside the <web-app-deploy>. The <document-directory> is used to match web-apps.

Example: resin.xml overriding web.xml

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">
<host id="">

<web-app-deploy path="webapps">
  <web-app context-path="/wiki"
             document-directory="wiki">
    <context-param database="jdbc/wiki">
  </web-app>
</web-app-deploy>

</host>
</cluster>
</resin>
```

27.10.2 versioning

The versioning attribute of the <web-app-deploy> tag improves web-app version updates by enabling a graceful update of sessions. The web-apps are named with numeric suffixes, e.g. foo-10, foo-11, etc, and can be browsed as /foo. When a new version of the web-app is deployed, Resin continues to send current session requests to the previous web-app. New sessions go to the new web-app version. So users will not be aware of the application upgrade.

Chapter 28

Web Server: Web Apps

28.1 Overview

configuration Deploying with the command-line. Servlets and Filters resin:import app-default.xml Resources: databases, queues, beans. URL rewrite Security

28.2 defining web-xml in resin.xml

All web-apps are defined in the resin.xml, because Resin uses explicit configuration over defaults. While this gives flexibility, most sites use one of a few basic configurations. The most common is a <web-app-deploy> which defines a dynamic directory for both command-line deployment and a place to copy .war files.

28.2.1 web-app-deploy

The following example shows the most minimal configuration for a web app deployment using the webapps directory. The <web-app-deploy> enables both command-line and explicit jar deployment. The <resin:import> of app-default.xml defines default servlet behavior like WEB-INF/web.xml, WEB-INF/lib, JSPs and the static file servlet. The default <cluster> and default <host> give a place for the web-app to exist.

Example: basic web-app-deploy

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

<cluster id="">

  <resin:import path="classpath:META-INF/app-default.xml"/>

<host id="">

  <web-app-deploy path="webapps"/>

</host>

</cluster>

</resin>
```

28.2.2 web-app

In some deployments, the web-apps are defined in the resin.xml instead of a general deployment in a webapps directory. A site might want more specific control in the resin.xml or might just prefer to have all the web-apps listed explicitly. The <web-app> tag defines a new web-application.

Example: explicit web-app

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

<cluster id="">

  <resin:import path="classpath:META-INF/app-default.xml"/>

<host id="">

  <web-app id="/my-web-app root-directory="my-root-dir"/>

</host>

</cluster>

</resin>
```

Each web-app has several components: the URL path, the root directory, an archive path, the internal cloud identifier, and the internal cloud archive. URL path - the context-path for servlets. It's the URL prefix that dispatches to the web-app. root-directory - the expanded directory on the filesystem which contains the files and classes for the web-app. archive-path - the path to a .war file with the archived web-app (optional). internal id - the internal unique cloud name of the web-app, typically something like "production/webapp/foo.com/my-web-app". internal cloud archive - an internal copy of the web-app archive which is distributed across all servers in the cloud.

28.2.3 web-app deployment in the cloud and the command-line

When you deploy a .war on the command-line, Resin first stores the archive in its internal cloud repository using the internal unique identifier. The <web-app> or <web-app-deploy> will look in the internal repository for that name and deploy it just as if it was a .war file.

For example, the default root web-app is named "production/webapp/default/ROOT". When you deploy the ROOT.war to the default virtual-host, Resin saves it internally as "production/webapp/default/ROOT", copies the .war to all servers in the cluster, and then expands it just like a ROOT.war file. The ROOT.war is saved in triplicate on the three triad servers for reliability.

Thinking of the command-line deployment as an internal cloud archive is an accurate model.

28.3 web-app-default

Web-app configuration can be shared across several web-apps using the web-app-default tag. The contents are applied to all web-apps. Resin's app-default.xml uses the web-app-default to define standard servlets like the *.jsp mapping, the WEB-INF/lib directory and the static file servlet.

If you have a shared *.jar library, you can define the shared classloaders a web-app-default. Each web-app will then use the same *.jar files without needing to copy them into each .war file.

Example: web-app-default of website-jars

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="urn:java:com.caucho.resin">

<cluster id="">
```

```
<host-default>
  <web-app-default>

    <class-loader>
      <library-loader path="/var/resin/website-jars"/>
    </class-loader>

  </web-app-default>
</host-default>

<host id="">

  <web-app-deploy path="webapps"
            expand-preserve-fileset="WEB-INF/work/**"/>

</host>

</cluster>

</resin>
```

28.4 Servlets and Filters

Resin's <web-app> includes all the standard Servlet configuration. The WEB-INF/web.xml parsing uses the same Resin code as the <web-app> or resin-web.xml configuration. So all standard Servlet configuration is possible in a <web-app> or <web-app-default>.

In fact, the web.xml is just a convention handled by the app-default.xml file. It's defined as:

Definition: web.xml in app-default.xml

```
<web-app-default>

  <resin:import path="WEB-INF/web.xml" optional="true"/>

</web-app-default>
```

28.5 app-default.xml

Resin's standard servlet implementation is configured with a resin:import of the app-default.xml. app-default.xml defines the WEB-INF/web.xml, WEB-INF/classes, WEB-INF/lib, JSPs and the static servlet. In other words, if it's missing from the resin.xml, Resin will not behave as a standard servlet engine.

28.6 Resources: databases, queues, beans

Resources in a web-app are typically defined in the resin-web.xml file. The resin-web.xml has the same syntax as the web.xml. Using two files lets you split out the Resin-specific configuration into the resin-web.xml and keep the standard servlet configuration in the web.xml.

Common resources like <database> have their own configuration tags listed in the reference guide.. Other resources like third-party libraries and application beans can use the CDI-style XML configuration, which is general enough to configure non-CDI objects and store them in JNDI as well as configure CDI-style beans.

Example: WEB-INF/resin-web.xml database configuration

```
<web-app xmlns="http://caucho.com/ns/resin">

<database jndi-name='jdbc/test_mysql'>
  <driver type="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource">
    <url>jdbc:mysql://localhost:3306/test</url>
    <user></user>
    <password></password>
  </driver>
</database>

</web-app>
```

Third party libraries and application beans can be configured using the CDI XML configuration and saved to JNDI. The class name is given by the XML namespace and the tag name.

The following example creates a com.mycom.mylib.MyBean instance, configures a field, and saves it in JNDI at the name "java:comp/env/my-bean".

Example: WEB-INF/resin-web.xml JNDI configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

<mylib:MyBean xmlns:mylib="urn:java:com.mycom.mylib"
                 resin:Jndi="java:comp/env/my-bean">
    <myField>myValue</myField>
</mylib:MyBean>

</web-app>
```

28.7 URL rewriting

Resin's URL rewriting tags can be configured in the resin-web.xml to conditionally rewrite URLs from a "pretty" external view to a more practical internal one and redirect pages.

Example: dispatch for a wiki

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

<resin:Dispatch target="\.(php|gif|css|png|js)"/>
<resin:Dispatch>
  <resin:IfFileExists/>
</resin:Dispatch>

<resin:Dispatch regexp="^" target="/index.php"/>

</web-app>
```

Chapter 29

Web Server: HTTP Proxy Cache

29.1 Overview

For many applications, enabling the proxy cache can improve your application's performance dramatically. When Quercus runs Mediawiki with caching enabled, Resin can return results as fast as static pages. Without caching, the performance is significantly slower.

Table 29.1: Mediawiki Performance

CACHE PERFORMANCE	NON-CACHE PERFORMANCE
4316 requests/sec	29.80 requests/sec

To enable caching, your application will need to set a few HTTP headers. While a simple application can just set a timed cache with max-age , a more sophisticated application can generate page hash digests with ETag and short-circuit repeated If-None-Match responses.

If subsections of your pages are cacheable but the main page is not, you can cache servlet includes just like caching top-level pages. Resin's include processing will examine the headers set by your include servlet and treat them just like a top-level cache.

29.2 HTTP Caching Headers

Table 29.2: HTTP Server to Client Headers

HEADER	DESCRIPTION
Cache-Control: private	Restricts caching to the browser only, forbidding proxy-caching.
Cache-Control: max-age= n	Specifies a static cache time in seconds for both the browser and proxy cache.
Cache-Control: s-maxage= n	Specifies a static cache time in seconds for the proxy cache only.
Cache-Control: no-cache	Disables caching entirely.
ETag: hash or identifier	Unique identifier for the page's version. Hash-based values are better than date/versioning, especially in clustered configurations.
Last-Modified: time of modification	Accepted by Resin's cache, but not recommended in clustered configurations.

Table 29.2: (continued)

HEADER	DESCRIPTION
Vary: header-name	Caches the client's header, e.g. Cookie, or Accept-encoding

Table 29.3: HTTP Client to Server Headers

HEADER	DESCRIPTION
If-None-Match	Specifies the ETag value for the page
If-Modified-Since	Specifies the Last-Modified value for the page

29.2.1 Cache-Control: max-age

Setting the max-age header will cache the results for a specified time. For heavily loaded pages, even setting short expires times can significantly improve performance. Pages using sessions should set a "Vary: Cookie" header, so anonymous users will see the cached page, while logged-in users will see their private page.

Example: 15s cache

```
<%@ page session="false" %>
<%! int counter; %>
<%
response.addHeader("Cache-Control", "max-age=15");
%>
Count: <%= counter++ %>
```

max-age is useful for database generated pages which are continuously, but slowly updated. To cache with a fixed content, i.e. something which has a valid hash value like a file, you can use ETag with If-None-Match .

29.2.2 ETag and If-None-Match

The ETag header specifies a hash or digest code for the generated page to further improve caching. The browser or cache will send the ETag as a If-None-Match value when it checks for any page updates. If the page is the same, the application will return a 304 NOT_MODIFIED response with an empty body. Resin's FileServlet automatically provides this capability for static pages. In general, the ETag is the most effective caching technique, although it requires a bit more work than max-age .

To handle clustered servers in a load-balanced configuration, the calculated ETag should be a hash of the result value, not a timestamp or version. Since each server behind a load balancer will generate a different timestamp for the files, each server would produce a different tag, even though the generated content was identical. So either producing a hash or ensuring the ETag value is the same is critical.

ETag servlets will often also use <cache-mapping> configuration to set a max-age or s-maxage . The browser and proxy cache will cache the page without revalidation until max-age runs out. When the time expires, it will use If-None-Match to revalidate the page.

When using ETag , your application will need to look for the If-None-Match header on incoming requests. If the value is the same, your servlet can return 304 NOT-MODIFIED. If the value differs, you'll return the new content and hash.

Example: ETag servlet

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class MyServlet extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  {
    String etag = getCurrentEtag();

    String ifNoneMatch = req.getHeader("If-None-Match");

    if (ifNoneMatch != null && ifNoneMatch.equals(etag)) {
      res.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
      return;
    }

    res.setHeader("ETag", etag);

    ... // generate response
  }
}
```

Example: HTTP headers for ETag match

```
C: GET /test-servlet HTTP/1.1

S: HTTP/1.1 200 OK
S: ETag: xm8vz29I
S: Cache-Control: max-age=15s
S: ...

C: GET /test-servlet HTTP/1.1
C: If-None-Match: xm8vz29I

S: HTTP/1.1 304 Not Modified
S: Cache-Control: max-age=15s
S: ...
```

Example: HTTP headers for ETag mismatch

```
C: GET /test-servlet HTTP/1.1
C: If-None-Match: UXi456Ww

S: HTTP/1.1 200 OK
S: ETag: xM81x+3j
S: Cache-Control: max-age=15s
S: ...
```

29.2.3 Expires

Although max-age tends to be easier and more flexible, an application can also set the Expires header to enable caching, when the expiration date is a specific time instead of an interval. For heavily loaded pages, even setting short expires times can significantly improve performance. Sessions should be disabled for caching.

The following example sets expiration for 15 seconds. So the counter should update slowly.

Example: expires

```
<%@ page session="false" %>
<%! int counter; %>
<%
long now = System.currentTimeMillis();
```

```
response.setDateHeader("Expires", now + 15000);
%>
Count: <%= counter++ %>
```

Expires is useful for database generated pages which are continuously, but slowly updated. To cache with a fixed content, i.e. something which has a valid hash value like a file, you can use ETag with If-None-Match .

29.2.4 If-Modified-Since

The If-Modified-Since headers let you cache based on an underlying change date. For example, the page may only change when an underlying source page changes. Resin lets you easily use If-Modified by overriding methods in HttpServlet or in a JSP page.

Because of the clustering issues mentioned in the ETag section, it's generally recommended to use ETag and If-None-Match and avoid If-Modified-Since. In a load balanced environment, each backend server would generally have a different Last-Modified value, while would effectively disable caching for a proxy cache or a browser that switched from one backend server to another.

The following page only changes when the underlying *test.xml* page changes.

```
<%@ page session="false" %>
<%!
int counter;

public long getLastModified(HttpServletRequest req)
{
  String path = req.getRealPath("test.xml");
  return new File(path).lastModified();
}
%>
Count: <%= counter++ %>
```

If-Modified pages are useful in combination with the cache-mapping configuration.

29.2.5 Vary

In some cases, you'll want to have separate cached pages for the same URL depending on the capabilities of the browser. Using gzip compression is the most important example. Browsers which can understand gzip-compressed files receive the compressed page while simple browsers will see the uncompressed page. Using the "Vary" header, Resin can cache different versions of that page.

Example: vary caching for on gzip

```
<%
  response.addHeader("Cache-Control", "max-age=3600");
  response.addHeader("Vary", "Accept-Encoding");
%>

Accept-Encoding: <%= request.getHeader("Accept-Encoding") %>
```

The "Vary" header can be particularly useful for caching anonymous pages, i.e. using "Vary: Cookie". Logged-in users will get their custom pages, while anonymous users will see the cached page.

29.3 Included Pages

Resin can cache subpages even when the top page can't be cached. Sites allowing user personalization will often design pages with jsp:include subpages. Some subpages are user-specific and can't be cached. Others are common to everybody and can be cached.

Resin treats subpages as independent requests, so they can be cached independent of the top-level page. Try the following, use the first expires counter example as the included page. Create a top-level page that looks like:

Example: top-level non-cached page

```
<% if (! session.isNew()) { %>
<h1>Welcome back!</h1>
<% } %>

<jsp:include page="expires.jsp"/>
```

Example: cached include page

```
<%@ page session="false" %>
<%! int counter; %>
<%
response.setHeader("Cache-Control", "max-age=15");
%>
Count: <%= counter++ %>
```

29.4 Caching Anonymous Users

The Vary header can be used to implement anonymous user caching. If a user is not logged in, he will get a cached page. If he's logged in, he'll get his own page. This feature will not work if anonymous users are assigned cookies for tracking purposes.

To make anonymous caching work, you must set the Vary: Cookie If you omit the Vary header, Resin will use the max-age to cache the same page for every user.

Example: Vary: Cookie for anonymous users

```
<%@ page session="false" %>
<%! int \_counter; %>
<%
response.addHeader("Cache-Control", "max-age=15");
response.addHeader("Vary", "Cookie");

String user = request.getParameter("user");
%>
User: <%= user %> <%= counter++ %>
```

The top page must still set the max-age or If-Modified header, but Resin will take care of deciding if the page is cacheable or not. If the request has any cookies, Resin will not cache it and will not use the cached page. If it has no cookies, Resin will use the cached page.

When using Vary: Cookie, user tracking cookies will make the page uncacheable even if the page is the same for all users. Resin chooses to cache or not based on the existence of any cookies in the request, whether they're used or not.

29.5 Configuration

29.5.1 cache

<cache> configures the proxy cache (requires Resin Professional). The proxy cache improves performance by caching the output of servlets, jsp and php pages. For database-heavy pages, this caching can improve performance and reduce database load by several orders of magnitude.

The proxy cache uses a combination of a memory cache and a disk-based cache to save large amounts of data with little overhead.

Management of the proxy cache uses the ProxyCacheMXBean .

Table 29.4: <cache> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Path to the persistent cache files.	cache/
disk-size	Maximum size of the cache saved on disk.	1024M
enable	Enables the proxy cache.	true
enable-range	Enables support for the HTTP Range header.	true
entries	Maximum number of pages stored in the cache.	8192
max-entry-size	Largest page size allowed in the cache.	1M
memory-size	Maximum heap memory used to cache blocks.	8M
rewrite-vary-as-private	Rewrite Vary headers as Cache-Control: private to avoid browser and proxy-cache bugs (particularly IE).	false

```
element cache {
  disk-size?
  & enable?
  & enable-range?
  & entries?
  & path?
  & max-entry-size?
  & memory-size?
  & rewrite-vary-as-private?
}
```

Example: enabling proxy cache

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <cache entries="16384" disk-size="2G" memory-size="256M"/>

        <server id="a" address="192.168.0.10"/>

        <host host-name="www.foo.com">
    </cluster>
</resin>
```

29.5.2 rewrite-vary-as-private

Because not all browsers understand the Vary header, Resin can rewrite Vary to a Cache-Control: private. This rewriting will cache the page with the Vary in Resin's proxy cache, and also cache the page in the browser. Any other proxy caches, however, will not be able to cache the page.

The underlying issue is a limitation of browsers such as IE. When IE sees a Vary header it doesn't understand, it marks the page as uncacheable. Since IE only understands "Vary: User-Agent", this would mean IE would refuse to cache gzipped pages or "Vary: Cookie" pages.

With the <rewrite-vary-as-private> tag, IE will cache the page since it's rewritten as "Cache-Control: private" with no Vary at all. Resin will continue to cache the page as normal.

29.5.3 cache-mapping

cache-mapping assigns a max-age and Expires to a cacheable page, i.e. a page with an ETag or Last-Modified setting. It does not affect max-age or Expires cached pages. The FileServlet takes advantage of cache-mapping because it provides the ETag servlet.

Often, you want a long Expires time for a page to a browser. For example, any gif will not change for 24 hours. That keeps browsers from asking for the same gif every five seconds; that's especially important for tiny formatting gifs. However, as soon as that page or gif changes, you want the change immediately available to any new browser or to a browser using reload.

Here's how you would set the Expires to 24 hours for a gif, based on the default FileServlet.

Example: caching .gif files for 24h

```
<web-app xmlns="http://caucho.com/ns/resin">

  <cache-mapping url-pattern='*.gif'
                 expires='24h'/>
</web-app>
```

The cache-mapping automatically generates the Expires header. It only works for cacheable pages setting If-Modified or ETag. It will not affect pages explicily setting Expires or non-cacheable pages. So it's safe to create a cache-mapping for *.jsp even if only some are cacheable.

29.6 Debugging caching

When designing and testing your cached page, it's important to see how Resin is caching the page. To turn on logging for caching, you'll add the following to your resin.xml:

Example: adding caching log

```
<resin xmlns="http://caucho.com/ns/resin">

  <logger name="com.caucho.server.cache" level="fine"/>

  ...

</resin>
```

The output will look something like the following:

```
[10:18:11.369] caching: /images/caucho-white.jpg etag="AAAAPbkEyoA" length=6190
[10:18:11.377] caching: /images/logo.gif etag="AAAAOQ9zLeQ" length=571
[10:18:11.393] caching: /css/default.css etag="AAAANzMooDY" length=1665
[10:18:11.524] caching: /images/pixel.gif etag="AAAANpcE4pY" length=61

...

[10:18:49.303] using cache: /css/default.css
[10:18:49.346] using cache: /images/pixel.gif
[10:18:49.348] using cache: /images/caucho-white.jpg
[10:18:49.362] using cache: /images/logo.gif
```

29.7 Administration

29.7.1 /resin-admin

The /resin-admin URL provides an overview of the current state of Resin.

29.7.1.1 block cache miss ratio

The block cache miss ratio tells how often Resin needs to access the disk to read a cache entry. Most cache requests should come from memory to improve performance, but cache entries are paged out to disk when the cache gets larger. It's very important to keep the <memory-size> tag of the <cache> large enough so the block cache miss ratio is small.

29.7.1.2 proxy cache miss ratio

The proxy cache miss ratio measures how often cacheable pages must go to their underlying servlet instead of being cached. The miss ratio does not measure the non-cacheable pages.

29.7.1.3 invocation miss ratio

The invocation miss ratio measures how often Resin's invocation cache misses. The invocation cache is used for both cacheable and non-cacheable pages to save the servlet and filter chain construction. A miss of the invocation cache is expensive, since it will not only execute the servlet, but also force the servlet and filter chain to be rebuilt. The <entries> field of the <cache> controlls the invocation miss ratio.

29.7.2 BlockManagerMXBean

BlockManagerMXBean returns statistics about the block cache. Since Resin's block cache is used for the proxy cache as well as clustered sessions and JMS messages, the performance of the block cache is very important. The block cache is a memory/paging cache on top of a file-based backing store. If the block cache misses, a request needs to go to disk, otherwise it can be served directly from memory.

```
resin:type=BlockManager
```

```
public interface BlockManagerMXBean {
  public long getBlockCapacity();

  public long getHitCountTotal();
  public long getMissCountTotal();
}
```

29.7.3 ProxyCacheMXBean

The ProxyCacheMXBean provides statistics about the proxy cache as well as operations to clear the cache. The hit and miss counts tell how effectively the cache is improving performance.

```
resin:type=ProxyCache
```

```
public interface ProxyCacheMXBean {
  public long getHitCountTotal();
  public long getMissCountTotal();

  public CacheItem []getCacheableEntries(int max);
  public CacheItem []getUncacheableEntries(int max);

  public void clearCache();
  public void clearCacheByPattern(String hostRegexp, String urlRegexp);
  public void clearExpires();
}
```

Chapter 30

Advanced Topics

30.1 Classloaders

30.1.1 class-loader

Adds dynamic classloaders to the current environment.

Each environment (<server>, <host>, <web-app>) etc, can add dynamic classloaders. The environment will inherit the parent classloaders. Each <class-loader> is comprised of several implementing loader items: library-loader for WEB-INF/lib, compiling-loader for WEB-INF/classes, even make-loader for more complex auto-compilation.

<class-loader> in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <prologue>
    <class-loader>
      <compiling-loader path="WEB-INF/classes"/>

      <library-loader path="WEB-INF/lib"/>
    </class-loader>
  </prologue>
</web-app>
```

30.1.2 compiling-loader

Configures an auto-compiling WEB-INF/classes -style class loader.

The compiling-loader will automatically compile Java code into .class files loading them.

Table 30.1: compiling-loader Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
args	Additional arguments to be passed to the Java compiler. Resin 3.0	none
batch	If true, multiple changed *.java files will be compiled in a single batch. Resin 3.0.7	true
encoding	I18N encoding for the Java compiler. Since Resin 3.0	none
path	Filesystem path for the class loader. Since Resin 3.0	required

Table 30.1: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
source	Java source directory. Since Resin 3.0	value of path
require-source	If true, .class files without matching .java files will be deleted. Since Resin 3.0	false

30.1.3 library-loader

Configures a jar library, WEB-INF/lib -style class loader.

The library-loader will add jar files in its path to the current classpath. Jar files are recognized when they have a filename extension of

`.jar` or `.zip` .

Table 30.2: library-loader Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required

See com.caucho.loader.DirectoryLoader .

30.1.4 tree-loader

Configures a jar library, WEB-INF/lib -style class loader similar to library-loader , but will also find `.jar` and `.zip` files in subdirectories.

Table 30.3: tree-loader Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required

See com.caucho.loader.TreeLoader .

30.1.5 make-loader

Configures a custom make-style loader.

30.1.6 servlet-hack

Use of servlet-hack is discouraged. Using servlet-hack violates the JDK's classloader delegation model and can produce surprising ClassCastExceptions.

servlet-hack reverses the normal class loader order. Instead of parent classloaders having priority, child classloaders have priority.

30.1.7 simple-loader

Configures a simple classes -style class loader.

class files in the specified directory will be loaded without any special compilation steps (in contrast with compiling-loader.)

Table 30.4: simple-loader Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required
prefix	Class package prefix to only load to a subset of classes. Resin 3.0	none

30.2 JDK 5.0 and JMX

JDK 5.0 includes a JMX implementation that is used to provide local and remote administration of a Resin server.

Start Resin and allow local JMX administration

```
win> ./resin.exe -Dcom.sun.management.jmxremote
unix> bin/resin.sh -Dcom.sun.management.jmxremote
```

Start jconsole

```
win> jconsole.exe
unix> jconsole

Choose Resin's JVM from the "Local" list.
```

Start Resin and allow remote JMX administration

```
win> ./resin.exe -Dcom.sun.management.jmxremote.port=9999
unix> bin/resin.sh -Dcom.sun.management.jmxremote.port=9999
```

Without some configuration effort, the previous command will not work. Password configuration and SSL configuration is required by the JDK implementation of remote JMX. Detailed instructions are included in the JDK documentation.

The following is useful for testing, but should be done with caution as the port is not protected by password or by SSL, and if not protected by a firewall is accessible by anyone who can guess the port number.

Start Resin and remote JMX - disable password checking and SSL

```
win> ./resin.exe -Dcom.sun.management.jmxremote.port=9999
                    -Dcom.sun.management.jmxremote.ssl=false
                    -Dcom.sun.management.jmxremote.authenticate=false

unix> bin/resin.sh -Dcom.sun.management.jmxremote.port=9999 \
                      -Dcom.sun.management.jmxremote.ssl=false \
                      -Dcom.sun.management.jmxremote.authenticate=false
```

Start jconsole

```
win> jconsole.exe
unix> jconsole

Enter the host name and port number (9999) on the "Remote" tab
```

Setting a password for remote JMX access

```
$ cd $JAVA_HOME/jre/lib/management
$ cp jmxremote.password.template jmxremote.password
$ chmod u=rw jmxremote.password
$ vi jmxremote.password

Set a password for "monitorRole" and "controlRole":

monitorRole 12monitor
controlRole 55control
```

Start Resin and remote JMX - disable SSL

```
win> ./resin.exe -Dcom.sun.management.jmxremote.port=9999
                    -Dcom.sun.management.jmxremote.ssl=false

unix> bin/resin.sh -Dcom.sun.management.jmxremote.port=9999 \
                      -Dcom.sun.management.jmxremote.ssl=false
```

Start jconsole

```
win> jconsole.exe
unix> jconsole
```

Enter the host name and port number (9999) on the "Remote" tab Enter the username and password on the "Remote" tab

30.3 Instrumenting Resources

Instrumenting resources so JMX can manage them consists of the following steps: For a class `MyFoo` , create an interface `MyFooMBean` with the management interface. Class `MyFoo` needs to implement the `MyFooMBean` interface. Register `MyFoo` with the JMX server.

30.3.1 Instrumenting a servlet

Resin will automatically register any servlet which implement an MBean interface. By default, the JMX name will be:

```
web-app:j2eeType=Servlet,name= servlet-name
```

Table 30.5: ObjectName attributes

ATTRIBUTE	VALUE
j2eeType	Servlet
WebModule	the contextPath
J2EEApplication	the host?
J2EEServer	the server-id?

The domain is web-app , the type property is javax.servlet.Servlet and the name property is the value of <servlet-name>.

JMX clients will use the name to manage the servlet. For example, a client might use the pattern web-app:type=javax.servlet.Servlet,* to retrieve all managed servlets.

MyServletMBean.java

```
package test;

public interface MyServletMBean {
  public int getCount();
}
```

MyServlet.java

```
package test;

import java.io.*;
import javax.servlet.*;

public class MyServlet extends GenericServlet implements MyServletMBean {
  private int count;

  public int getCount()
  {
    return count;
  }

  public void service(ServletRequest request,
                      ServletResponse response)
    throws IOException
  {
    PrintWriter out = response.getWriter();

    count++;

    out.println("Hello, world");
  }
}
```

30.4 Managing Resources

Managing resources uses the JMX API, primarily using the `MBeanServer` object. In Resin, each web-app has its own `MBeanServer` .

Getting the Count attribute

```
import javax.management.*;

...

MBeanServer server = MBeanServerFactory.createMBeanServer();

ObjectName name = new ObjectName("web-app:j2eeType=javax.servlet.Servlet," +
                                 "name=hello");

Object value = server.getAttribute(name, "Count");

out.println("Count: " + value);
```

30.5 Interpreting the proxy cache hit ratio

The proxy cache is Resin's internal proxy cache (in Resin Pro). The hit ratio marks what percentage of requests are served out of the cache, i.e. quickly, and which percentage are taking the full time.

The proxy cache hit ratio is useful for seeing if you can improve your application's performance with better caching. For example, if you had a news site like www.cnn.com, you should have a high hit rate to make sure you're not overtaxing the database.

If you have a low value, you might want to look at your heavily used pages to see if you can cache more.

Chapter 31

Reference

31.1 AbstractAuthenticator

While this case is rare, it may sometimes be useful to create your own Resin custom authenticator (for example to use a legacy resource as an authentication store). The Resin security framework provides an abtract base class (com.caucho.security.AbstractAuthentica) that you can extend to do this.

The following is a simple example that you can use a starting point for your application:

WEB-INF/resin-web.xml - Custom Authenticator Configuration

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:foo="urn:java:com.caucho.foo">
  ...
  <foo:MyAuthenticator>
    <foo:foo>bar</foo:foo>
  </foo:MyAuthenticator>
  ...
</web-app>
```

MyAuthenticator.java

```
package com.foo;

import com.caucho.security.AbstractAuthenticator;
import com.caucho.security.PasswordUser;

public class MyAuthenticator extends AbstractAuthenticator {
  private PasswordUser \_user;

  public MyAuthenticator()
  {
    \_user = new PasswordUser("harry", "quidditch",
                              new String[] { "user" });
  }

  public PasswordUser getUser(String userName)
  {
    if (userName.equals(_user.getName()))
      return \_user;
    else
      return null;
  }
}
```

31.2 <accept-listen-backlog>

<accept-listen-backlog> configures operating system TCP listen queue size for the port.

When a browser connects to a server, the server's operating system handles the TCP initialization before handing the socket to the server's application. The operating system will hold the opened connections in a small queue, until the application is ready to receive them. When the queue fills up, the operating system will start refusing new connections.

31.3 <accept-thread-max>

<accept-thread-max> configures the maximum number of threads listening for new connections on this port. <accept-thread-max> works with <accept-thread-min> to handle spiky loads without creating and destroying too many threads.

Socket connections are associated with a thread which handles the request. In Resin, a number of threads wait to accept a new connection and then handle the request. <accept-thread-max> specifies the maximum number of threads which are waiting for a new connection.

Larger values handle spiky loads better but require more threads to wait for the connections. Smaller values use less threads, but may be slower handling spikes.

31.4 <accept-thread-min>

<accept-thread-min> configures the minimum number of threads listening for new connections on this port <accept-thread-min> works with <accept-thread-max> to handle spiky loads without creating and destroying too many threads.

Socket connections are associated with a thread which handles the request. In Resin, a number of threads wait to accept a new connection and then handle the request. <accept-thread-min> specifies the minimum number of threads which are waiting for a new connection. If many connections appear rapidly with a small value of <accept-thread-min>, the application may pause until a new thread is available for the new connection.

Larger values handle spiky loads better but require more threads to wait for the connections. Smaller values use less threads, but may be slower handling spikes.

31.5 <access-log>

<access-log> configures the access log file.

As a child of web-app , overrides the definition in the host that the web-app is deployed in. As a child of host , overrides the definition in the server that the host is in.

The default archive format is path + ".%Y%m%d" or path + ".%Y%m%d.%H" if rollover-period < 1 day.

The access log formatting variables follow the Apache variables:

Table 31.1: format patterns

PATTERN	DESCRIPTION
%b	result content length
%D	time taken to complete the request in microseconds (since 3.0.16)
%h	remote IP addr
%{ xxx }i	request header xxx
%{ xxx }o	response header xxx
%{ xxx }c	cookie value xxx
%n	request attribute

Table 31.1: (continued)

PATTERN	DESCRIPTION
%r	request URL
%s	status code
%S	requested session id
%{ xxx }t	request date with optional time format string.
%T	time taken to complete the request in seconds
%u	remote user
%U	request URI
%v	name of the virtual host serving the request

The default format is:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
```

resin:type allows for custom logging. Applications can extend a custom class from com.caucho.http.log.AccessLog . admin/config-candi.xtp can be used to set bean parameters in the custom class.

Table 31.2: <access-log> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Output path for the log entries, see admin/logging.xtp#path .	required
path-format	Selects a format for generating path names. The syntax is the same as for archive-format.	optional
archive-format	the format for the archive filename when a rollover occurs, see Rollovers .	see below
auto-flush	true to flush the memory buffer with each request.	false
auto-flush-time	sets time interval for flushing the memory buffers	60s
exclude	access logging exclude patterns for request URIs. Access to matching URIs does not get logged.	none
format	Access log format.	see above
hostname-dns-lookup	log the dns name instead of the IP address (has a performance hit).	false
rollover-period	how often to rollover the log. Specify in days (15D), weeks (2W), months (1M), or hours (1h). See admin/logging.xtp#rollover .	none
rollover-size	maximum size of the file before a rollover occurs, in bytes (50000), kb (128kb), or megabytes (10mb). See admin/logging.xtp#rollover .	1mb
rollover-count	maximum number of rollover files before the oldest ones get overwritten. See admin/logging.xtp#rollover .	1mb
resin:type	a class extending com.caucho.server.log.AccessLog for custom logging	com.caucho.server.log.AccessLog
init	Resin-IoC initialization for the custom class	n/a

```
element access-log {
  path?
  & path-format?
  & archive-format?
  $amp;auto-flush?
  & auto-flush-time?
  & exclude*
  & format?
  & hostname-dns-lookup?
  & rollover-period?
  & rollover-size?
  & rollover-count?
  & resin:type?
  & init?
}
```

Example: <access-log> in host configuration

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="app-tier">

  <host id="">
    <access-log path='log/access.log'>
      <rollover-period>2W</rollover-period>
    </access-log>
  </host>
</cluster>
</resin>
```

Example: custom access log

```
<resin xmlns="http://caucho.com/ns/resin">

<cluster id="app-tier">

  <host id='foo.com'>

    <access-log>
      <test:MyLog xmlns:test="urn:java:test">
                  path='${resin.root}/foo/error.log'
                  rollover-period='1W'>
          <test:foo>bar</test:foo>
      </test:MyLog>
    </access-log>
    ...
  </host>

</cluster>
</resin>
```

31.6 <active-wait-time>

<active-wait-time> sets a 503 busy timeout for requests trying to access a restarting web-app. If the timeout expires before the web-app complete initialization, the request will return a 503 Busy HTTP response.

```
element active-wait-time {
  r_period-Type
}
```

31.7 <address>

The server <address> defines the IP interface for Resin cluster communication and load balancing. It will be an internal IP address like 192.168.* for a clustered configuration or 127.* for a single-server configuration. No wild cards are allowed because the other cluster servers and load balancer use the address to connect to the server.

server address

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <http port="80"/>
    </server-default>

    <server id="web-a" address="192.168.1.1" port="6800"/>
    <server id="web-b" address="192.168.1.2" port="6800"/>

    ...
  </cluster>

  <cluster id="app-tier">
    <server id="app-a" address="192.168.2.11" port="6800"/>
    <server id="app-b" address="192.168.2.12" port="6800"/>

    ...
  </cluster>
</resin>
```

31.8 <allow-forward-after-flush>

The <allow-forward-after-flush> flag configures whether IllegalStateExcdeption is thrown when using forward() method after response has been committed. Flag configures behavior of servlet and jsp.

```
element allow-forward-after-flush {
  r_boolean-Type
}
```

31.9 <allow-servlet-el>

The <allow-servlet-el> flag configures whether <servlet> tags allow EL expressions in the init-param.

```
element allow-servlet-el {
  r_boolean-Type
}
```

31.10 <archive-path>

<archive-path> configures the location of the web-app's .war file. In some configurations, the .war expansion might not use the webapps/ directory, but will still want automatic war expantion.

```
element archive-path {
  r_path-Type
}
```

Example: resin.xml explicit archive location

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <host id="">

    <web-app id="/foo"
             root-directory="/var/resin/foo"
             archive-path="/var/resin/wars/foo.war"/>

  </host>
</cluster>
</resin>
```

31.11 <auth-constraint>

Requires that authenticated users fill the specified role. In Resin's JdbcAuthenticator, normal users are in the "user" role. Think of a role as a group of users.

The roles are defined by the authenticators (see admin/security-overview.xtp). When using Resin's <resin:AdminAuthenticator> as a default, the role name is resin-admin . (See admin/resin-admin.xtp .)

Table 31.3: <auth-constraint> Attributes

ATTRIBUTE	DESCRIPTION
role-name	Roles which are allowed to access the resource.

```
element auth-constraint {
  description*,

  role-name*
}
```

31.12 <authenticator>

The <authenticator> tag as been replaced by CDI-style authenticator configuration. It exists only for backward compatibility. To upgrade, see the authenticator-specific tags like #resin:XmlAuthenticator .

The new name will be the old class="..." name. So class="com.caucho.security.XmlAuthenticator" becomes <resin:XmlAuthenticator>.

XmlAuthenticator in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:XmlAuthenticator password-digest="none">
    <resin:user name="Harry Potter" password="quidditch" group="user,gryffindor"/>
    <resin:user name="Draco Malfoy" password="pureblood" group="user,slytherin"/>
  </resin:XmlAuthenticator>
  ...
</web-app>
```

31.13 <bean>

The <bean> tag as been replaced by CDI-style configuration. It exists only for backward compatibility.

The new tag name will be the old class="..." name and the XML prefix will be the package. So class="com.mycom.FooBean" becomes <mycom:FooBean xmlns:mycom="urn:com.mycom">.

MyBean in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin"
            xmlns:mypkg="urn:java:com.mycom.mypkg">
  ...
  <mypkg:MyBean resin:JndiName="java:comp/env/my-name">
    <my-attribute>my-value</my-attribute>
  </mypkg:MyBean>
  ...
</web-app>
```

31.14 <cache>

<cache> configures the proxy cache (requires Resin Professional). The proxy cache improves performance by caching the output of servlets, jsp and php pages. For database-heavy pages, this caching can improve performance and reduce database load by several orders of magnitude.

The proxy cache uses a combination of a memory cache and a disk-based cache to save large amounts of data with little overhead.

Management of the proxy cache uses the ProxyCacheMXBean .

Table 31.4: <cache> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Path to the persistent cache files.	cache/
disk-size	Maximum size of the cache saved on disk.	1024M
enable	Enables the proxy cache.	true
enable-range	Enables support for the HTTP Range header.	true
entries	Maximum number of pages stored in the cache.	8192
max-entry-size	Largest page size allowed in the cache.	1M
memory-size	Maximum heap memory used to cache blocks.	8M
rewrite-vary-as-private	Rewrite Vary headers as Cache-Control: private to avoid browser and proxy-cache bugs (particularly IE).	false

```
element cache {
  disk-size?
  & enable?
  & enable-range?
  & entries?
  & path?
```

```
  & max-entry-size?
  & memory-size?
  & rewrite-vary-as-private?
}
```

Example: enabling proxy cache

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <cache entries="16384" disk-size="2G" memory-size="256M"/>

        <server id="a" address="192.168.0.10"/>

        <host host-name="www.foo.com">
    </cluster>
</resin>
```

31.15 <cache-mapping>

<cache-mapping> specifies max-age and Expires times for cacheable pages.

See admin/http-proxy-cache.xtp for more information.

<cache-mapping> is intended to provide Expires times for pages that have ETags or Last-Modified specified, but do not wish to hard-code the max-age timeout in the servlet. For example, Resin's FileServlet relies on cache-mapping to set the expires times for static pages. Using cache-mapping lets cacheable pages be configured in a standard manner.

<cache-mapping> does not automatically make pages cacheable. Your servlets must already set the ETag (or Last-Modified) header to activate <cache-mapping>. cache-mapping requires an enabled <cache>. If the cache is disabled, cache-mapping will be ignored. cache-mapping does not automatically make a page cacheable. Only cacheable pages are affected by cache-mapping, i.e. pages with an ETag or Last-Modified.

The time intervals default to seconds, but will allow other time intervals .

Table 31.5: <cache-mapping> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
expires	A time interval to be used for the HTTP Expires header.	
max-age	A time interval to be used for the "Cache-Control max-age=xxx" header. max-age affects proxies and browsers.	
s-max-age	A time interval to be used for the "Cache-Control s-max-age=xxx" header. s-max-age affects proxy caches (including Resin), but not browsers.	
url-pattern	A pattern matching the url: /foo/* , /foo , or *.foo	
url-regexp	A regular expression matching the url	

```
element cache-mapping {
  (url-pattern | url-regexp)
  & expires?
```

```
  & max-age?
  & s-max-age?
}
```

Example: cache-mapping in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

  <cache-mapping url-pattern='/*'
                 max-age='10'/>

  <cache-mapping url-pattern='*.gif'
                 max-age='15m'/>

</web-app>
```

31.16 <case-insensitive>

<case-insensitive> specifies whether the environment context is case sensitive or insensitive.

Because some operating systems are case-insensitive, it is important for security reasons for Resin to behave differently for case-sensitive and case-insensitive directories. For example, when case-insensitive is true, url-patterns will match in a case-insensitive manner, so TEST.JSP will work like test.jsp.

```
r_case-insensitive = element case-insensitive {
  r_boolean-Type
}
```

31.17 <character-encoding>

<character-encoding> specifies the default character encoding for the environment.

```
r_character-encoding = element character-encoding {
  string
}
```

Example: utf-8 as default character encoding

```
<resin xmlns="http://caucho.com/ns/resin">
  <character-encoding>utf-8</character-encoding>
  ...

</resin>
```

31.18 <class-loader>

<class-loader> configures a dynamic classloader for the current environment.

Each environment (<cluster>, <host>, <web-app>) etc, can add dynamic classloaders. The environment will inherit the parent classloaders. Each <class-loader> is comprised of several implementing loader items: library-loader for WEB-INF/lib, compiling-loader for WEB-INF/classes.

For web-apps, the classloaders generally belong in a <prologue> section, which ensures that Resin evaluates them first. The evaluation order is particularly important in cases like resin-web.xml vs web.xml, because the resin-web.xml is evaluated after the web.xml.

Table 31.6: <class-loader> Attributes

ELEMENT	DESCRIPTION
<compiling-loader>	Automatically compiles sources code to classes. It its the default loader for WEB-INF/classes.
<library-loader>	Loads jar files from a directory. It is the default loader for WEB-INF/lib.
<simple-loader>	Loads classes from a directory, but does not compile them automatically.
<tree-loader>	Loads jar files from a directory, recursively searching subdirectories.

```
r_class-loader = element class-loader {
  r_compiling-loader*

  & r_library-loader*

  & r_simple-loader*

  & r_tree-loader*
}
```

Example: WEB-INF/resin-web.xml defined <class-loader>

```
<web-app xmlns="http://caucho.com/ns/resin">
  <prologue>
    <class-loader>
      <compiling-loader path="WEB-INF/classes"/>

      <library-loader path="WEB-INF/lib"/>
    </class-loader>
  </prologue>
</web-app>
```

31.19 <close-dangling-connections>

<close-dangling-connections> closes open connections at the end of a request and logs a warning and stack trace.

31.20 <cluster>

<cluster> configures a set of identically-configured servers. The cluster typically configures a set of <server>s, each with some ports, and a set of virtual <host>s.

Only one <cluster> is active in any on server. At runtime, the <cluster> is selected by the <server> with id matching the -server on the command line.

Table 31.7: <cluster> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
id	The cluster identifier.	required
access-log	An access-log shared for all virtual hosts.	

Table 31.7: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
cache	Proxy cache for HTTP-cacheable results.	
connection-error-page	IIS error page to use when isapi_srun to Resin connection fails	
ear-default	default values for deployed ear files	
error-page	Custom error-page when virtual-hosts fail to match	
host	Configures a virtual host	
host-default	Configures defaults to apply to all virtual hosts	
host-deploy	Automatic host deployment based on a deployment directory	
ignore-client-disconnect	Ignores socket exceptions thrown because browser clients have prematurely disconnected	false
invocation-cache-size	Size of the system-wide URL to servlet invocation mapping cache	16384
invocation-cache-max-url-length	Maximum URL length saved in the invocation cache	256
max-uri-length	Maximum URI length allowed in request	1024
machine	Configuration for grouping <server> onto physical machines	
ping	Periodic checking of server URLs to verify server activity	
redeploy-mode	"automatic" or "manual"	automatic
resin:choose	Conditional configuration based on EL expressions	
resin:import	Imports a custom cluster.xml files for a configuration management	
resin:if	Conditional configuration based on EL expressions	
rewrite-dispatch	rewrites and dispatches URLs using regular expressions, similar to mod_rewrite	
root-directory	The root filesystem directory for the cluster	${resin.root}
server	Configures JVM instances (servers). Each cluster needs at least one server	
server-default	Configures defaults for all server instances	
server-header	Configures the HTTP "Server: Resin/xxx" header	Resin/Version
session-cookie	Configures the servlet cookie name	JSESSIONID
session-sticky-disable	Disables sticky-sessions on the load balancer	false
url-character-encoding	Configures the character encoding for URLs	utf-8
url-length-max	Configures the maximum length of an allowed URL	8192
web-app-default	Configures defaults to apply to all web-apps in the cluster	

```
element cluster {
  attribute id { string }
  & admin/config-overview.xtp
  & access-log?
  & cache?
  & connection-error-page?
  & ear-default*
  & error-page*
  & host*
  & host-default*
  & host-deploy*
  & ignore-client-disconnect?
  & invocation-cache-size?
  & invocation-cache-max-url-length?
  & machine*
  & ping*
  & redeploy-mode?
  & resin:choose*
  & resin:import*
  & resin:if*
  & rewrite-dispatch?
  & root-directory?
  & server*
  & server-default*
  & server-header?
  & session-cookie?
  & session-sticky-disable?
  & url-character-encoding?
  & url-length-max?
  & web-app-default*
}
```

Example: cluster-default

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <server-default>
            <http port="8080"/>
        </server-default>

        <server id="a" address="192.168.0.10"/>
        <server id="b" address="192.168.0.11"/>

        <host host-name="www.foo.com">
          ...
        </host>
    </cluster>
</resin>
```

31.21 <cluster-default>

<cluster-default> defines default cluster configuration for all clusters in the <resin> server.

Example: cluster-default

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster-default>
        <cache entries="16384" memory-size="64M"/>
    </cluster-default>
```

```
    <cluster id="web-tier">
        ...
    </cluster>

    <cluster id="app-tier">
        ...
    </cluster>
</resin>
```

31.22 <cluster-port>

<cluster-port> configures the cluster and load balancing socket, for load balancing, distributed sessions, and distributed management.

When configuring Resin in a load-balanced cluster, each Resin instance will have its own <server> configuration, which Resin uses for distributed session management and for the load balancing itself.

When configuring multiple JVMs, each <server> has a unique <server-id> which allows the -server command-line to select which ports the server should listen to.

Table 31.8: <cluster-port> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
address	hostname of the interface to listen to	*
jsse-ssl	configures the port to use JSSE for SSL	none
openssl	configures the port to use OpenSSL	none
port	port to listen to	required
read-timeout	timeout waiting to read from idle client	65s
write-timeout	timeout waiting to write to idle client	65s
accept-listen-backlog	The socket factory's listen backlog for receiving sockets	4000
tcp-no-delay	sets the NO_DELAY socket parameter	true

31.23 <compiling-loader>

<compiling-loader> automatically compiles Java code into .class files before loading them.

Table 31.9: <compiling-loader> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
args	Additional arguments to be passed to the Java compiler. Resin 3.0	
batch	If true, multiple changed *.java files will be compiled in a single batch. Resin 3.0.7	true
encoding	I18N encoding for the Java compiler. Since Resin 3.0	

Table 31.9: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required
source	Java source directory. Since Resin 3.0	value of path
require-source	If true, .class files without matching .java files will be deleted. Since Resin 3.0	false

Example: WEB-INF/resin-web.xml <compiling-loader>

```
<web-app xmlns="http://caucho.com/ns/resin">
  <prologue>
    <class-loader>
      <compiling-loader path="WEB-INF/classes"
                        source="WEB-INF/src"/>
    </class-loader>
  </prologue>
</web-app>
```

31.24 <connection>

Initialize all http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html objects managed by database .

Table 31.10: <connection> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
catalog	Sets catalog property on `java.sql.Connection` object.	
read-only	Sets readOnly property on `java.sql.Connection` object.	false
transaction-isolation	Sets transaction isolation property on `java.sql.Connection` object. All levels defined in `java.sql.Connection.TRANSACTION_*` are supported via value set: none, read-committed, read-uncommitted, repeatable-read, serializable.	unspecified

```
element connection {
  catalog?
  & read-only?
  & transaction-isolation?
}
```

Setting values for Connections in a pool

```
<web-app xmlns="http://caucho.com/ns/resin">
```

```
  <database>
    <driver type="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource">
      <url>jdbc:mysql://localhost:3306/test</url>
      <user></user>
      <password></password>
    </driver>

    <connection>
      <transaction-isolation>serializable</transaction-isolation>
      <catalog>my-catalog</catalog>
    </connection>
  </database>
</web-app>
```

31.25 <connection-error-page>

<connection-error-page> specifies an error page to be used by IIS when it can't contact an app-tier Resin. This directive only applies to IIS.

```
element connection-error-page {
  string
}
```

31.26 <connection-wait-time>

<connection-wait-time> configures the time a `getConnection`

call should wait when then pool is full before trying to create an overflow connection.

31.27 <constraint>

Defines a custom constraint. Applications can define their own security constraints to handle custom authentication requirements.

Table 31.11: <constraint> Attributes

ATTRIBUTE	DESCRIPTION
resin:type	A class that extends AbstractConstraint
init	initialization parameters, set in the object using Bean-style setters and getters

```
element constraint {
  class
  & init?
}
```

31.28 <context-param>

Initializes application (ServletContext) variables. context-param defines initial values for `application.getInitParameter("fo`
. See also ServletContext.getInitParameter() .

Table 31.12: <context-param> Attributes

ATTRIBUTE	DESCRIPTION
param-name	Named parameter
param-value	Parameter value
foo	Parameter name

```
element context-param {
  (param-name, param-value)*
  | (attribute * { string })*
  | (element * { string })*
}
```

Example: context-param in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

  <context-param>
    <param-name>baz</param-name>
    <param-value>value</param-value>
  </context-param>

  <!-- shortcut -->
  <context-param foo="bar"/>

</web-app>
```

31.29 <cookie-http-only>

The <cookie-http-only> flag configures the Http-Only attribute for all Cookies generated from the web-app. The Http-Only attribute can add security to a website by not forwarding HTTP cookies to SSL HTTPS requests.

```
element cookie-http-only {
  r_boolean-Type
}
```

Example: <cookie-http-only> in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">

  <host id="www.foo.com">
    <web-app id="" root-directory="/var/resin/foo">
      <cookie-http-only>true</cookie-http-only>
    <web-app id="">
  </host>

  <host id="www.foo.com:443">
    <web-app id="" root-directory="/var/resin/foo-secure">
```

```
      <secure/>
    <web-app id="">
  </host>

</cluster>
</resin>
```

31.30 cron trigger syntax

A time syntax used in a number of tags. Originally used in the Unix cron program. The http://en.wikipedia.org/wiki/Crontab contains this descriptive chart:

```
# +---------------- minute (0 - 59)
# |  +------------- hour (0 - 23)
# |  |  +---------- day of month (1 - 31)
# |  |  |  +------- month (1 - 12)
# |  |  |  |  +---- day of week (0 - 6) (Sunday=0 or 7)
# |  |  |  |  |
  *  *  *  *  *
```

Table 31.13: cron patterns

PATTERN	DESCRIPTION
*	matches all time periods
15	matches the specific time, e.g. 15 for minutes
15,45	matches a list of times, e.g. every :15 and :45
*/5	matches every n times, e.g. every 5 minutes
1-5	matches a range of times, e.g. mon, tue, wed, thu, fri (1-5)

31.31 <development-mode-error-page>

<development-mode-error-page> enables browser error reporting with extra information. Because it can expose internal data, it is not generally recommended in production systems. (The information is generally copied to the log.

31.32 <database>

<database> defines a database (i.e. DataSource) resource.

The admin/database.xtp section has more details on the configuration.

Table 31.14: <database> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
backup-driver	Configures a backup database driver. If Resin can't connect to any of the main drivers, it will use one of the. backups	

Table 31.14: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
close-dangling-connections	If an application does not close a Connection by the end of the request, Resin will close it automatically an issue a warning.	true
connection	Defines initialization attributes for new connections, e.g. setting the transaction-isolation.	true
connection-wait-time	When max-connections has been reached, how long Resin will wait for a connection to become idle before giving up.	10min
driver	Configures the database driver, giving the driver's class name as well as its JDBC URL and any other configuration.	required
jndi-name	The JNDI name to register the connection's `DataSource` under. If the name can be relative to java:comp/env .	
max-active-time	The maximum time Resin will allow a connection to remain open before forcing a close.	6 hours
max-close-statements	The maximum number of Statements Resin will hold to automatically close when the Connection closes.	256
max-connections	The maximum number of Connections allowed.	128
max-create-connections	The maximum number of connection creation allowed at one time.	5
max-idle-count	The maximum number of Connections in the idle pool.	1024
max-idle-time	The maximum time a connection will spend in the idle pool before closing.	30s
max-overflow-connections	The number of extra connection creation if the number of connections exceeds to pool size.	0
max-pool-time	The total time a connection can be used before it is automatically closed instead of returned to the idle pool.	24h
name	The IoC name to save the `ConnectionFactory` as, used with `@Named` to inject the resource.	
password	The JDBC password for the connection.	
ping	If true, Resin will ping the database before returning a connection from the pool (if ping-interval is exceeded).	false
ping-interval	How often an idle connection should ping the database to ensure it is still valid.	1s
ping-query	A custom query used to ping the database connection.	
ping-table	A table used to ping the database connection.	

Table 31.14: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
prepared-statement-cache-size	How many `PreparedStatements` to save in the prepared statement cache.	0
save-allocation-stack-trace	If true, saves the location of the connection allocation as a stack trace.	false
spy	Enables spy logging of database statements. The logging occurs with name="com.caucho.sql" and level="fine".	false
transaction-timeout	Sets the transaction timeout.	none
user	Sets the authentication user.	
wrap-statements	If true, Resin wraps statements and automatically closes them on connection close.	true
xa	Enables automatic enlistment of `Connections` with any `UserTransaction` . Disabling <xa> means the connection are independent of transactions, useful for read-only connections.	true
xa-forbid-same-rm	Workaround flag to handle certain database drivers that do not properly implement the XAResource API.	false

```
database = element database {
  backup-driver*
  & close-dangling-connections?
  & connection?
  & connection-wait-time?
  & driver+
  & jndi-name?
  & max-active-time?
  & max-close-statements?
  & max-connections?
  & max-create-connections?
  & max-idle-count?
  & max-idle-time?
  & max-overflow-connections?
  & max-pool-time?
  & name?
  & password?
  & ping?
  & ping-interval?
  & ping-query?
  & ping-table?
  & prepared-statement-cache-size?
  & save-allocation-stack-trace?
  & spy?
  & transaction-timeout?
  & user?
  & wrap-statements?
  & xa?
  & xa-forbid-same-rm?
}
```

```
backup-driver = element backup-driver {
  class?
  & url?
  & element * { * }?
}

connection = element connection {
  catalog?
  & read-only?
  & transaction-isolation?
}

driver = element driver {
  class?
  & url?
  & element * { * }?
}
```

Example: WEB-INF/resin-web.xml database

```
<web-app xmlns="http://caucho.com/ns/resin">

<database jndi-name='jdbc/test_mysql'>
  <driver class="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource">
    <url>jdbc:mysql://localhost:3306/test</url>
    <user></user>
    <password></password>
  </driver>
</database>

</web-app>
```

31.33 <database-default>

<database-default> defines default database values to be used for any <database> definition, or runtime database creation (see DatabaseManager).

```
element database-default {
  r_database-Content
}
```

Example: WEB-INF/resin-web.xml idle-time defaults

```
<web-app xmlns="http://caucho.com/ns/resin">

  <database-default>
    <max-idle-time>10s</max-idle-time>
  </database-default>

</web-app>
```

31.34 <dependency>

<dependency> adds dependent files which should force a reload when changed, like web.xml and resin-web.xml.

Table 31.15: <dependency> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path to the dependent file. Since Resin 3.0	required

```
element dependency {
  string
}
```

Example: struts dependency

```
<web-app xmlns="http://caucho.com/ns/resin">
  <dependency path="WEB-INF/struts-config.xml"/>
  ...
</web-app>
```

31.35 <dependency-check-interval>

<dependency-check-interval> Configures how often the environment context should be checked for changes. The default value is set low for development purposes, deployments should use something larger like 5m or 1h.

Resin automatically checks each environment for updates, generally class or configuration updates. Because these checks can take a considerable amount of time, deployment servers should use high values like 60s or more while development machines will want low values like 2s.

The interval defaults to the parent's interval. So the web-app will default to the host's value.

```
element dependency-check-interval {
  string
}
```

Example: deployment dependency-check-interval

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <dependency-check-interval>1h<dependency-check-interval>

    <server id="app-a" .../>

    <host id=""/>
      ...
  </cluster>
</resin>
```

31.36 <driver>

<driver> configures a database driver for a connection pool. The individual driver information is available from the driver vendor or on the http://wiki.caucho.com/Main_Page .

The content of the driver tag configures bean properties of the driver class, e.g. url, user, password.

```
element driver {
  type,
  *
}
```

31.37 <ear-default>

<ear-default> configures defaults for .ear resource, i.e. enterprise applications.

31.38 <ear-deploy>

Specifies ear expansion.

ear-deploy can be used in web-apps to define a subdirectory for ear expansion.

Table 31.16: <ear-deploy> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-path	The path to the directory containing ear files	path
ear-default	resin.xml default configuration for all ear files, e.g. configuring database, JMS or EJB defaults.	
expand-cleanup-fileset	Specifies the files which should be automatically deleted when a new .ear version is deployed.	
expand-directory	directory where ears should be expanded	value of path
expand-prefix	automatic prefix of the expanded directory	*ear*
expand-suffix	automatic suffix of the expanded directory	
lazy-init	if true, the ear file is only started on first access	false
path	The path to the deploy directory	required
redeploy-mode	"automatic" or "manual". If automatic, detects new .ear files automatically and deploys them.	automatic
url-prefix	optional URL prefix to group deployed .ear files	

```
element ear-deploy {
  path
  & archive-directory?
  & ear-default?
  & expand-cleanup-fileset?
  & expand-directory?
  & expand-path?
  & expand-prefix?
  & expand-suffix?
  & lazy-init?
  & redeploy-mode?
  & require-file*
  & url-prefix?
}
```

31.39 <ejb-message-bean>

<ejb-message-bean> configures a bean as a message listener. The listener can be a simple bean that just implements the `javax.jms.MessageListener interface` . No other packaging or complications are necessary. Resin will retrieve messages from a configured queue and pass them to the listener as they arrive. The listeners are typically pooled.

The bean has full access to Resin-IoC capabilities, including dependency injection, transaction attributes, and aspect interception.

The message bean can plug into custom messaging systems. The application will need to define a ResourceAdapter and an `ActivationSpec` .

Table 31.17: <ejb-message-bean> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
activation-spec	Configures a custom message-listener driver	
class	Classname of the listener bean	required
destination	Queue or Topic for JMS message receiving	
destination-type	javax.jms.Queue or javax.jms.Topic	
init	IoC configuration for the listener bean	
message-consumer-max	The number of listener instances to create for the pool.	5

```
element ejb-message-bean {
  class
  & init?
  & (activation-spec?
     | (destination?
        & destination-type?
        & destination-name?
        & message-consumer-max?)
    )
}
```

31.40 <ejb-server>

Configures an EJB server.

Table 31.18: <ejb-server> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
auto-compile	enables auto-compilation of EJB stubs and skeletons	true
create-database-schema	enables JPA auto-creation of missing database tables	false
data-source	specifies the default database for JPA	
config-directory	specifies a directory containing *.ejb configuration files	
ejb-descriptor	path to a *.ejb file to load	
ejb-jar	path to a jar file containing a META-INF/ejb-jar.xml with EJBs	

Table 31.18: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
jndi-prefix	prefix for JNDI registration of EJBs	
validate-database-schema	verifies the actual database tables against the JPA definitions	true
jms-connection-factory	specifies the default JMS ConnectionFactory for message beans	
xa-data-source	specifies a separate database for transactions	data-source

```
element ejb-server {
  auto-compile
  & create-database-schema
  & data-source
  & config-directory
  & ejb-descriptor
  & ejb-jar
  & jndi-prefix
  & validate-database-schema
  & jms-connection-factory
  & xa-data-source
}
```

31.41 <ejb-stateful-bean>

<ejb-stateful-bean> is deprecated and replaced by CDI-style configuration. EJB stateful beans are now either scanned automatically if configured with the @Stateful annotation or can be configured as CDI beans with the <ee:Stateful> tag.

Stateful bean in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:ee="urn:java:ee"
         xmlny:mypkg="urn:java:com.mycom.mypkg">

  <mypkg:MyBean>
    <ee:Stateful/>

    <my-attribute>my-value</my-attribute>
  </mypkg:MyBean>

</web-app>
```

31.42 <ejb-stateless-bean>

<ejb-stateless-bean> is deprecated and replaced by CDI-style configuration. EJB stateless beans are now either scanned automatically if configured with the @Stateless annotation or can be configured as CDI beans with the <ee:Stateless> tag.

Stateless bean in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:ee="urn:java:ee"
         xmlny:mypkg="urn:java:com.mycom.mypkg">
```

```
  <mypkg:MyBean>
    <ee:Stateless/>

    <my-attribute>my-value</my-attribute>
  </mypkg:MyBean>

</web-app>
```

31.43 <environment-system-properties>

By default, Resin's `System.getProperties()` is environment-dependent, so the settings in on web-app do not affect the properties in any other web-app. Some sites may need to disable this virtualization capability, when using certain JVM agents.

```
element environment-system-properties {
  r_boolean-Type
}
```

31.44 <env-entry>

<env-entry> configures a JNDI scalar value for JNDI-based application configuration.

Some application beans prefer to retrieve configuration data from JNDI, including String, Integer, and Double constants. env-entry configures that data in the current context.

The value can not use JSP-EL expressions, because the env-entry is part of the JavaEE spec. To set EL expressions, use resin:set instead.

Table 31.19: <env-entry> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
env-entry-name	JNDI name to store the value. Since Servlet 2.1	required
env-entry-type	Java type for the value. Since Servlet 2.1	required
env-entry-value	Value to be stored. Since Servlet 2.1	required

```
element env-entry {
  description*,

  env-entry-name,

  env-entry-type,

  env-entry-value
}
```

The example configuration stores a string in java:comp/env/greeting. Following the J2EE spec, the env-entry-name is relative to java:comp/env. If the env-entry is in the <host> context, it will be visible to all web-apps in the host.

Example: WEB-INF/resin-web.xml with env-entry

```
<web-app xmlns="http://caucho.com/ns/resin">
  <env-entry>
    <env-entry-name>greeting</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>Hello, World</env-entry-value>
  </env-entry>

  <servlet ...>
  </servlet>
</web-app>
```

The following servlet fragment is a typical use in a servlet. The servlet only looks up the variable once and stores it for later use.

Example: GreetingServlet.java

```
import java.io.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet extends HttpServlet {
  private String greeting;

  public void init()
    throws ServletException
  {
    try {
      Context env =
        (Context) new InitialContext().lookup("java:comp/env");
      greeting = (String) env.lookup("greeting");
    } catch (NamingException e) {
      throw new ServletException(e);
    }
  }

  ...
}
```

31.45 <error-page>

Allows applications to customize the response generated for errors. By default, Resin returns a 500 Servlet Error and a stack trace for exceptions and a simple 404 File Not Found for error pages. With <error-page>, you may specify a handler page for these errors.

The handler page has several request attributes set so that it may log, display, or otherwise use information about the error that occured. The following table describes the available attributes.

Table 31.20: Request attributes for error handling

ATTRIBUTE	TYPE
javax.servlet.error.status_code	java.lang.Integer
javax.servlet.error.message	java.lang.String
javax.servlet.error.request_uri	java.lang.String
javax.servlet.error.servlet_name	java.lang.String
javax.servlet.error.exception	java.lang.Throwable
javax.servlet.error.exception_type	java.lang.Class

Table 31.21: <error-page> Attributes

ATTRIBUTE	DESCRIPTION
error-code	Select the error page based on an HTTP status code
exception-type	Select the error page based on a Java exception
location	The error page to display

```
element error-page {
  (error-code | exception-type)?
  & location
}
```

Catching File Not Found

```
<web-app xmlns="http://caucho.com/ns/resin">
  <error-page>
    <error-code>404</error-code>
    <location>/file_not_found.jsp</location>
  </error-page>
</web-app>
```

Catching Exceptions

```
<web-app xmlns="http://caucho.com/ns/resin">
   <error-page exception-type="java.lang.NullPointerException"
               location="/nullpointer.jsp"/>
</web-app>
```

Using request attributes to obtain information about the request that caused the error

```
<%@ page session="false" isErrorPage="true" %>

<html>
<head><title>404 Not Found</title></head>
<body>
<h1>404 Not Found</h1>

The url <code>${requestScope["javax.servlet.error.request_uri"]}</code>
was not found.
</body>
</html>
```

31.46 <expand-cleanup-fileset>

The expand-cleanup-fileset tag lets you configure which files are to be kept on a .war redeploy. By default, Resin deletes the entire .war directory on a restart.

31.47 <fileset>

<fileset> provides the ability to match a set of files. It is modelled after the ant tag by the same name. The fileset matches files from a base directory defined by *dir*. Files can be included by patterns defined by <include> tags or excluded by patterns defined in <exclude> tags.

A pattern can contain two special characters: *and* . *matches any part of path, but does not match the path separator.* matches any part of a path, including the path separator.

Table 31.22: <fileset> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
dir	the starting directory	required
include	an include pattern	do not include all files
exclude	an exclude pattern	do not exclude any files

```
element fileset {
  dir
  & exclude*
  & include*
```

Matching jars in WEB-INF/lib (non-recursive)

```
<fileset dir="WEB-INF/lib">
  <include name="*.jar"/>
</fileset>

MATCH    lib/foo.jar
MATCH    lib/bar.jar
NO MATCH lib/baz/foo.jar
```

Matching jars in WEB-INF/lib (recursive)

```
<fileset dir="WEB-INF/tree">
  <include name="**/*.jar"/>
</fileset>

MATCH    lib/foo.jar
MATCH    lib/bar.jar
MATCH    lib/baz/foo.jar
```

31.48 <filter>

Defines a filter name for later mapping. Because Filters are fully integrated with admin/config-candi.xtp , they can use dependency-injection, transactional aspects, custom interception with @InterceptorType, and event handling with @Observes.

Table 31.23: <filter> Attributes

ATTRIBUTE	DESCRIPTION
filter-name	The filter's name (alias)
filter-class	The filter's class (defaults to filter-name), which extends javax.servlet.Filter
init	Resin-IoC initialization configuration, see admin/config-candi.xtp
init-param	Initialization parameters, see FilterConfig.getInitParameter

```
element filter {
  filter-name
  & filter-class
  & init*
  & init-param*
}
```

Defining a filter name image

```
<web-app xmlns="http://caucho.com/ns/resin">

  <filter>
    <filter-name>image</filter-name>
    <filter-class>test.MyImage</filter-class>
    <init-param>
      <param-name>title</param-name>
      <param-value>Hello, World</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>image</filter-name>
    <url-pattern>/images/*</url-pattern>
  </filter-mapping>

</web-app>
```

init-param shortcut syntax

```
<web-app id='/'>

<filter filter-name='test.HelloWorld'>
  <init-param foo='bar'/>

  <init-param>
    <param-name>baz</param-name>
    <param-value>value</param-value>
  </init-param>
</servlet>

</web-app>
```

31.49 <filter-mapping>

Maps url patterns to filters. filter-mapping has two children, url-pattern and filter-name . url-pattern selects the urls which should execute the filter.

`filter-name` can either specify a filter class directly or it can specify a filter alias defined by filter .

Table 31.24: <filter-mapping> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
dispatcher	REQUEST, INCLUDE, FORWARD, ERROR	REQUEST
filter-name	The filter name	required
url-pattern	A pattern matching the url: /foo/* , /foo , or *.foo	
url-regexp	A regular expression matching the portion of the url that follows the	

```
element filter-mapping {
  (url-pattern | url-regexp | servlet-name)+
  & filter-name
  & dispatcher*
}
```

Example: resin-web.xml filters

```
<web-app xmlns="http://caucho.com/ns/resin">

  <filter>
    <filter-name>test-filter</filter-name>
    <filter-class>test.MyFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>test-filter</filter-name>
    <url-pattern>/hello/*</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>test.HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

31.50 <form-login-config>

Configures authentication using forms. The login form has specific parameters that the servlet engine's login form processing understands. If the login succeeds, the user will see the original page. If it fails, she will see the error page.

The form itself must have the action j_security_check . It must also have the parameters j_username and j_password . Optionally, it can also have j_uri and j_use_cookie_auth . j_uri gives the next page to display when login succeeds. j_use_cookie_auth allows Resin to send a persistent cookie to the user to make following login easier.

j_use_cookie_auth gives control to the user whether to generate a persistent cookie. It lets you implement the "remember me" button. By default, the authentication only lasts for a single session.

Table 31.25: <form-login-config> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
form-login-page	The page to be used to prompt the user login	
form-error-page	The error page for unsuccessful login	
internal-forward	Use an internal redirect on success instead of a sendRedirect	false
form-uri-priority	If true, the form's j_uri will override a stored URI	false

Table 31.26: Special form input names

ATTRIBUTE	DESCRIPTION
j_security_check	The form's mandatory action
j_username	The user name
j_password	The password
j_uri	Optional Resin extension for the successful display page.
j_use_cookie_auth	Optional Resin extension to allow cookie login.

```
element form-login-config {
  form-login-page,
  form-error-page,
  internal-forward,
  form-uri-priority
}
```

Example: login.html

```
<form action='j_security_check' method='POST'>
<table>
<tr><td>User:<td><input name='j_username'>
<tr><td>Password:<td><input name='j_password'>
<tr><td colspan=2>hint: the password is 'quidditch'
<tr><td><input type=submit>
</table>
</form>
```

31.51 <group-name>

<group-name> configures the operating system group Resin should run as. Since the HTTP port 80 is protected in Unix, the web server needs to start as root to bind to port 80. For security, Resin should switch to a non-root user after binding to port 80.

resin.xml with user-name

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">

    <server-default>
      <http port="80"/>

      <user-name>resin</user-name>
      <group-name>www</group-name>
    </server-default>

    <server id="web-a"/>
    ...
  </cluster>
</resin>
```

31.52 <health:ActionSequence>

Executes a sequence of child health actions in order.

Table 31.27: <health:ActionSequence> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:ActionSequence> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:ActionSequence>
    <health:DumpThreads/>
    <health:DumpHeap/>
    <health:IfHealthCritical time="5m"/>
  </health:ActionSequence>

</cluster>
```

31.53 <health:And>

Qualifies an action to match if all of the child predicates match.

Note: <health:And> is implied and thus not strictly necessary except when used in conjunction with more complex combining conditions.

Table 31.28: <health:And> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:And> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:And>
      <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
      <health:IfHealthCritical health-check="${memoryPermGenHealthCheck}"/>
    </health:And>
  </health:Restart>

</cluster>
```

31.54 <health:AnomalyAnalyzer>

AnomalyAnalyzer examines a meter value, checking for deviations from the average value. So unusual changes like a spike in blocked threads can be detected, logged, and trigger health actions.

Table 31.29: <health:AnomalyAnalyzer> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
meter	Name of the meter to analyze (ie. from <health:JmxMeter>)	String	required
health-event	A string to use to match using <health:IfHealthEvent>	String	None: when absent no health event will fire
min-samples	Minimum number of samples required to calculate an average	int	60 (typically 1 hour of data)
sigma-threshold	The number of standard deviations for a sample to be considered an anomaly.	int	5

Example: <health:AnomalyAnalyzer> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JmxMeter>
    <name>JVM|Thread|JVM Blocked Count</name>
    <objectName>resin:type=JvmThreads</objectName>
    <attribute>BlockedCount</attribute>
  </health:JmxMeter>

  <health:AnomalyAnalyzer>
    <meter>JVM|Thread|JVM Blocked Count</meter>
    <health-event>caucho.thread.anomaly.jvm-blocked</health-event>
  </health:AnomalyAnalyzer>

  <health:DumpThreads>
    <health:IfHealthEvent regexp="caucho.thread"/>
    <health:IfNotRecent time="15m"/>
  </health:DumpThreads>

</cluster>
```

31.55 <health:CallJmxOperation>

Executes a JMX MBean operation with parameters.

Note: Calling a JMX operation can also be performed on-demand using the jmx-call command line.

Table 31.30: <health:CallJmxOperation> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
objectName	The JMX MBean name	String	N/A
operation	The method name	String	N/A
operationIndex	Unique method index in case multiple methods match operation	int	-1

Table 31.30: (continued)

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
param	Method parameters that will be converted to the appropriate type.	String	N/A

Example: <health:CallJmxOperation> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:CallJmxOperation>
    <objectName>java.lang:type=Threading</objectName>
    <operation>resetPeakThreadCount</operation>
    <health:IfNotRecent time='5m'/>
  </health:CallJmxOperation>

</cluster>
```

31.56 <health:ConnectionPoolHealthCheck>

Monitors the health of Resin database connection pools. See <database> for additional information.

Table 31.31: <health:ConnectionPoolHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true

Table 31.32: <health:ConnectionPoolHealthCheck> Conditions

HEALTHSTATUS	CONDITION
WARNING	Upon exceeding max-connections .
CRITICAL	Upon exceeding max-overflow-connections .

Example: <health:ConnectionPoolHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:ConnectionPoolHealthCheck/>

</cluster>
```

31.57 <health:CpuHealthCheck>

Monitors CPU usage. On multi-core machines, each CPU is checked individually.

Table 31.33: <health:CpuHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
warning-threshold	CPU usage warning threshold	int (percentage 0-100)	95
critical-threshold	CPU usage critical threshold	int (percentage 0-100)	200 (disabled)

Table 31.34: <health:CpuHealthCheck> Conditions

HEALTHSTATUS	CONDITION
WARNING	Upon exceeding warning-threshold on any CPU.
CRITICAL	Upon exceeding critical-threshold on any CPU.

Example: <health:CpuHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:CpuHealthCheck>
    <warning-threshold>95</warning-threshold>
    <critical-threshold>99</critical-threshold>
  </health:CpuHealthCheck>

</cluster>
```

31.58 <health:DumpHeap>

Create a memory heap dump. The heap dump will be logged to the internal log database and to the resin log file using `com.caucho.health.action.DumpHeap` as the class at info level.

Note: Creating a heap dump can also be performed on-demand using the heap-dump command line, and from /resin-admin.

Table 31.35: <health:DumpHeap> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
hprof	Creates an HPROF format dump instead of human readable type.	boolean	false

Table 31.35: (continued)

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
hprof-path	Output path write HPROF files, if hprof = true	String	log/heap.hprof
hprof-path-format	Selects a format for generating dynamic path names using timestamp tokens.	String	
log	If true, JMX dump is written to the server log in addition to being stored in the internal Resin database	boolean	true

Example: <health:DumpHeap> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:DumpHeap/>

</cluster>
```

Example: <health:DumpHeap> in health.xml with hprof-path-format

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:DumpHeap>
    <hprof>true</hprof>
    <hprof-path-format>${resin.home}/log/dump-%H:%M:%S.%s.hprof</hprof-path-format>
    <health:OnAbnormalStop/>
  </health:DumpHeap>

</cluster>
```

31.59 <health:DumpHprofHeap>

Shortcut for <health:DumpHeap hprof=*true*/>

31.60 <health:DumpJmx>

Health action to create a dump of all JMX attributes and values. The JMX dump will be logged to the internal log database and to the resin log file using `com.caucho.health.action.DumpJmx` as the class at info level.

Note: Creating a JMX dump can also be performed on-demand using the jmx-dump command line.

Table 31.36: <health:DumpJmx> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
log	If true, JMX dump is written to the server log in addition to being stored in the internal Resin database	boolean	false

Example: <health:DumpJmx> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:DumpJmx/>

</cluster>
```

31.61 <health:DumpThreads>

Create a thread dump. The thread dump will be logged to the internal log database and log file using `com.caucho.health.action.`

as the class at info level.

Note: Creating a thread dump can also be performed on-demand using the thread-dump command line.

Table 31.37: <health:DumpThreads> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
only-active	Output only currently active threads (RUNNABLE state)	boolean	false
log	If true, JMX dump is written to the server log in addition to being stored in the internal Resin database	boolean	true

Example: <health:DumpThreads> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:DumpThreads>
    <only-active>false</only-active>
  </health:DumpThreads>

</cluster>
```

31.62 <health:ExecCommand>

Execute an operating system shell command.

Table 31.38: <health:ExecCommand> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
command	The command to execute, relative to dir if set	String	N/A
dir	The directory to execute from	java.io.File	N/A
timeout	Timeout on execution of the command, after which it will be killed if not complete	Period	2s
env	A custom env variable, available to the command.	Name/Value Pair	N/A (System variables are available by default)

Example: <health:ExecCommand> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:ExecCommand>
    <dir>/tmp</dir>
    <command>remediation.sh</command>
    <timeout>2s</timeout>
    <env>
      <name>resin_home</name>
      <value>${resin.home}</value>
    </env>
    <env>
      <name>password</name>
      <value>foo</value>
    </env>
  </health:ExecCommand>

</cluster>
```

31.63 <health:ExprHealthCheck>

Evaluates user supplied EL expressions to a boolean.

Table 31.39: <health:ExprHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
fatal-test	EL expression that will trigger FATAL if it evaluates to true	Expr	N/A

Table 31.39: (continued)

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
critical-test	EL expression that will trigger CRITICAL if it evaluates to true	Expr	N/A
warning-test	EL expression that will trigger WARNING if it evaluates to true	Expr	N/A

Table 31.40: <health:ExprHealthCheck> Conditions

HEALTHSTATUS	CONDITION
FATAL	If any fatal-test expression evaluates to true
CRITICAL	If any critical-test expression evaluates to true
WARNING	If any warning-test expression evaluates to true

Example: <health:ExprHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:ExprHealthCheck>
    <critical-test>${mbean('java.lang:type=Threading').ThreadCount > 100}</critical-test>
  </health:ExprHealthCheck>

</cluster>
```

31.64 <health:FailSafeRestart>

A timed restart of Resin, normally used in conjunction with an ActionSequence to gather shutdown information

Table 31.41: <health:FailSafeRestart> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
timeout	Time to force a restart if one has not yet occurred.	Period	N/A

Example: <health:FailSafeRestart> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">
```

```
  <health:ActionSequence>
    <health:FailSafeRestart timeout="10m"/>
    <health:DumpThreads/>
    <health:DumpHeap/>
    <health:StartProfiler active-time="5m"/>
    <health:Restart/>

    <health:IfHealthCritical time="5m"/>
  </health:ActionSequence>

</cluster>
```

31.65 <health:HealthSystem>

Configures overall health checking frequency and recheck rules. This element is present in health.xml for clarity, but is not strictly required since it will be created upon startup with default values.

Table 31.42: <health:HealthSystem> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	All health checking enabled or disabled	boolean	true
startup-delay	The time after startup before actions are triggered (checks still execute).	Period	15m
period	The time between checks	Period	5m
recheck-period	The time between rechecks	Period	30s
recheck-max	The number of rechecks before returning to the normal checking period	int	10

Example: <health:HealthSystem> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HealthSystem>
    <enabled>true</enabled>
    <startup-delay>15m</startup-delay>
    <period>5m</period>
    <recheck-period>30s</recheck-period>
    <recheck-max>10</recheck-max>
  </health:HealthSystem>

</cluster>
```

31.66 <health:HealthSystemHealthCheck>

Monitors the health system itself by using a separate thread to detect if health checking is frozen or taking too long.

Table 31.43: <health:HealthSystemHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
thread-check-period	The polling frequency of the independent thread.	Period	1m
freeze-timeout	The max time for no health checks to occur to declare the health system frozen	Period	15m

Table 31.44: <health:HealthSystemHealthCheck> Conditions

HEALTHSTATUS	CONDITION
FATAL	If health checking has not occurred within freeze-timeout .
FATAL	If health checking has not completed within an acceptable time, calculated using <health:HealthSystem> startup-delay , period , and recheck-period .

Example: <health:HealthSystemHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HealthSystemHealthCheck>
    <thread-check-period>1m</thread-check-period>
    <freeze-timeout>15m</freeze-timeout>
  </health:HealthSystemHealthCheck>

</cluster>
```

31.67 <health:HeartbeatHealthCheck>

Monitors for heartbeats from other members of the cluster.

Table 31.45: <health:HeartbeatHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true

Table 31.46: <health:HeartbeatHealthCheck> Conditions

HEALTHSTATUS	CONDITION
WARNING	If no heartbeat has been received from a know member of the cluster.
WARNING	If a heartbeat has not been received in the last 180 seconds from a known member of the cluster.

Example: <health:HeartbeatHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HeartbeatHealthCheck/>

</cluster>
```

31.68 <health:HttpStatusHealthCheck>

Monitors one or more URLs on the current Resin instance by making an HTTP GET request and comparing the returned HTTP status code to a pattern.

Table 31.47: <health:HttpStatusHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
ping-host	The server's ping host (for use where url is a URI)	String	N/A
ping-port	The server's ping port (for use where url is a URI)	int	80
url	A URL or URI to be tested	String	N/A
socket-timeout	The socket connection timeout	Period	10s
regexp	The HTTP status regular expression	java.util.regex.Pattern	200

Table 31.48: <health:HttpStatusHealthCheck> Conditions

HEALTHSTATUS	CONDITION
CRITICAL	If the HTTP GET request failed to connect or the status code does not match the regexp.

Example: <health:HttpStatusHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck>
    <ping-host>localhost</ping-host>
    <ping-port>8080</ping-port>
    <url>/custom-test-1.jsp</url>
    <url>/custom-test-2.jsp</url>
    <socket-timeout>2s</socket-timeout>
    <regexp>^2|^3</regexp>
  </health:HttpStatusHealthCheck>

</cluster>
```

In some clustered configurations it may be simpler to use the

`<server-default>` `<ping-url>` element rather than specifying the host and port in the health check itself. `<ping-url>` will dynamically construct a URL using the current server host. To enabled this, configure HttpStatusHealthCheck with no `<url>` elements and add

`<ping-url>` to `<server>` or

`<server-default>` .

Example: <health:HttpStatusHealthCheck> using ping-url

```
<!-- resin.xml -->
<resin xmlns="http://caucho.com/ns/resin">

  <cluster id="web-tier">
    <server-default>
      <ping-url>/ping-test.jsp<ping-url>
    </server-default>
    ...
  </cluster>
</resin>

<!-- health.xml -->
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck ee:Named="serverHttpPingCheck">
    <socket-timeout>5s</socket-timeout>
    <regexp>200</regexp>
  </health:HttpStatusHealthCheck>

</cluster>
```

31.69 <health:IfCron>

Qualify an action to execute if the current time is in an active range configured by cron-style times. This can be used both to schedule regular actions or to prevent restarts or other actions during critical times.

Table 31.49: <health:IfCron> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enable-at	The cron enable times	CronType	N/A
disable-at-at	The cron diable times	CronType	N/A

Example: <health:IfCron> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfCron>
      <enable-at>0 0 * * *</enable-at>
      <disable-at>5 0 * * *</disable-at>
    </health:IfCron>
  </health:Restart>

</cluster>
```

31.70 <health:IfExpr>

Qualifies an action to execute based on the evaluation of an JSP EL expression. Expression can include references to system properties, config properties, and JMX mbean attributes.

Table 31.50: <health:IfExpr> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
test	the JSP-EL expression value	Expr	N/A

Example: <health:IfExpr> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfExpr>
      <test>${mbean('java.lang:type=Threading').ThreadCount > 100}</test>
    </health:IfExpr>
  </health:Restart>

</cluster>
```

31.71 <health:IfHealthCritical>

Qualifies an action to match if health status is CRITICAL.

Table 31.51: <health:IfHealthCritical> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
time	The minimum amount of time since the status started	Period	N/A

Example: <health:IfHealthCritical> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
  </health:Restart>

</cluster>
```

31.72 <health:IfHealthEvent>

Causes an action to fire in response to a matching health event. This is usually used in combination with AnomalyAnalyzer with a <health-event> attribute.

Table 31.52: <health:IfHealthEvent> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
regexp	A regular expression the event must match.	java.util.regex.Pattern	required

Example: <health:IfHealthEvent> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JmxMeter>
    <name>JVM|Thread|JVM Blocked Count</name>
    <objectName>resin:type=JvmThreads</objectName>
    <attribute>BlockedCount</attribute>
  </health:JmxMeter>

  <health:AnomalyAnalyzer>
```

```
    <meter>JVM|Thread|JVM Blocked Count</meter>
    <health-event>caucho.thread.anomaly.jvm-blocked</health-event>
  </health:AnomalyAnalyzer>

  <health:DumpThreads>
    <health:IfHealthEvent regexp="caucho.thread"/>
    <health:IfNotRecent time="15m"/>
  </health:DumpThreads>

</cluster>
```

31.73 <health:IfHealthFatal>

Qualifies an action to match if health status is FATAL.

Table 31.53: <health:IfHealthFatal> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
time	The minimum amount of time since the status started	Period	N/A

Example: <health:IfHealthFatal> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthFatal health-check="${memoryTenuredHealthCheck}"/>
  </health:Restart>

</cluster>
```

31.74 <health:IfHealthOk>

Qualifies an action to match if health status is OK.

Table 31.54: <health:IfHealthOk> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
time	The minimum amount of time since the status started	Period	N/A

Example: <health:IfHealthOk> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:Not>
      <health:IfHealthOk health-check="${memoryTenuredHealthCheck}"/>
    </health:Not>
  </health:Restart>

</cluster>
```

31.75 <health:IfHealthUnknown>

Qualifies an action to match if health status is UNKNOWN.

Table 31.55: <health:IfHealthUnknown> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
time	The minimum amount of time since the status started	Period	N/A

Example: <health:IfHealthUnknown> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthUnknown health-check="${memoryTenuredHealthCheck}"/>
  </health:Restart>

</cluster>
```

31.76 <health:IfHealthWarning>

Qualifies an action to match if health status is WARNING.

Table 31.56: <health:IfHealthWarning> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
time	The minimum amount of time since the status started	Period	N/A

Example: <health:IfHealthWarning> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthWarning health-check="${memoryTenuredHealthCheck}"/>
  </health:Restart>

</cluster>
```

31.77 <health:IfMessage>

Qualifies an action to match the health result message to a regular expression.

Table 31.57: <health:IfMessage> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)
regexp	The health message match regular expression	java.util.regex.Pattern	N/A

Example: <health:IfMessage> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthCritical/>
    <health:IfMessage health-check="${httpStatusCheck}" regexp="Not Found"/>
  </health:Restart>

</cluster>
```

31.78 <health:IfNotRecent>

Qualifies an action to match at most an amount of time after the last execution. This is usefull to prevent unecessary frequent execution of an action.

Table 31.58: <health:IfNotRecent> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
time	The time before the action can execute again.	Period	N/A

Example: <health:IfNotRecent> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:HttpStatusHealthCheck ee:Named="httpStatusCheck">
    <url>http://localhost:8080/test-ping.jsp</url>
  </health:HttpStatusHealthCheck>

  <health:DumpHeap>
    <health:IfHealthCritical healthCheck="${httpStatusCheck}"/>
    <health:IfNotRecent time='5m'/>
  </health:DumpHeap>

</cluster>
```

31.79 <health:IfRecovered>

Qualifies an action to match upon recovery. Recovery is defined as the state change from FATAL, CRITICAL, or WARNING to OK.

Table 31.59: <health:IfRecovered> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
health-check	The target health check	HealthCheck	N/A (Overall Resin health will be used if absent)

Example: <health:IfRecovered> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SendMail>
    <to>admin@yourdomain</to>
    <health:IfRecovered health-check="${cpuHealthCheck}"/>
  </health:SendMail>

</cluster>
```

31.80 <health:IfRechecked>

Qualifies an action to match only after the required number of rechecks have been performed. Since rechecking is not a health check specific condition, this predicate simply matches when recheck cycle count matches the HealthSystem parameter recheck-max .

Table 31.60: <health:IfRechecked> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:IfRechecked> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthFatal/>
    <health:IfRechecked/>
  </health:Restart>

</cluster>
```

31.81 <health:IfUptime>

Qualifies an action to match an amount of time after startup.

Table 31.61: <health:IfUptime> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
limit	The time after startup (at least)	Period	N/A

Example: <health:IfUptime> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfUptime limit="12h"/>
  </health:Restart>

</cluster>
```

31.82 <health:JmxDeltaMeter>

Creates a meter that graphs the difference between the current and previous values of a numeric JMX MBean attribute.

Table 31.62: <health:JmxDeltaMeter> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
name	The name of the meter to display in /resin-admin (see #Meter names)	String	N/A
objectName	The JMX MBean name	String	N/A
attribute	The MBean attribute to sample	String	N/A

Example: <health:JmxDeltaMeter> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JmxDeltaMeter>
    <name>JVM|Compilation|Compilation Time</name>
    <object-name>java.lang:type=Compilation</object-name>
    <attribute>TotalCompilationTime</attribute>
  </health:JmxDeltaMeter>

</cluster>
```

31.83 <health:JmxMeter>

Creates a meter that graphs the current value of a numeric JMX MBean attribute.

Table 31.63: <health:JmxMeter> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
name	The name of the meter to display in /resin-admin (see #Meter names)	String	N/A
objectName	The JMX MBean name	String	N/A
attribute	The MBean attribute to sample	String	N/A

Example: <health:JmxMeter> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JmxMeter>
    <name>OS|Memory|Physical Memory Free</name>
    <object-name>java.lang:type=OperatingSystem</object-name>
    <attribute>FreePhysicalMemorySize</attribute>
  </health:JmxMeter>
```

```
</cluster>
```

31.84 <health:JvmDeadlockHealthCheck>

Monitors for deadlocked threads, as determined by the JVM.

Table 31.64: <health:JvmDeadlockHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true

Table 31.65: <health:JvmDeadlockHealthCheck> Conditions

HEALTHSTATUS	CONDITION
FATAL	If deadlocked threads are detected.

Example: <health:JvmDeadlockHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:JvmDeadlockHealthCheck/>

</cluster>
```

31.85 <health:LicenseHealthCheck>

Checks for expiring Resin Pro license.

Table 31.66: <health:LicenseHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
warning-period	License check warning period	Period	30D

Table 31.67: <health:LicenseHealthCheck> Conditions

HEALTHSTATUS	CONDITION
WARNING	If license expires in less than warning-period.

Example: <health:LicenseHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:LicenseHealthCheck>
    <warning-period>30D</warning-period>
  </health:LicenseHealthCheck>

</cluster>
```

31.86 <health:MemoryPermGenHealthCheck>

Monitors the amount of free memory in the JVM PermGen memory pool. Requests a garbage collection if memory falls too low.

Note: This check does not apply to all JVM vendor implementations, and will report UNKNOWN if there is no PermGen pool.

Table 31.68: <health:MemoryPermGenHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
memory-free-min	The critical minimum amount of free memory.	Bytes	1m
free-warning	The warning threshold percentage.	double (percentage 0.0 - 100.0)	0.01
objectName	Explicitly set the MBean name to query for memory stats. When left unset the health check will search available MBeans for the appropriate memory stats MBean.	javax.management.ObjectName	N/A

Table 31.69: <health:MemoryPermGenHealthCheck> Conditions

HEALTHSTATUS	CONDITION
UNKNOWN	If there is no PermGen pool (JVM vendor dependent) or the appropriate MBean could not be determined.
WARNING	If free memory is below free-warning percentage after a GC.
CRITICAL	If free memory is below memory-free-min after a GC.

Example: <health:MemoryPermGenHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:MemoryPermGenHealthCheck>
    <memory-free-min>1m</memory-free-min>
    <free-warning>0.01</free-warning>
  </health:MemoryPermGenHealthCheck>

</cluster>
```

31.87 <health:MemoryTenuredHealthCheck>

Monitors the amount of free memory in the JVM Tenured memory pool. Requests a garbage collection if memory falls too low. This check will monitor heap memory on JVMs where there is no tenured pool.

Note: memory-free-min will default to the value of

`<server> <memory-free-min>` if present in resin.xml.

Table 31.70: <health:MemoryTenuredHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true
memory-free-min	The critical minimum amount of free memory.	Bytes	1m
free-warning	The warning threshold percentage.	double (percentage 0.0 - 100.0)	0.01
objectName	Explicitly set the MBean name to query for memory stats. When left unset the health check will search available MBeans for the appropriate memory stats MBean.	javax.management.ObjectName	N/A

Table 31.71: <health:MemoryTenuredHealthCheck> Conditions

HEALTHSTATUS	CONDITION
UNKNOWN	If there is no PermGen pool (JVM vendor dependent) or the appropriate MBean could not be determined.
WARNING	If free memory is below free-warning percentage after a GC.
CRITICAL	If free memory is below memory-free-min after a GC.

Example: <health:MemoryTenuredHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:MemoryTenuredHealthCheck>
    <memory-free-min>1m</memory-free-min>
    <free-warning>0.01</free-warning>
  </health:MemoryTenuredHealthCheck>

</cluster>
```

31.88 <health:Nand>

Qualifies an action to match if all of the child predicates fail.

Table 31.72: <health:Nand> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:Nand> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:Nand>
      <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
      <health:IfHealthCritical health-check="${memoryPermGenHealthCheck}"/>
    </health:Nand>
  </health:Restart>

</cluster>
```

31.89 <health:Nor>

Qualifies an action to match if none of the child predicates match.

Table 31.73: <health:Nor> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:Nor> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
```

```
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:Nor>
      <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
      <health:IfHealthCritical health-check="${memoryPermGenHealthCheck}"/>
    </health:Nor>
  </health:Restart>

</cluster>
```

31.90 <health:Not>

Qualifies an action to match if the child predicate is false.

Table 31.74: <health:Not> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:Not> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
    <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
    <health:Not>
      <health:IfCron>
        <enable-at>0 7 * * *</enable-at>
        <disable-at>0 11 * * *</disable-at>
      </health:IfCron>
    </health:Not>
  </health:Restart>

</cluster>
```

31.91 <health:OnAbnormalStop>

Qualifies an action to match only when Resin is stopping with an non-OK exit code.

Table 31.75: <health:OnAbnormalStop> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
normal-exit-code	Add an OK exit code; one that will not trigger this condition.	com.caucho.env.shutdown.ExitCode	[CONFIG_MODIFIED,WATCHDOG_EXIT

Example: <health:OnAbnormalStop> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:PdfReport snapshot='true'>
    <health:OnAbnormalStop/>
  </health:PdfReport

</cluster>
```

31.92 <health:OnRestart>

Qualifies an action to match only when Resin is restarted by the watchdog. This generally only occurs during an error condition. #health:OnStart will fire during this event also.

Table 31.76: <health:OnRestart> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:OnRestart> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SendMail>
    <to>admin@yourdomain.com</to>
    <health:OnRestart/>
  </health:SendMail>

</cluster>
```

31.93 <health:OnStart>

Qualifies an action to match only when Resin is starting.

Table 31.77: <health:OnStart> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:OnStart> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
```

```
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SendMail>
    <to>admin@yourdomain.com</to>
    <health:OnStart/>
  </health:SendMail>

</cluster>
```

31.94 <health:OnStop>

Qualifies an action to match only when Resin is stopping.

Table 31.78: <health:OnStop> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:OnStop> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SendMail>
    <to>admin@yourdomain.com</to>
    <health:OnStop/>
  </health:SendMail>

</cluster>
```

31.95 <health:Or>

Qualifies an action to match if any of the child predicates match.

Table 31.79: <health:Or> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:Or> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart>
```

```
    <health:Or>
      <health:IfHealthCritical health-check="${memoryTenuredHealthCheck}"/>
      <health:IfHealthCritical health-check="${memoryPermGenHealthCheck}"/>
    </health:Or>
  </health:Restart>

</cluster>
```

31.96 <health:PdfReport>

Health action to generate a PDF report from a PHP script.

Table 31.80: <health:PdfReport> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
path	Path to a PDF generating .php file	String	${resin.home}/doc/admin/pdf-gen.php
report	Report type key	String	Summary
period	Report look back period of time	Period	7D
log-path	PDF output directory	String	${resin.logDirectory}

Example: <health:PdfReport> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:PdfReport>
    <path>${resin.home}/doc/admin/pdf-gen.php</path>
    <report>Summary</report>
    <period>7D</report>
    <health:IfCron value="0 0 * * 0"/>
  </health:PdfReport>

</cluster>
```

31.97 <health:Restart>

Restart Resin. Resin will exit with the reason HEALTH.

Table 31.81: <health:Restart> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT

Example: <health:Restart> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Restart/>

</cluster>
```

31.98 <health:SendMail>

Send an email containing a summary of the current Resin health status.

Table 31.82: <health:SendMail> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
to	A "TO:" address; a mail recipient	String	N/A
from	The "FROM:" address	String	resin@localhost
mail	A <mail> resource to use, see example	javax.mail.Session	N/A

Without the mail parameter, the default behaviour for sending mail is to contact an SMTP server at host 127.0.0.1 (the localhost) on port 25. System properties can be used to configure a different SMTP server.

resin.xml - smtp server configuration

```
  <system-property mail.smtp.host="127.0.0.1"/>
  <system-property mail.smtp.port="25"/>
```

Example: <health:SendMail> in health.xml using system properties

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SendMail>
    <to>admin@yourdomain.com</to>
    <to>another_admin@yourdomain.com</to>
    <from>resin@yourdomain.com</from>
  <health:SendMail>

</cluster>
```

Much more advanced SMTP options can be set by configuring a <mail> resource to use for sending health alerts.

Example: <health:SendMail> in health.xml referencing a <mail> resource

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <mail name="healthMailer">
```

```
    <smtp-host>mail.yourdomain.com</smtp-host>
    <smtp-port>25</smtp-port>
    <from>resin@yourdomain.com</from>
  </mail>

  <health:SendMail mail="${healthMailer}">
    <to>admin@yourdomain.com</to>
  <health:SendMail>

</cluster>
```

31.99 <health:SetJmxAttribute>

Sets a JMX attribute to a value.

Note: Setting a JMX attribute can also be performed on-demand using the jmx-set command line.

Table 31.83: <health:SetJmxAttribute> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
objectName	The JMX MBean name	String	N/A
attribute	The attribute name	String	N/A
value	New attribute value that will be converted to the appropriate type.	String	N/A

Example: <health:SetJmxAttribute> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:SetJmxAttribute>
    <objectName>java.lang:type=ClassLoading</objectName>
    <attribute>Verbose</attribute>
    <value>true</value>
    <health:OnStart/>
  </health:SetJmxAttribute>

</cluster>
```

31.100 <health:Snapshot>

A specific sequence of health actions: thread dump, heap dump, jmx dump, and pdf report. This is intended to generate a permanent representation, or "snapshot" of the system at a point in time that includes all the information necessary to debug server issues. It is usually intended to run in response to a unexpected issue.

Table 31.84: <health:Snapshot> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
log	Output to server log in addition to pdf report	Boolean	false
path	Path to a PDF generating .php file	String	${resin.home}/doc/admin/pdf-gen.php
report	Report type key	String	Summary
period	Report look back period of time	Period	7D

Example: <health:Snapshot> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:Snapshot>
    <health:OnAbnormalStop/>
  </health:Snapshot>

</cluster>
```

31.101 <health:StartProfiler>

Starts a profiler session. Results are logged to the internal database and the Resin log file at INFO level.

Note: Starting the profiler can also be performed on-demand using the profile command line, and from /resin-admin.

Table 31.85: <health:StartProfiler> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
active-time	The amount of time to run the profiler	Period	5s
sampling-rate	The sampling rate	Period	10ms
depth	The stack trace depth (use smaller number (8) for smaller impact, larger for more information.)	int	16

Example: <health:StartProfiler> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:ActionSequence>
    <health:FailSafeRestart timeout="10m"/>
    <health:DumpThreads/>
    <health:DumpHeap/>
```

```
    <health:StartProfiler active-time="5m"/>
    <health:Restart/>

    <health:IfHealthCritical time="5m"/>
  </health:ActionSequence>

</cluster>
```

31.102 <health:TransactionHealthCheck>

Monitors the Resin transaction manager for commit failures.

Table 31.86: <health:TransactionHealthCheck> Attributes

ATTRIBUTE	DESCRIPTION	TYPE	DEFAULT
enabled	Check is enabled or disabled	boolean	true

Table 31.87: <health:TransactionHealthCheck> Conditions

HEALTHSTATUS	CONDITION
WARNING	If there were commit failures since the last check.

Example: <health:TransactionHealthCheck> in health.xml

```
<cluster xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:health="urn:java:com.caucho.health"
         xmlns:ee="urn:java:ee">

  <health:TransactionHealthCheck/>

</cluster>
```

31.103 <home-cluster>

The <home-cluster> specifies the default --cluster value for dynamic servers. When Resin starts and elastic server, it looks for a <home-cluster> if the --cluster option is missing. The home cluster will be cluster that the Resin server tries to join.

```
element home-cluster {
  string
}
```

31.104 <host>

<host> configures a virtual host. Virtual hosts must be configured explicitly.

It is recommended that any <host> using a regexp include a <host-name> to set the canonical name for the host.

Table 31.88: <host> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
id	primary host name	none
regexp	Regular expression based host matching	none
host-name	Canonical host name	none
host-alias	Aliases matching the same host	none
secure-host-name	Host to use for a redirect to SSL	none
root-directory	Root directory for host files	parent directory
startup-mode	*automatic*, *lazy*, or *manual*, see startup-mode	automatic

Example: explicit host

```
<host host-name="www.foo.com">
  <host-alias>foo.com</host-alias>
  <host-alias>web.foo.com</host-alias>

  <root-directory>/opt/www/www.foo.com</root-directory>

  <web-app id="/" root-directory="webapps/ROOT">

  </web-app>
  ...
</host>
```

Example: regexp host

```
<host regexp="([^.]+)\.foo\.com">
  <host-name>${host.regexp[1]}.foo.com</host-name>

  <root-directory>/var/resin/hosts/www.${host.regexp[1]}.com</root-directory>

  ...
</host>
```

31.105 <host-alias>

<host-alias> defines a URL alias for matching HTTP requests. Any number of <host-alias> can be used for each alias.

The host-alias can be used either in the resin.xml or in a host.xml when use host-deploy together with resin:import.

Since the <host-deploy> and <host> tags lets you add a host.xml file to customize configuration, the <host-alias> can also fit in the custom host.xml page.

```
element host-alias {
  string
}
```

Example: host-alias in the resin.xml

```
<resin xmlns="http://caucho.com">
<cluster id="">

  <host id="www.foo.com" root-directory="/var/resin/foo.com">
```

```
    <host-alias>foo.com</host-alias>

    <web-app id=""/>
  </host>

</cluster>
</resin>
```

Example: host-alias in a /var/resin/hosts/foo/host.xml

```
<host xmlns="http://caucho.com">

  <host-name>www.foo.com</host-name>
  <host-alias>foo.com</host-alias>

  <web-app id="" root-directory="htdocs"/>

</host>
```

31.106 <host-alias-regexp>

<host-alias-regexp> defines a regular expression for matching URLs for a given virtual host.

```
element host-alias-regexp {
  string
}
```

Example: host-alias-regexp in the resin.xml

```
<resin xmlns="http://caucho.com">
<cluster id="">

  <host id="www.foo.com" root-directory="/var/resin/foo.com">
    <host-alias-regexp>.*foo.com</host-alias-regexp>

    <web-app id=""/>
  </host>

</cluster>
</resin>
```

31.107 <host-default>

Defaults for a virtual host.

The host-default can contain any of the host configuration tags. It will be used as defaults for any virtual host.

31.108 <host-deploy>

<host-deploy> configures an automatic deployment directory for virtual hosts.

The host-deploy will add an EL variable ${host.name}, referring to the name of the host jar file.

Table 31.89: <host-deploy> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-directory	path to the archive directory	path
path	path to the deploy directory	required
expand-cleanup-fileset	an ant-style fileset defining which directories to cleanup when an archive is redeployed	
expand-directory	path to the expansion directory	path
host-default	defaults for the expanded host	
host-name	the default hostname, based on the directory	

```
element host-deploy {
  archive-directory?
  & expand-cleanup-fileset?
  & expand-directory?
  & host-default?
  & host-name?
  & path?
}
```

The following example configures /var/resin/hosts as a host deployment directory. Each virtual host will have a webapps directory for .war deployment. So the directory /var/resin/hosts/www.foo.com/webapps/bar/test.jsp would serve the URL http://www.foo.com/-bar/test.jsp .

<host-deploy>

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <root-directory>/var/resin</root-directory>

    <host-deploy path="hosts">
      <host-default>
        <resin:import path="host.xml" optional="true"/>

        <web-app-deploy path="webapps"/>
      </host-default>
    </host-deploy>
  </cluster>
</resin>
```

31.109 <host-name>

<host-name> defines the canonical name for a virtual host. The <host-name> will be used in Resin's logging, management, and is available in the host's variables.

```
element host-name {
  string
}
```

31.110 <http>

<http> configures a HTTP or HTTPS port listening for HTTP requests.

When configuring multiple JVMs, each <http> will have a unique <server-id> which allows the -server command-line to select which ports the server should listen to. The virtual-host attribute overrides the browser's Host directive, specifying the explicit host and port for

`request.getServerName()` and `getServerPort()` . It is not used in most virtual host configurations. Only IP-based virtual hosts which wish to ignore the browser's Host will use @virtual-host.

Table 31.90: <http> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
address/host	IP address of the interface to listen to	*
port	port to listen to	required
accept-listen-backlog	configures operating system TCP listen queue size for the port	4000
accept-thread-idle-timeout	configures minimum spare accept-thread idle timeout	120s
accept-thread-min	configures the minimum number of threads listening for new connections	1
accept-thread-max	configures the maximum number of threads listening for new connections	unbound
keepalive-connection-time-max	configures an absolute max time for keepalive connections	600s
keepalive-max	configures the maximum number of sockets which can be used directly for keepalive connections	256
keepalive-select-enable	enables the select manager for keepalives	true
keepalive-select-thread-timeout	configures a short timeout allowing the select manager to wait for a keepalive before detaching the thread	1s
keepalive-timeout	configures how long a keepalive connection should wait for a new request before closing	120s
suspend-reaper-timeout	configures check interval for suspended (async)connections	60s
suspend-time-max	configures timeout for suspended (async) connections	600s
socket-timeout	configures timeout for accepted socket's write and read operations	120s
throttle-concurrent-max	configures maximum concurrent requests (Resin Pro)	unlimited
tcp-no-delay	sets the NO_DELAY socket parameter	true
socket-listen-backlog	The socket factory's listen backlog for receiving sockets	100
virtual-host	forces all requests to this <http> to use the named virtual host	none
openssl	configures the port to use OpenSSL	none
jsse-ssl	configures the port to use JSSE for SSL	none

```
element http {
  (id | server-id)
  & (address | host )?
  & port?
  & accept-listen-backlog?
  & accept-thread-idle-timeout?
  & accept-thread-min?
  & accept-thread-max?
  & connection-max?
  & keepalive-connection-time-max?
  & keepalive-max?
  & keepalive-select-enable?
  & keepalive-select-thread-timeout?
  & keepalive-timeout?
  & secure?
  & socket-timeout?
  & suspend-reaper-timeout?
  & suspend-time-max?
  & tcp-no-delay?
  & throttle-concurrent-max?
  & virtual-host?
  & (openssl | jsse-ssl)?
}
```

31.111 <idle-time>

The <idle-time> specifies the timeout for lazy-idle web-apps. In some configurations, web-apps are created only on demand and are closed when no requests access the web-app. The idle-time configures when those web-apps should be freed.

For example, the resin-doc web-app uses idle-time for its child web-apps because there are a large number of sub-web-apps for the individual tutorials.

```
element idle-time {
  r_period-Type
}
```

31.112 <ignore-client-disconnect>

ignore-client-disconnect configures whether Resin should ignore disconnection exceptions from the client, or if it should send those exceptions to the application.

```
element ignore-client-disconnect {
  r_boolean-Type
}
```

31.113 <invocation-cache-size>

Configures the number of entries in the invocation cache. The invocation cache is used to store pre-calculated servlet and filter chains from the URLs. It's also used as the basis for proxy caching.

```
element invocation-cache-size {
  r_int-Type
}
```

31.114 <invocation-cache-max-url-length>

Configures the longest entry cacheable in the invocation cache. It is used to avoid certain types of denial-of-service attacks.

```
element invocation-cache-max-url-length {
  r_int-Type
}
```

31.115 <javac>

<javac> configures the Java compiler for automatically compiled files.

The javac configuration is used for JSP, PHP, EJB and compiling-loader configuration. Thus you can deploy JSPs to Resin and it will use the compiler to generate classes at runtime. In other words, you do not need to precompile your JSPs, Java files, et al. to deploy them on Resin.

The internal compiler (recommended) requires tools.jar from the JDK installation, so a JDK must be used (not a JRE). Sometimes the internal compiler causes errors, creating exceptions or simply hanging and taking up a thread. The solution is to change the compiler to use an external compiler.

The javac compiler is included with the JDK. It performs the same function as the internal compiler, however it is executed as an external process and is less prone to the problems described for the internal compiler. In resin.xml with the javac configuration option:

Table 31.91: <javac> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
args	extra arguments to pass to the compiler	
compiler	the compiler name: eclipse, internal, or a command-line	
encoding	the character encoding to use	utf-8
max-batch	the maximum number of source files to batch into one compilation	64

```
element javac {
  args*
  & compiler
  & encoding?
  & max-batch?
}
```

Internal compiler

```
<resin xmlns="http://caucho.com/ns/resin">

 <javac compiler="internal" args=""/>

</resin>
```

javac JDK compiler

```
<resin xmlns="http://caucho.com/ns/resin">
```

```
 <javac compiler="javac" args=""/>

  ...

</resin>
```

31.116 <jndi-link>

<jndi-link> creates a symbolic link from one jndi name to another, or links to a foreign JNDI context.

Resin's JNDI can link to foreign JNDI contexts. For example, third-party EJB servers will often expose their EJB beans through a JNDI context. jndi-link will create the appropriate InitialContextFactory, configure it, and lookup the foreign JNDI objects.

Table 31.92: <jndi-link> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
factory	Class name of the JNDI InitialContextFactory. Since Resin 1.2	optional
foreign-name	The target name of the symbolic link, or the sub-context of the foreign JNDI context. Since Resin 1.2	none
init-param	Configuration parameters for the JNDI environment passed to InitialContextFactory. Since Resin 1.2	none
jndi-name	The JNDI name to use for the link. Resin 3.0	required

```
element jndi-link {
  jndi-name
  & factory?
  & foreign-name?
  & init-param*
}
```

Example: A JNDI symbolic link for a DataSource

```
<web-app xmlns="http://caucho.com/ns/resin"dd>
  <database jndi-name="jdbc/oracle">
    ...
  </database>

  <jndi-link jndi-name="java:comp/env/jdbc/gryffindor">
    <foreign-name>java:comp/env/jdbc/oracle</foreign-name>
  </jndi-link>

  <jndi-link jndi-name="java:comp/env/jdbc/slytherin">
    <foreign-name>java:comp/env/jdbc/oracle</foreign-name>
  </jndi-link>
</web-app>
```

Example: A JNDI foreign context for all EJB

```
<web-app xmlns="http://caucho.com/ns/resin">
  <jndi-link jndi-name='java:comp/env/ejb'>
```

```
    <factory>com.caucho.ejb.hessian.HessianContextFactory</factory>
    <init-param java.naming.provider.url='http://ejb.hogwarts.com:80/hessian'/>
  </jndi-link>
</web-app>
```

Example: A JNDI foreign context for selected EJB

```
<web-app xmlns="http://caucho.com/ns/resin">
  <jndi-link jndi-name='java:comp/env/remote-ejb'>
    <factory>com.caucho.ejb.hessian.HessianContextFactory</factory>
    <init-param java.naming.provider.url='http://ejb.hogwarts.com:80/hessian'/>
  </jndi-link>

  <jndi-link jndi-name="java:comp/env/ejb/Foo">
    <foreign-name>java:comp/env/remote-ejb/Foo</foreign-name>
  </jndi-link>

  <jndi-link jndi-name="java:comp/env/ejb/Bar">
    <foreign-name>java:comp/env/local-ejb/Bar</foreign-name>
  </jndi-link>
</web-app>
```

31.117 <jpa-persistence>

Table 31.93: <jpa-persistence> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
create-database-schema	If true, Amber will automatically create the database schema	false
cache-size	Size of the entity cache	32k
data-source	database used for JTA	
jdbc-isolation	JDBC isolation level used for connections	
read-data-source	Data source to be used for read-only queries	data-source
validate-database-schema	enables validation of the database tables on startup	false
xa-data-source	database to use in transactions	data-source

```
element jpa-persistence {
  create-database-schema?
  & cache-size?
  & cache-timeout?
  & data-source?
  & jdbc-isolation?
  & persistence-unit*
  & persistence-unit-default*
  & read-data-source?
  & validate-database-schema?
  & xa-data-source?
}

element persistence-unit {
  name
```

```
  & jta-data-source?
  & non-jta-data-source?
  & provider?
  & transaction-type?
  & properties?
}

element persistence-unit-default {
  & jta-data-source?
  & non-jta-data-source?
  & provider?
  & transaction-type?
  & properties?
}

element properties {
  element property {
     name
     & value
  }*
}
```

31.118 <jpa-persistence-unit>

Replacement for jpa-persistence.

31.119 <jsp>

Configures JSP behavior.

Table 31.94: <jsp> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
auto-compile	Automatically compile changed JSP files	true
deferred-syntax-allowed-as-literal	enables the #{...} syntax as text contents	true
dependency-check-interval	How often to check the jsp for changes, -1 disables	inherited
el-ignored	Ignore EL expressions in JSP text	false
fast-jstl	Optimize JSTL code generation	true
ignore-el-exception	Ignore exceptions generated in EL expressions. For debugging, set to false	true
is-xml	Default JSP pages to use XML syntax	false
page-encoding	Sets the default page encoding	ISO-8859-1
precompile	Try to load precompiled JSP pages	true
print-null-as-blank	If true, expressions evaluating to null are not printed	false
recompile-on-error	Recompile the JSP file when an Error occurs in loading	false
recycle-tags	Reuse tag instances when possible for performance	true
require-source	Return 404 when JSP source is deleted	false

Table 31.94: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
scriping-invalid	Disables all Java scripting and expression in JSP pages	false
session	Creates sessions for each JSP page	true
static-page-generates-class	If true, JSPs with no active content still generate a .class	true
tld-dir	restricts the directory to scan for .tld files, improving startup performance	WEB-INF
tld-file-set	adds an ant-style pattern for .tld scanning	WEB-INF
trim-directive-whitespace	if true, trims whitespace around JSP directives	false
validate-taglib-schema	if true, validate .tld files against the .tld schema. Set to false to handle invalid .tld files	true
velocity-enabled	if true, velocity-style tags are allowed	false
character-encoding	Sets JSP response character encoding; overrides character encoding defined at web-app level	

```
element jsp {
  auto-compile
  & deferred-syntax-allowed-as-literal?
  & dependency-check-interval?
  & el-ignored?
  & fast-jstl?
  & ide-hack?
  & ignore-el-exception?
  & is-xml?
  & page-encoding?
  & precompile?
  & print-null-as-blank?
  & recompile-on-error?
  & recycle-tags?
  & require-source?
  & scripting-invalid?
  & session?
  & static-page-generates-class?
  & tld-dir?
  & tld-file-set?
  & trim-directive-whitespaces?
  & validate-taglib-schema?
  & velocity-enabled?
}
```

31.120 <jsp-config>

<jsp-config> configure standard settings for JSP files.

Table 31.95: <jsp-config> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
url-pattern	selects the URLs which this jsp-config applies to	
el-ignored	If true, EL expressions are ignored	false
page-encoding	Defines the default page encoding for the JSP file	ISO-8859-1
scripting-invalid	If true, Java scripting is forbidded in the JSP page	false
trim-directive-whitespaces	If true, extra whitespace is trimmed around JSP directives	false
is-xml	If true, for XML syntax for JSP pages	false
include-prelude	Includes JSP fragments before the JSP page as headers	
include-coda	Includes JSP fragments before the JSP page as footers	

```
element jsp-config {
  taglib*,
  jsp-property-group*
}

element jsp-property-group {
  url-pattern*,
  deferred-syntax-allowed-as-literal?,
  el-ignored?,
  page-encoding?
  scripting-invalid?
  trim-directive-whitespaces?
  is-xml?
  include-prelude*
  include-coda*
}
```

31.121 <jvm-arg>

<jvm-arg> configures JVM arguments to be passed to Resin on the command line, typically -X memory parameters and -D defines.

standard jvm-args

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <jvm-arg>-Xmx512m</jvm-arg>
      <jvm-arg>-Xss1m</jvm-arg>
      <jvm-arg>-verbosegc</jvm-arg>
    </server-default>

    <server id="app-a" address="192.168.2.10"/>

    ...
  </cluster>
</resin>
```

31.122 <jvm-classpath>

<jvm-classpath> adds a classpath entry when starting the JVM.

adding a classpath

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <jvm-classpath>/tmp/test-classpath;/jvm-classpath>
    </server-default>

    <server id="app-a" address="192.168.2.10"/>

    ...
  </cluster>
</resin>
```

31.123 <keepalive-max>

<keepalive-max> configures the maximum number of sockets which can be used directly for connections. In Resin Professional, the allows for a much larger number of keepalive sockets, since it can detach threads from connections. Without the select manager, each connection is associated with a thread.

A value of -1 disables keepalives.

Keepalives are an important TCP technique used with HTTP and Resin's load-balancing to avoid the heavy network cost of creating a new socket. Since an initial HTTP request is usually immediately followed by extra requests to load files like images and stylesheets, it's generally more efficient to keep the socket open for a short time instead of creating a new one. The socket keepalive is even more important for Resin's load balancing, to avoid creating extra sockets between the web-tier and the app-tier and to make distributed sessions more efficient.

Higher values of <keepalive-max> improve network efficiency but increase the number of threads waiting for new client data.

keepalive-thread-max in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <http port="80"/>

      <thread-max>512</thread-max>

      <keepalive-max>100</keepalive-max>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.124 <keepalive-select-enable>

<keepalive-select-enable> enables the select manager for keepalives. The select manager is a Resin Professional feature allowing more keepalives by detaching threads from sockets.

Normally, this should be left enabled.

31.125 <keepalive-select-thread-timeout>

<keepalive-select-thread-timeout> is a short timeout allowing the select manager to wait for a keepalive before detaching the thread. This value would not normally be changed.

31.126 <keepalive-timeout>

<keepalive-timeout> configures how long a keepalive connection should wait for a new request before closing.

Keepalives are used both for HTTP connections and for load-balancing and clustering connections. HTTP connections generally have a single HTML page, followed by a number of image requests. By using keepalives, all the requests can use a single socket. The <keepalive-timeout> should be long enough to catch all the HTTP burst requests, but can close after the burst is complete. A value of 5s or 15s is generally sufficient.

The load-balancing and clustering keepalives have a different timeout behavior. Since load-balancing sockets are reused for multiple clients, they can have longer timeouts.

keepalive-thread-max in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <http port="80"/>

      <thread-max>512</thread-max>

      <keepalive-max>100</keepalive-max>
      <keepalive-timeout>15s</keepalive-timeout>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.127 <lazy-servlet-validate>

<lazy-servlet-validate> defers validation of servlet classes until the servlet is used. Some servlet libraries are bundled with web.xml files which include servlets with no available classes. Since Resin will normally send an error in this situation, <lazy-servlet-validate> lets you turn the validation off.

```
element lazy-servlet-validate {
  r_boolean-Type
}
```

31.128 <library-loader>

<library-loader> configures a jar library, WEB-INF/lib -style class loader.

The library-loader will add jar files in its path to the current classpath. Jar files are recognized wihen they have a filename extension of `.jar` or `.zip` .

See DirectoryLoader .

Table 31.96: <library-loader> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
fileset	An ant-style fileset	
path	Filesystem path for the class loader. Since Resin 3.0	required

```
element library-loader {
  fileset
  | path
}

element fileset {
  dir
  & exclude*
  & include*
}
```

31.129 <listener>

<listener> configures servlet event listeners. The listeners are registered based on interfaces they implement. The listener instances are fully admin/config-candi.xtp aware, including dependency injection, observing events, and supporting aspects.

Table 31.97: listener interfaces

INTERFACE	DESCRIPTION
javax.servlet.ServletContextListener	Called when the web-app starts and stops
javax.servlet.ServletContextAttributeListener	Called when the web-app attributes change
javax.servlet.ServletRequestListener	Called when the request starts and stops
javax.servlet.ServletRequestAttributeListener	Called when request attributes change
javax.servlet.http.HttpSessionListener	Called when HTTP sessions start or stop
javax.servlet.http.HttpSessionAttributeListener	Called when HTTP sessions attributes change
javax.servlet.http.HttpSessionActivationListener	Called when HTTP sessions passivate or activate

Table 31.98: <listener> Attributes

ATTRIBUTE	DESCRIPTION
listener-class	classname of the listener implementation
init	IoC initialization of the listener

```
element listener {
  listener-class,
  init?
}
```

31.130 <load-balance-connect-timeout>

<load-balance-connect-timeout> configures the maximum time a client connection to a cluster-port should take. The admin/clustering-overview.xtp and deploy-ref.xtp#session-config use load-balance-connect-timeout to connect to backend or peer servers in the cluster.

Lower values detect failed servers more quickly, but a too-low value can timeout too quickly for a live server with some network congestion.

load-balance-connect-timeout

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <server-default>
          <load-balance-connect-timeout>2s</load-balance-connect-timeout>
        </server-default>

        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.11" port="6800"/>

        <host id="">
          ...
    </cluster>
</resin>
```

31.131 <load-balance-recover-time>

<load-balance-recover-time> is the maximum time the admin/clustering-overview.xtp will consider the server dead after a failure before retrying the connection.

Resin uses the load-balance-recover-time to avoid wasting time trying to connect to an unavailable app-tier server.

Lower values let the load balancer use a restarted server faster, but lower values also increase the overhead of trying to contact unavailable servers.

load-balance-recover-time

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <server-default>
          <load-balance-recover-time>10s</load-balance-recover-time>
        </server-default>

        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.11" port="6800"/>

        <host id="">
          ...
    </cluster>
</resin>
```

31.132 <load-balance-idle-time>

<load-balance-idle-time> is the maximum time the admin/clustering-overview.xtp and deploy-ref.xtp#session-config will leave an idle socket before closing it.

The default value is normally sufficient, since it tracks the keepalive of the cluster port.

load-balance-idle-time must be less than the keepalive value of the target cluster-port .

The load balancer and distributed sessions reuse sockets to the cluster peer and servers to improve TCP performance. The load-balance-idle-time limits the amount of time those sockets can remain idle.

Higher values may improve the socket pooling, but may also increase the chance of connecting to a closed server.

31.133 <load-balance-warmup-time>

The time the admin/clustering-overview.xtp uses to throttle connections to an server that's just starting up.

Java web-applications often start slowly while they initialize caches. So a newly-started application will often be slower and consume more resources than a long-running application. The warmup-time increases Resin's reliability by limiting the number of requests to a new app-tier server until the server has warmed up.

Larger values give the application a longer time to warm up.

load-balance-warmup-time

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <server-default>
          <load-balance-warmup-time>60s</load-balance-warmup-time>
        </server-default>

        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.11" port="6800"/>

        <host id="">
          ...
    </cluster>
</resin>
```

31.134 <load-balance-weight>

load-balance-weight assigns a load-balance weight to a backend server. Servers with higher values get more requests. Servers with lower values get fewer requests.

In some cases, some servers may be more powerful than others. load-balance-weight lets the load-balancer assign more connections to the more powerful machines.

Test and profiling servers can also use load-balance-weight to receive a small number of connections for profiling purposes.

Larger values tell the load-balancer to assign more requests to the app-tier server.

load-balance-weight

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.10" port="6800"/>

        <server id="test" address="192.168.0.100" port="6800">
            <load-balance-weight>1</load-balance-weight>
        </server>

        <host id="">
          ...
    </cluster>
</resin>
```

31.135 <login-config>

Configures the login method for authentication, one of BASIC, DIGEST or FORM.

See also: admin/security-overview.xtp for an overview.

Table 31.99: <login-config> Attributes

ATTRIBUTE	DESCRIPTION
auth-method	Authentication method, either BASIC for HTTP Basic Authentication, FORM for form based authentication, or DIGEST for HTTP admin/security-overview.xtp .
authenticator	Specifies the authenticator to use to lookup users and passwords.
class	Defines a custom class which extends http://caucho.com/resin-javadoc/com/caucho/server/-security/AbstractLogin.html
form-login-config	Configuration for form login.
init	Initialization for the custom login class
realm-name	The realm name to use in HTTP authentication

HTTP Authentication is defined in the RFC http://www.faqs.org/rfcs/rfc2617.html .

HTTP digest authentication is discussed in admin/security-overview.xtp .

```
element login-config {
  class?
  & auth-method?
  & authenticator?
  & form-login-config?
  & init?
  & realm-name?
```

31.136 <log-handler>

Configure a log handler for the JDK java.util.logging.* API. java.util.logging has two steps: configure a set of log handlers, and configure the levels for each logger. The <log-handler> creates a destination for logs, sets a minimum logging level for the handler, and attaches the handler to a logging name.

In addition to configuring custom handlers, <log-handler> has the most common configuration build-in: logging to a rotating file. Most of the configuration attributes are used for the rotating file and are shared with the other logging configuration.

Table 31.100: <log-handler> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-format	the format for the archive filename when a rollover occurs, see Rollovers .	see below
class	configures a custom Handler class	
format	An EL expression string to format the current output line.	
formatter	Configures a custom java.util.logging.Formatter to format the output.	

Table 31.100: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
level	The log level for the handler. Typically, the handler's level will be finer than the logger's level	info
mbean-name	an mbean name, see MBean control .	no mbean name, no mbean registration
name	A hierarchical name, typically aligned with the Java packaging names. The handler will be registered with the Logger with the matching name.	match all names
path	Output path for the log messages, see "Log Paths"	required
path-format	Selects a format for generating path names. The syntax is the same as for archive-format	optional
timestamp	a timestamp format string to use at the beginning of each log line.	"[%Y/%m/%d %H:%M:%S.%s] "
rollover-count	maximum number of rollover files before the oldest ones get overwritten. See Rollovers .	none
rollover-cron	cron-style specification on rollover times.	none
rollover-period	how often to rollover the log. Specify in days (15D), weeks (2W), months (1M), or hours (1h). See Rollovers .	none
rollover-size	maximum size of the file before a rollover occurs, in bytes (50000), kb (128kb), or megabytes (10mb). See Rollovers .	1mb

```
element log-handler {
  archive-format?
  & class?
  & filter?
  & format?
  & formatter?
  & level?
  & mbean-name?
  & name
  & path?
  & path-format?
  & rollover-count?
  & rollover-period?
  & rollover-size?
  & timestamp?
  & use-parent-handlers?
}
```

The following example is a standard log handler writing to a rollover file. Because the handler's level is "all", the <logger> configuration will set the actual logging level.

Example: logging to a rollover file

```
<web-app xmlns="http://caucho.com/ns/resin">

  <log-handler name="" level="all"
```

```
       timestamp="[%Y/%m/%d %H:%M:%S.%s] {%{thread}} "/>

  <logger name="com.caucho" level="info"/>

</web-app>
```

31.137 <logger>

<log> configures JDK 1.4 java.util.logger Logger level.

The admin/logging.xtp describes log in detail.

Table 31.101: <logger> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
level	the java.util.logging level: finest, finer, fine, config, info, warning, severe	info
name	the java.util.logging name, typically a classname	required
use-parent-handlers	if true, parent handlers are also invoked	true

```
element logger {
  name
  & level?
  & use-parent-handlers?
}
```

Example: compilation logging

```
<resin xmlns="http://caucho.com/ns/resin">
  <log name="" level="all" path="log/debug.log"/>
  <logger name="com.caucho.java" level="fine"/>

  <cluster id="app-tier">
    ...
  </cluster>
</resin>
```

Example: logging to a JMS queue

```
<web-app xmlns="http://caucho.com/ns/resin"
    xmlns:resin="urn:java:com.caucho.resin">

  <resin:MemoryQueue ee:Named="myQueue"/>

  <logger name="qa.test">
    <resin:JmsLogHandler level="warning">
      <target>${myQueue}</target>

      <resin:TimestampLogFormatter/>
    </resin:JmsLogHandler>
  </logger>

</web-app>
```

31.138 Log format string

The format for log tags is used to specify a format string for each log message. `format` recognizes EL-expressions. The EL variable log is a com.caucho.log.ELFormatter.ELFormatterLogRecord object.

log format string

```
<log name='' level='all' path='stderr:' timestamp="[%H:%M:%S.%s]"
     format=" ${log.level} ${log.loggerName} ${log.message}"/>
```

Table 31.102: log EL variable *log* is a LogRecord

ACCESSOR	VALUE
${log.level}	The level of the log record
${log.name}	The source loggers name
${log.shortName}	A shorter version of the source loggers name, "Foo" instead of "com.hogwarts.Foo"
${log.message}	The message, with no formatting or localization
${log.millis}	event time in milliseconds since 1970
${log.sourceClassName}	Get the name of the class that issued the logging request (may not be available at runtime)
${log.sourceMethodName}	Get the name of the method that issued the logging request (may not be available at runtime)
${log.threadID}	Get an int identifier of the thread where the logging request originated
${log.thrown}	Get any java.lang.Throwable associated with the logging request

You can also use the admin/config-el.xtp in your format string:

log format string using an Environment EL variable.

```
<host ...>

  <web-app>
    <log name='' level='all' path='log/debug.log' timestamp="[%H:%M:%S.%s]"
         format=" [ ${app.contextPath} ] ${log.message}"/>

    ...
  </web-app>

  ...

</host>
```

```
[14:55:10.189] [/foo] 'null' returning JNDI java:
       model for EnvironmentClassLoader[web-app:http://localhost:8080/foo]
[14:55:10.189] [/foo] JNDI lookup 'java:comp/env/caucho/auth'
       exception javax.naming.NameNotFoundException: java:comp/env/caucho/auth
[14:55:10.199] [/foo] Application[http://localhost:8080/foo] starting
```

The config-el.xtp#fmt.sprintf function can space pad the values and make the results look a little nicer:

fmt.sprintf() in log format string

```
<log name='' level='all' path='stderr:' timestamp="[%H:%M:%S.%s]"
     format=" ${fmt.sprintf('%-7s %45s %s',log.level,log.loggerName,log.message)}"/>
```

```
[14:28:08.137] INFO com.caucho.vfs.QJniServerSocket Loaded Socket JNI library.
[14:28:08.137] INFO com.caucho.server.port.Port http listening to *:8080
[14:28:08.137] INFO com.caucho.server.resin.ServletServer ServletServer[] starting
[14:28:08.307] INFO com.caucho.server.port.Port hmux listening to localhost:6802
[14:28:08.437] INFO com.caucho.server.host.Host Host[] starting
```

config-el.xtp#fmt.sprintf and config-el.xtp#fmt.timestamp can be used to produce CSV files:

CSV log files

```
<log name='' level='all' path='log/debug.csv' timestamp=""
     format="${fmt.sprintf('%vs,%d,%d,%vs,%vs',fmt.timestamp('%Y-%m-%d %H:%M:%S.%s'),
               log.threadID, log.level.intLevel(), log.loggerName, log.message)}"/>
```

```
"2003-11-17 14:46:14.529",10,800,"com.caucho.vfs.QJniServerSocket",
            "Loaded Socket JNI library."
"2003-11-17 14:46:14.549",10,800,"com.caucho.server.port.Port",
            "http listening to *:8080"
"2003-11-17 14:46:14.549",10,800,"com.caucho.server.resin.ServletServer",
            "ServletServer[] starting"
"2003-11-17 14:46:14.719",10,800,"com.caucho.server.port.Port",
            "hmux listening to localhost:6802"
"2003-11-17 14:46:14.850",10,800,"com.caucho.server.host.Host",
            "Host[] starting"
"2003-11-17 14:46:15.100",10,800,"com.caucho.server.webapp.Application",
            "Application[http://localhost:8080/freelistbm] starting"
```

31.139 <mail>

<mail> configures a javax.mail.Session object and makes it available in Resin-IoC/WebBeans. Mail properties can be configured using the properties attribute. Some of the most common properties can be configured directly on the <mail> tag.

Table 31.103: <mail> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
authenticator	sets a custom javamail authenticator, either with EL or a custom resin:type bean	
debug	sets the mail.debug flag	
from	sets the mail.from property	
host	sets the mail.host property	
imap-host	sets the mail.imap.host property	
imap-port	sets the mail.imap.port property	
imap-user	sets the mail.imap.user property	
init	IoC configuration for other properties	
jndi-name	JNDI name to store the mail Session	
name	Resin-IoC/WebBeans @Named value	
password	sets the password for authentication	
pop3-host	sets the mail.pop3.host property	
pop3-port	sets the mail.pop3.port property	
pop3-user	sets the mail.pop3.user property	
properties	general mail properties in property file format	

Table 31.103: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
smtp-host	sets the mail.smtp.host property	
smtp-port	sets the mail.smtp.port property	
smtp-user	sets the mail.smtp.user property	
store-protocol	sets the mail.store.protocol property	
transport-protocol	sets the mail.transport.protocol property	
user	sets the mail.user property	

```
element mail {
  authenticator?
  & debug?
  & from?
  & host?
  & imap-host?
  & imap-port?
  & imap-user?
  & init?
  & jndi-name?
  & name?
  & pop3-host?
  & pop3-port?
  & pop3-user?
  & smtp-host?
  & smtp-port?
  & smtp-user?
  & store-protocol?
  & transport-protocol?
  & user?
}
```

Example: mail

```
<web-app xmlns="http://caucho.com/ns/resin">

  <mail jndi-name="java:comp/env/mail">
    <from>noreply@foo.com</from>
    <smtp-host>localhost</smtp-host>
    <smtp-port>25</smtp-port>

    <properties>
      mail.smtp.starttls.enable=true
    </properties>
  </mail>
</web-app>
```

31.140 <max-active-time>

<max-active-time> configures the maximum time a connection can be active before Resin will automatically close it. Normally, the max-active-time should not be configured, since Resin will also automatically close a connection at the end of a request.

Sites should generally leave max-active-time at the default.

31.141 <max-close-statements>

<max-close-statements> configures how many open statements Resin should save to for the connection close. Since the JDBC

`Connection.close()` call automatically closes any open statements, Resin's database pool needs to keep track of any open statements to close them in case the application has forgotten. The <max-close-statements> is primarily needed for older database drivers implementing the `java.sql.Driver` interface.

31.142 <max-connections>

<max-connections> configures the maximum number of open connections allowed for Resin's database pool. Sites can use <max-connections> to throttle the number of database connections for an overloaded server. When `max-connections`

is reached and an application calls `getConnection` , Resin will wait connection-wait-time or until a connection is freed before allocating a new connection.

31.143 <max-create-connections>

<max-create-connections> configures the maximum number of simultaneous connection creations. Since connection creation is slow and database access can be spiky, Resin's pool limits the number of new connections to the database at any time. Once a connection has succeeded, a new connection can proceed.

31.144 <max-idle-time>

<max-idle-time> configures the maximum time a connection can remain idle before Resin automatically closes it. Since idle databases tie up resources, Resin will slowly close idle connections that are no longer needed.

Higher values of <max-idle-time> will connections to remain in the idle pool for a longer time. Lower values will close idle connections more quickly.

31.145 <max-overflow-connections>

<max-overflow-connections> extends <connection-max> temporarily in case of overflow. After the <connection-wait-time> expires, Resin can create an overflow connection to handle unforseen load spikes.

31.146 <max-pool-time>

<max-pool-time> configures the maximum time the connection can remain open. A connection could theoretically remain open, switching between active and idle, for an indefinite time. The <max-pool-time> allows a site to limit to total time of that connection.

Most sites will leave <max-pool-time> at the default.

31.147 <max-uri-length>

Sets limit on longest URIs that can be served by Resin.

```
element max-uri-length {
  r_int-Type
}
```

31.148 <memory-free-min>

<memory-free-min> improves server reliability by detecting low-memory situations caused by memory leaks and forcing a clean server restart. Since Resin's service reliably restarts the server, a website can improve stability by forcing a restart before memory becomes a major problem. The memory-free-min restart will also log a warning, notifying the developers that a potential memory leak needs to be resolved.

When free heap memory gets very low, the garbage collector can run continually trying to free up extra memory. This continual garbage collection can send the CPU time to 100%, cause the site to become completely unresponsive, and yet take a long time before finally failing to an out of memory error (forcing an unclean restart). To avoid this situation, Resin will detect the low-memory condition and gracefully restart the server when free memory becomes too low.

The ultimate solution to any memory leak issues is to get a memory profiler, find the leaking memory and fix it. <memory-free-min> is just a temporary bandage to keep the site running reliably until the memory leak can be found and fixed.

memory-free-min resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <thread-max>512</thread-max>

      <memory-free-min>1m</memory-free-min>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.149 <mime-mapping>

Maps url patterns to mime-types.

Table 31.104: <mime-mapping> Attributes

ATTRIBUTE	DESCRIPTION
extension	url extension
mime-type	the mime-type

```
element mime-mapping {
  extension,
  mime-type
}
```

Example: WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

  <mime-mapping>
    <extension>.foo</extension>
    <mime-type>text/html</mime-type>
  </mime-mapping>

  <!-- resin shortcut syntax -->
```

```
  <mime-mapping extension='.bar'
                mime-type='text/html'/>

</web-app>
```

Resin has a long list of default mime types in $RESIN_HOME/conf/app-default.xml

31.150 <multipart-form>

Enables multipart-mime for forms and file uploads. multipart-mime is disabled by default.

For an uploaded file with a form name of foo , the parameter value contains the path name to a temporary file containing the uploaded file. foo.filename contains the uploaded filename, and foo.content-type contains the content-type of the uploaded file.

Table 31.105: <multipart-form> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
upload-max	maximum size of an upload request (in kb).	no limit

If the upload is larger than the limit or if multipart-form processing is disabled, Resin will not parse the request and will set an error message in the " caucho.multipart.form.error " request attribute. The " caucho.multipart.form.error.size " will contain the attempted upload size.

Requests can set the maximum by setting the request attribute " caucho.multipart.form.upload-max " with an Integer or Long value.

By default, multipart-form is disabled.

```
element multipart-form {
  enable?
  & upload-max?
}
```

31.151 <path-mapping>

Maps url patterns to real paths. If using a server like IIS, you may need to match the server's path aliases.

Table 31.106: <path-mapping> Attributes

ATTRIBUTE	DESCRIPTION
url-pattern	A pattern matching the url: /foo/* , /foo , or *.foo
url-regexp	A regular expression matching the portion of the url that follows the
real-path	The prefix of the real path. When used with url-regexp , allows substitution variables like $1 .

```
element path-mapping {
```

```
  (url-pattern | url-regexp)

  & real-path
}
```

Example: resin-web.xml aliasing paths

```
<web-app xmlns="http://caucho.com/ns/resin">

<path-mapping url-pattern='/resin/*'
              real-path='e:\resin'/>

<path-mapping url-regexp='/~([^/]*)'
              real-path='e:\home\$1'/>

</web-app>
```

31.152 <password>

<password> configures the database connection password. Sites requiring additional security for their passwords can use the <mypkg:MyDecryptor/> syntax to configure a password decoder.

31.153 <ping>

Starts a thread that periodically makes a request to the server, and restarts Resin if it fails. This facility is used to increase server reliability - if there is a problem with the server (perhaps from a deadlock or an exhaustion of resources), the server is restarted.

A failure occurs if a request to the url returns an HTTP status that is not 200.

Since the local process is restarted, it does not make sense to specify a url that does not get serviced by the instance of Resin that has the ping configuration. Most configurations use url's that specify *localhost* as the host.

This pinging only catches some problems because it's running in the same process as Resin itself. If the entire JDK freezes, this thread will freeze as well. Assuming the JDK doesn't freeze, the PingThread will catch errors like deadlocks.

Table 31.107: <ping> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
url	A url to ping.	required
sleep-time	Time to wait between pings. The first ping is always 15m after the server starts, this is for subsequent pings.	15m
try-count	If a ping fails, number of times to retry before giving up and restarting	required
freeze-timeout	Time with no response to detect a frozen jvm.	15m
retry-time	time between retries	1s
socket-timeout	time to wait for server to start responding to the tcp connection before giving up	10s

Example: resin.xml - simple usage of server ping

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
    <cluster id="app-tier">
        <ping url="http://localhost/"/>
        ...
    </cluster>
</resin>
```

Example: resin.xml - configured usage of server ping

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
    ...
    <cluster id="app-tier">
        <ping>
            <url>http://localhost:8080/index.jsp</url>
            <url>http://localhost:8080/webapp/index.jsp</url>
            <url>http://virtualhost/index.jsp</url>
            <url>http://localhost:443/index.jsp</url>

            <sleep-time>5m</sleep-time>
            <try-count>5</try-count>

            <!-- a very busy server -->
            <socket-timeout>30s</socket-timeout>
        </ping>
      ...
    </cluster>
</resin>
```

The class that corresponds to <ping> is PingThread .

31.154 Mail notification when ping fails

A refinement of the ping facility sends an email when the server is restarted.

resin.xml - mail notification when ping fails

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">
  ...
  <cluster id="web-tier">
    <ping resin:type="com.caucho.server.admin.PingMailer">
      <url>http://localhost:8080/index.jsp</url>
      <url>http://localhost:8080/webapp/index.jsp</url>

      <mail-to>fred@hogwarts.com</mail-to>
      <mail-from>resin@hogwarts.com</mail-from>
      <mail-subject>Resin ping has failed for server ${'${'}server.name}</mail-subject>
    </ping>
    ...
  </server>
</resin>
```

The default behaviour for sending mail is to contact a SMTP server at host 127.0.0.1 (the localhost) on port 25. System properties are used to configure a different SMTP server.

resin.xml - smtp server configuration

```
  <system-property mail.smtp.host="127.0.0.1"/>
  <system-property mail.smtp.port="25"/>
```

31.155 <ping>

<ping> enables connection validation. When <ping> is enabled, Resin will test the connection with <ping-query> or <ping-table> before returning a connection to the user. If the connection fails the test, Resin will close it and return a new connection.

For efficiency, Resin will only validate the connection if it has been idle for longer than <ping-interval> .

31.156 <ping-table>

<ping-table> configures the database table Resin should use to verify if a connection is still valid when returned from the pool.

31.157 <ping-query>

<ping-query> specifies the query to use for validating if a database connection is still valid when returned from the idle pool.

31.158 <ping-interval>

<ping-interval> configures when Resin should validate an idle connection. Connections which have been idle for less than <ping-interval> are assumed to be still valid without validation. Connections idle for longer than <ping-interval> are validated.

Sites can force a validation by setting <ping-interval> to 0.

31.159 <port>

The server <port> defines the TCP port for Resin cluster communication and load balancing. Most server instances will use a common port like 6800, while machines with multiple servers may use multiple ports like 6800 and 6801.

multiple servers on a machine

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <server id="app-a" address="192.168.1.11" port="6800"/>
    <server id="app-b" address="192.168.1.11" port="6801"/>

    <server id="app-b" address="192.168.2.12" port="6800"/>
    <server id="app-c" address="192.168.2.12" port="6801"/>

    ...
  </cluster>
</resin>
```

31.160 <port-default>

Defines default port parameters for all <http>, <protocol>, and <cluster-port>.

31.161 <prepared-statement-cache-size>

<prepared-statement-cache-size> configures how many prepared statements Resin should cache for each connection. Caching prepared statement can improve performance for some database drivers by avoiding repeated parsing of the query SQL.

31.162 <protocol>

<protocol> configures a remoting protocol for a Java bean. The bean is configured with the <servlet> and <servlet-mapping> tags, since it will process HTTP URL requests.

Protocol drivers extend the com.caucho.remote.server.AbstractProtocolServletFactory interface and can register a URI alias to simplify configuration.

Table 31.108: <protocol> Attributes

ATTRIBUTE	DESCRIPTION
class	Classname of the protocol driver implementing ProtocolServletFactory
init	Optional IoC initialization for the protocol driver
uri	Protocol configuration shortcut

Table 31.109: Current drivers

URI SCHEME	DESCRIPTION
burlap:	The burlap XML protocol
cxf:	The CXF SOAP implementation
hessian:	The Hessian protocol
xfire:	The XFire SOAP implementation

```
element protocol {
  (class | uri)
  & init?
}
```

31.163 <redeploy-check-interval>

<redeploy-check-interval> specifies how often Resin should check if a .war file has been updated or added to a <web-app-deploy> directory.

```
element redeploy-check-interval {
  r_period-Type
}
```

31.164 <redeploy-mode>

<redeploy-mode> specifies how Resin handles updates to web-apps and .war files. By default, Resin restarts web-apps when classes or configuration files change.

Table 31.110: <redeploy-mode> Attributes

MODE	DESCRIPTION
automatic	checks for redeployment and auto-redeploy if modified
manual	does not check for redeployment. Only checks if manual (JMX)

```
element redeploy-mode {
  automatic | manual
}
```

31.165 <rewrite-real-path>

<rewrite-real-path> lets you rewrite the URL to physical path mapping, to allow aliasing or for filesystem organization.

Table 31.111: <rewrite-real-path> Attributes

ATTRIBUTE	DESCRIPTION
real-path	specifies the URL to real-path mapping
regexp	a regular expression matching a URL
replacement	specifies a replacement pattern for URL to URL rewriting
rewrite	rewrites a URL to another URL as a preprocessing-step
target	specifies the target for URL to real-path mapping

```
element rewrite-real-path {
  element rewrite {
    regexp
    & replacement
  }*

  & element rewrite {
    regexp
    & target
  }*
}
```

Example: aliasing /foo to /bar

```
<web-app xmlns="http://caucho.com/ns/resin">

  <rewrite-real-path>
    <real-path regexp="^/foo" target="/bar"/>
  </rewrite-real-path>

</web-app>
```

31.166 <rewrite-vary-as-private>

Because not all browsers understand the Vary header, Resin can rewrite Vary to a Cache-Control: private. This rewriting will cache the page with the Vary in Resin's proxy cache, and also cache the page in the browser. Any other proxy caches, however, will not be able to cache the page.

The underlying issue is a limitation of browsers such as IE. When IE sees a Vary header it doesn't understand, it marks the page as uncacheable. Since IE only understands "Vary: User-Agent", this would mean IE would refuse to cache gzipped pages or "Vary: Cookie" pages.

With the <rewrite-vary-as-private> tag, IE will cache the page since it's rewritten as "Cache-Control: private" with no Vary at all. Resin will continue to cache the page as normal.

31.167 <root-directory>

<root-directory> specifies the base directory for the contexts. All EL-style directory paths are relative to the root-directory.

```
element root-directory {
  r_path-Type
}
```

Example: cluster root-directory

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <root-directory>/var/resin/app-tier</root-directory>

        <server id="a" ...>

        <host host-name="www.foo.com">
    </cluster>
</resin>
```

31.168 <reference>

<reference> configures a JNDI ObjectFactory. Some legacy resources are configured using an ObjectFactory syntax. The <reference> tag provides a compatible way to configure those objects. More modern resources should use <bean> or <component> for IoC configuration.

JNDI ObjectFactories are used to create objects from JNDI references. The <reference> tag configures the ObjectFactory and stores it in JNDI.

Table 31.112: <reference> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
jndi-name	JNDI name for the reference. Since Resin 3.0	required
factory	Class name of the ObjectFactory. Resin 3.0	required
init	Bean-style initialization for the factory	none

```
element reference {
```

```
  factory
  & jndi-name
  & init-param*
}
```

Example: Hessian client reference

```
<web-app xmlns="http://caucho.com/ns/resin">

<reference>
  <jndi-name>hessian/hello</jndi-name>
  <factory>com.caucho.hessian.client.HessianProxyFactory</factory>
  <init url="http://localhost:8080/ejb/hello"/>
        type="test.HelloHome"/>
</reference>

</web-app>
```

31.169 <remote-client>

<remote-client> configures a proxy to a web-service. It uses a Java interface and a URI to select the web-service.

The URI is defined as: protocol:url=location , where location is typically a HTTP URL. See http://wiki4.caucho.com/Does_Resin_4_Sup for more information, including how to write an adapter for Resin remoting.

Table 31.113: <remote-client> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
class	Class name of the protocol implementation	required (or uri)
init	IoC initialization for the protocol implementation	
name	admin/config-candi.xtp#@Named binding for Resin-IoC	
jndi-name	JNDI binding name	
uri	Shortcut alias name for the protocol class	

Table 31.114: remote-client protocols

URI	DESCRIPTION
cxf :url=http://foo.com/hello/cxf	Defines a cxf service. See http://wiki.caucho.com/CXF for more information.
burlap :url=http://foo.com/hello/burlap	Defines a burlap service at http://foo.com/hello/burlap
hessian :url=http://foo.com/hello/hessian	Defines a hessian service at http://foo.com/hello/hessian
xfire :url=http://foo.com/hello/cxf	Defines a xfire client. See http://wiki.caucho.com/XFire for more information.

```
element remote-client {
  (class|uri)
```

```
  & name?
  & jndi-name?
  & interface
}
```

31.170 <resin>

<resin> is the top-level configuration tag for the resin.xml file. The <resin> tag needs to specify the Resin namespace, to allow for validation of the configuration file.

The environment of the top-level <resin> is the global classpath. This environment can be important for <log> configuration for threads which run with using only the global class loader. Because the <resin> environment does not have a dynamic class loader, dynamically loaded classes, like custom jars in resin/lib are not available.

```
element resin {
  env resources
  & cluster*
  & cluster-default*
  & environment-system-properties?
  & management?
  & min-free-memory?
  & root-directory?
  & security-manager?
  & security-provider?
  & watchdog-manager?
}
```

Example: minimal resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
  <root-directory>/var/resin</root-directory>

  <cluster id="web-tier">
    <server id="">
      <http address="*" port="8080"/>
    </server>

    <resin:import path="app-default.xml"/>

    <host id="">
      <web-app id="" root-directory="/var/resin/htdocs"/>
    </host>
  </cluster>
</resin>
```

31.171 <resin-system-auth-key>

<resin-system-auth-key> specifies authorization token for Resin's Security service.

```
element resin-system-auth-key {
  String
}
```

31.172 <resin:AdminAuthenticator>

Authenticator used by Resin for administration purposes such as the /resin-admin application. Also used by the watchdog and clustering code to authenticate servers, so changing principals or passwords in this authenticator requires a shutdown and restart of the watchdog and Resin instances.

Uses the same syntax as XmlAuthenticator.

Table 31.115: <resin:AdminAuthenticator> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
user	Specifies a user authentication record. There maybe zero, one or more records.	None
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64
path	Specifies the path to an XML file containing users and passwords.	None
logout-on-session-timeout	If true, the user will be logged out when the session times out.	true

31.173 <resin:Allow>

The <resin:Allow> tag is used to secure a particular URL pattern. Because it is affirmative, it must always include a nested condition expressing an authorization constraint. All access attempts that do not satisfy the authorization rule are denied access. This tag is the most common type of top level authorization tag.

Table 31.116: <resin:Allow> Attributes

ATTRIBUTE	DESCRIPTION
url-pattern	URL pattern describing the resource to be secured.
http-method	HTTP methods that the restriction applies to.

Protecting all pages for logged-in users

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:Allow url-pattern="/*">
    <resin:IfUserInRole role="user"/>
  </resin:Allow>
  ...
</web-app>
```

31.174 <resin:And>

Matches if all children match.

Matching both of two query parameters

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="" target="/match">
    <resin:And>
      <resin:IfQueryParam name="foo" regexp="foo"/>
      <resin:IfQueryParam name="bar" regexp="bar"/>
    </resin:And>
  </resin:Redirect>

</web-app>
```

31.175 <resin:BasicLogin>

As the name implies, HTTP basic authentication is the simplest mechanism for gathering login data for web applications. When a web page is secured through HTTP basic authentication, the brower renders a simple dialog box for the user to enter login information. This information is then sent to the server in clear-text using well-defined HTTP headers. This authentication mechanism can be convenient for quick protection of internal pages or administration when writing a form isn't necessary. If you use basic authentication for applications outside the fire-wall, it is highly recommended that you secure the transport layer using admin/security-ssl.xtp . The <resin:BasicLogin> tag is used to configure basic authentication.

WEB-INF/resin-web.xml resin:BasicLogin

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:BasicLogin/>

  <resin:Allow url-pattern="/foo/*">
     <resin:IfUserInRole role="user"/>
  </resin:Allow>

  <resin:XmlAuthenticator>
     ...
  </resin:XmlAuthenticator>

</web-app>
```

31.176 <resin:ByteStreamCache>

Distributed cache of InputStream/OutputStream byte streams across a cluster pod.

31.177 <resin:choose>

resin:choose implements an if, elsif, else.

The <resin:choose> schema is context-dependent. A <resin:choose> in a <web-app> will have <web-app> content, while a <resin:choose> in a <host> will have <host> content.

Table 31.117: <resin:choose> Attributes

ATTRIBUTE	DESCRIPTION
resin:when	A configuration section executed when matching a test condition
resin:otherwise	A fallback section executed when the tests fail

```
element resin:choose {
  resin:when*,
  resin:otherwise
}

element resin:when {
  attribute test { string },

  context-dependent content
}

element resin:otherwise {
  context-dependent content
}
```

Example: resin:choose usage pattern

```
<resin:choose>
  <resin:when test="${expr1}">
    ...
  </resin:when>

  <resin:when test="${expr2}">
    ...
  </resin:when>

  <resin:otherwise>
    ...
  </resin:otherwise>
<resin:choose>
```

31.178 <resin:ClusterCache>

JCache-style distributed object cache across a cluster pod (java.util.Map).

31.179 <resin:ClusterQueue>

Clustered queue.

31.180 <resin:ClusterSingleSignon>

Cluster-based single signon.

31.181 <resin:DatabaseAuthenticator>

The DatabaseAuthenticator asks a back-end relational database for the password matching a user's name. It uses the DataSource specified by the data-source attribute. data-source refers to an existing admin/database.xtp DataSource.

Table 31.118: <resin:DatabaseAuthenticator> Attributes

ATTRIBUTE	MEANING	DEFAULT
data-source	The pooled JDBC data source. Looks in the application attributes first, then in the global database pools.	None
password-query	An SQL query to get the user's password given the user name. The default query is shown in the code example below.	See below
cookie-auth-query	An SQL query to authenticate the user by a persistent cookie.	None
cookie-auth-update	A SQL update to match a persistent cookie to a user.	None
role-query	A SQL query to determine the user's role. By default, all users are in role "user", but no others.	None
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64
logout-on-session-timeout	If true, the user will be logged out when the session times out.	true

WEB-INF/resin-web.xml for DatabaseAuthenticator

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  <-- Authentication mechanism -->
  <resin:BasicLogin/>

  <-- Role-based authorization -->
  <resin:Allow url-pattern="/foo/*">
     <resin:IfUserInRole role="user"/>
  </resin:Allow>

  <-- The authenticator -->
  <resin:DatabaseAuthenticator'>
    <resin:data-source>test</resin:data-source>
    <resin:password-query>
      SELECT password FROM login WHERE username=?
    </resin:password-query>
    <resin:cookie-auth-query>
      SELECT username FROM LOGIN WHERE cookie=?
    </resin:cookie-auth-query>
    <resin:cookie-auth-update>
      UPDATE LOGIN SET cookie=? WHERE username=?
    </resin:cookie-auth-update>
    <resin:role-query>
      SELECT role FROM LOGIN WHERE username=?
    </resin:role-query>
```

```
  </resin:DatabaseAuthenticator>
</web-app>
```

31.182 <resin:Deny>

The <resin:Deny> tag is the opposite of the top level <resin:Allow>. It restricts access to a particular URL pattern based on any nested conditions. Access attempts that match the condition are denied access. If no conditions are specified, all access to a URL pattern is restricted.

Table 31.119: <resin:Deny> Attributes

ATTRIBUTE	DESCRIPTION
url-pattern	URL pattern describing the resource to be secured.
http-method	HTTP methods that the restriction applies to.

Security-constraint to protect static files

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <!-- protect all .properties files -->
  <resin:Deny url-pattern="*.properties"/>

  <!-- protect the config/ subdirectory -->
  <resin:Deny url-pattern="/config/*"/>
  ...
</web-app>
```

31.183 <resin:DigestLogin>

The HTTP protocol includes a method to indicate to the client that it should digest the password before sending it to the server. This is basically a more secure variant of HTTP basic authentication. The browser submits a digest to Resin instead of submitting a clear-text password. HTTP digest authentication protects the password in transmission.

When using the HTTP digest, Resin will respond to the browser when a secure URL is accessed and ask it to calculate a digest. The steps involved are: Resin provides the client a realm and some other information. The client obtains a user-name and password (usually through a dialog box with a web browser).

The client calculates a digest using the user-name, realm, password, and other information supplied by Resin. The client submits the digest to Resin. Resin does the same digest calculation as the client did. Resin compares the submitted digest and the digest it calculated. If they match, the user is authenticated.

The advantage of this method is that the clear-text password is protected in transmission, it cannot be determined from the digest that is submitted by the client to the server. The <resin:DigestLogin> tag is used to configure digest login.

Using HTTP Digest Authentication

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:DigestLogin/>
  ...
</web-app>
```

31.184 <resin:Dispatch>

Normal servlet dispatching with optional target rewriting.

Table 31.120: <resin:Dispatch> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
target	The target to which to dispatch the request.

Dispatching static resources directly

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Dispatch regexp="\.(jpg|html|txt|gif|png|js|css)$"/>
  <resin:Dispatch regexp="^" target="/controller"/>

</web-app>
```

31.185 <resin:ElasticCloudService>

Enables dynamic servers. When dynamic servers are enable, a new Resin server can join a cluster by contacting the hub triad without needing an explicit server configuration in the resin.properties or resin.xml.

31.186 <resin:FastCgiPort>

FastCGI requests, e.g. from nginx.

31.187 <resin:FastCgiProxy>

Proxies the request to a backend server using FastCGI as a proxy protocol.

Table 31.121: <resin:FastCgiProxy> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
address	An address of a FastCGI server (multiple allowed).

Proxying requests to local FastCGI processes

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

   <resin:FastCgiProxy regexp="\.rb$">
     <address>127.0.0.1:12345</address>
```

```
    <address>127.0.0.1:67890</address>
  </resin:FastCgiProxy>

</web-app>
```

31.188 <resin:FileQueue>

Filesystem based queue.

31.189 <resin:FileTopic>

Filesystem based topic.

31.190 <resin:Forbidden>

Send a HTTP forbidden response.

Table 31.122: <resin:Forbidden> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.

Forbidding access based for unsecured connections

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Forbidden regexp="/foo">
    <resin:Not>
      <resin:IfSecure/>
    </resin:Not>
  </resin:Forbidden>

</web-app>
```

31.191 <resin:FormLogin>

```
 Form-based login is the most common way collecting login
information. Using this login mechanism, you can plug-in a custom
login page with a form storing login information (usually two input
text fields for user-name and password). This custom login page can
then be used with the Resin security framework. This allows for a much
more seamless login mechanism integrated closely with your application,
especially in terms of look and feel.
```

When a URL is secured via form based login, the custom login form page is used to collect authentication information. If authentication succeeds, the user is redirected to the originally requested page. Otherwise, the user is forwarded to an error page (that can also be configured).

A login page can be anything that renders a valid login form such as HTML, Servlet, JSP or JSF. A valid login form must have the action j_security_check . It must also have the parameters j_username and j_password holding the username and password. Optionally, it can also have j_uri and j_use_cookie_auth . j_uri gives the next page to display when login succeeds. If the form-uri-priority is set to true, the user will be forwarded to the j_uri page regardless of what the originally requested page was. If the attribute is set to false (the default), the j_uri page is only used when the originally requested page was the login page itself. j_use_cookie_auth allows Resin to send a persistent cookie to the client to make subsequent logins automatic. When j_use_cookie_auth is set, Resin will store a persistent cookie on the client's machine after authentication succeeds. On all subsequent access, Resin detects the persistent cookie and automatically logs the user in instead of prompting for authentication. This essentially lets you implement "remember me" functionality common in many web-sites. By default, the authentication only lasts for a single session and no persistent login cookie is sent to the client.

The following table outlines all the login parameters recognized by Resin:

Table 31.123: j_security_check Parameters

PARAMETER	MEANING
j_username	The user name.
j_password	The password.
j_uri	Resin extension for the successful display page (optional).
j_use_cookie_auth	Resin extension to allow cookie login (optional).

Table 31.124: <resin:FormLogin> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
form-login-page	The page to be used to prompt the user login.	none
form-error-page	The error page for unsuccessful login.	none
internal-forward	Use an internal redirect on success or a `sendRedirect` .	false
form-uri-priority	If true, the login form's j_uri will override the originally requested URI.	false

WEB-INF/resin-web.xml resin:FormLogin

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:FormLogin form-login-page="/login.html"
                      form-error-page="/login_failure.html"/>
  ...
</web-app>
```

j_security_check form

```
<form action='j_security_check' method='POST'>
<table>
<tr><td>User:<td><input name='j_username'>
<tr><td>Password:<td><input name='j_password'>
<tr><td colspan=2>hint: the password is 'quidditch'
<tr><td><input type=submit>
</table>
</form>
```

31.192 <resin:Forward>

Forwards to the new URL using RequestDispatcher.forward with the target URL.

Table 31.125: <resin:Forward> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
target	The target URL to which to forward the request.
absolute‑target	The absolute target URL to which to forward the request. The absolute target is based on the host root, not relative to the webapp.

Forwarding requests across web-apps

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Forward regexp='^/foo' target='/bar/'/>

</web-app>
```

31.193 <resin:GlobalCache>

JCache-style distributed object cache across the entire Resin system.

31.194 <resin:HttpProxy>

Proxies the request to a backend server using HTTP as a proxy protocol.

Table 31.126: <resin:HttpProxy> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
address	An address of an HTTP server (multiple allowed).

Proxying requests to local HTTP servers

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

   <resin:HttpProxy regexp="\.cgi$">
     <address>127.0.0.1:12345</address>
     <address>127.0.0.1:67890</address>
   </resin:HttpProxy>

</web-app>
```

31.195 <resin:if>

resin:if executes part of the configuration file conditionally. resin:if can be particularly useful in combination with Java command-line properties like -Dfoo=bar to enable development mode or testing configuration.

The resin:if schema is context-dependent. For example, resin:if in a <web-app> will have web-app content while resin:if in a <host> will have host content.

Table 31.127: <resin:if> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
test	the test to perform	required

```
element resin:if {
  attribute test { string }

  context-dependent content
}
```

Example: enable debugging for -Ddevelopment

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:core="http://caucho.com/ns/resin/core">

  <resin:if test="${system['development']}">
    <logger name="com.foo" level="finer"/>
  </resin:if>

  ...
</resin>
```

31.196 <resin:IfAuthType>

Checks for the authentication type, request.getAuthType().

Table 31.128: <resin:IfAuthType> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
value	none, basic, client_cert, digest, or form	none

Redirecting based on authentication type

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/user-area' target='/secret'>
    <resin:IfAuthType value="BASIC"/>
  </resin:Redirect>
```

```
  <resin:Redirect regexp='^/user-area' target='/login'/>

  <resin:XmlAuthenticator password-digest='none'>
    <user name="Aladdin" password="open sesame" group="user"/>
  </resin:XmlAuthenticator>

  <resin:BasicLogin realm-name="foobar"/>

</web-app>
```

31.197 <resin:IfCookie>

Checks for the presence of a named HTTP cookie from request.getCookies().

Table 31.129: <resin:IfCookie> Attributes

ATTRIBUTE	DESCRIPTION
name	The name of the cookie to match.
regexp	A regular expression to match the value of the cookie.
value	A regular expression to match the value of the cookie. (synonym for regexp)

Redirecting if a cookie is set

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/welcome' target='/user-area'>
    <resin:IfCookie name="session"/>
  </resin:Redirect>

</web-app>
```

31.198 <resin:IfCron>

Matches if the current time is in an active range configured by cron-style times.

Table 31.130: <resin:IfCron> Attributes

ATTRIBUTE	DESCRIPTION
enable-at	A cron expression representing the start time(s) at which the predicate matches.
disable-at	A cron expression representing the end time(s) after which the predicate does not match.

Redirecting based on time

```
<web-app xmlns="http://caucho.com/ns/resin"
```

```
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/live-support' target='/hours'>
    <resin:IfCron>
      <resin:enable-at>0 17 * * *</resin:enable-at>
      <resin:disable-at>0 8 * * *</resin:disable-at>
    </resin:IfCron>
  </resin:Redirect>

</web-app>
```

31.199 <resin:IfFileExists>

Matches if the URL corresponds to an actual file. Has no attributes.

Dispatch only if the URL represents an existant file

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Dispatch>
    <resin:IfFileExists/>
  </resin:Dispatch>

  <resin:Redirect regexp="^" target='/does-not-exist'/>
</web-app>
```

31.200 <resin:IfHeader>

Tests for a HTTP header and value match.

Table 31.131: <resin:IfHeader> Attributes

ATTRIBUTE	DESCRIPTION
name	The name of the header to match.
regexp	A regular expression to match the value of the header.
value	A regular expression to match the value of the header. (synonym for regexp)

Redirect based on referrer

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/foo' target='/foo-for-example-customers'>
    <resin:IfHeader name="Referer" value="example.com"/>
  </resin:Redirect>

  <resin:Redirect regexp='^/foo' target='/everyone-else'/>
</web-app>
```

31.201 <resin:IfLocale>

Tests for a Locale match from the HTTP request (Accept-Language header).

Table 31.132: <resin:IfLocale> Attributes

ATTRIBUTE	DESCRIPTION
value	A regular expression to match locale.

Redirecting based on locale

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/your-lang' target='/francais'>
    <resin:IfLocale value="fr"/>
  </resin:Redirect>

  <resin:Redirect regexp='^/your-lang' target='/default'/>
</web-app>
```

31.202 <resin:IfLocalPort>

Compares the local port of the request, request.getLocalPort().

Table 31.133: <resin:IfLocalPort> Attributes

ATTRIBUTE	DESCRIPTION
value	The port to match.

Redirecting based on local port

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

    <resin:Redirect regexp='^/foo' target='/match-5555'>
      <resin:IfLocalPort value="5555"/>
    </resin:Redirect>

    <resin:Redirect regexp='^/foo' target='/match-6666'>
      <resin:IfLocalPort value="6666"/>
    </resin:Redirect>

    <resin:Redirect regexp='^/foo' target='/no-match'/>

</web-app>
```

31.203 <resin:IfMBeanEnabled>

Matches if a MBean is enabled().

Table 31.134: <resin:IfMBeanEnabled> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
name	The MBean name.	required
enabled	Sets the default enable/disable value.	true

The mbean name will be something like:

MBean name for maintenance

```
  resin:type=IfMBeanEnabled,name=my-name,Host=www.example.com,WebApp=/
```

The following example redirects URLs to a maintenance page if an mbean is enabled.

Redirecting based on an MBean

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

    <resin:Redirect regexp='^/foo' target='/scheduled-maintenance.jsp'>
      <resin:IfMBeanEnabled name="maintenance" enabled="false"/>
    </resin:Redirect>

</web-app>
```

31.204 <resin:IfMethod>

Compares the HTTP method, request.getMethod().

Table 31.135: <resin:IfMethod> Attributes

ATTRIBUTE	DESCRIPTION
value	The HTTP method to match.
method	The HTTP method to match (synonym of value).

Redirecting based on HTTP method

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

    <resin:Redirect regexp='^/foo' target='/foo/post'>
      <resin:IfMethod value="POST"/>
    </resin:Redirect>

</web-app>
```

31.205 <resin:IfNetwork>

The <resin:IfNetwork> tag allows or denies requests based on the IP address of the client. IP-constraint is very useful for protecting administration resources to an internal network. It can also be useful for denying service to known problem IPs.

Table 31.136: <resin:IfNetwork> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
value	An IP address to match (multiple allowed).	N/A
cache-size	The size of the IP address LRU cache used for performance.	256

The `/24` in the IP `192.168.17.0/24`

means that the first 24 bits of the IP are matched - any IP address that begins with `192.168.17.` will match. The usage of /bits is optional.

Admin Pages Allowed Only from 192.168.17.0/24

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:Allow url-pattern="/admin/*">
    <resin:IfNetwork value="192.168.17.0/24"/>
  </resin:Allow>
  ...
</web-app>
```

The following example shows how the tag can be used to construct an IP block list:

Block-out Known Trouble-Makers

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:Deny>
    <resin:IfNetwork>
       <resin:value>205.11.12.3</resin:value>
       <resin:value>213.43.62.45</resin:value>
       <resin:value>123.4.45.6</resin:value>
       <resin:value>233.15.25.35</resin:value>
       <resin:value>233.14.87.12</resin:value>
    </resin:IfNetwork>
  </resin:Deny>
  ...
</web-app>
```

Be careful with deny - some ISP's (like AOL) use proxies and the IP of many different users may appear to be the same IP to your server.

If only deny is used, then all IPs are allowed if they do not match a `deny` . If only allow is used, then an IP is denied unless it matches an `allow` . If both are used, then the IP must match both an `allow` and a

`deny`

31.206 <resin:IfQueryParam>

Tests for a HTTP query parameger, request.getParameter().

Table 31.137: <resin:IfQueryParam> Attributes

ATTRIBUTE	DESCRIPTION
name	The name of the query parameter to match.
regexp	A regular expression to match the value of the query parameter.
value	A regular expression to match the value of the query parameter. (synonym for regexp)

Redirecting based on query parameter

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

    <resin:Redirect regexp='^/foo' target='/login'>
      <resin:IfQueryParam name="area" value="login"/>
    </resin:Redirect>

</web-app>
```

31.207 <resin:IfRemoteUser>

Tests against the remote user, request.getRemoteUser().

Table 31.138: <resin:IfRemoteUser> Attributes

ATTRIBUTE	DESCRIPTION
value	The user to match.

Forbidding a path based on remote user

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Forbidden regexp="/harry">
    <resin:Not>
      <resin:IfRemoteUser value="harry"/>
    </resin:Not>
  </resin:Forbidden>

  <resin:XmlAuthenticator>
    <user>harry:H83FykjQaBwvqjGRyIRUHQ==:user</user>
  </resin:XmlAuthenticator>

  <resin:BasicLogin realm-name='global'/>
```

```
</web-app>
```

31.208 <resin:IfSecure>

The <resin:IfSecure> tag restricts access to secure transports, usually SSL.

Table 31.139: <resin:IfSecure> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
value	A boolean value against which HttpServletRequest.isSecure is matched.	true

In the following example, all pages in the web application are enforced to be accessible via SSL only.

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin"
  ...
  <resin:Allow>
    <resin:IfSecure/>
  </resin:Allow>
  ...
</web-app>
```

The default behaviour is for Resin to rewrite any URL that starts with "http:" by replacing the "http:" part with "https:", and then send a redirect to the browser because this configuration.

If the default rewriting of the host is not appropriate, you can set the secure-host-name for the host:

WEB-INF/resin-web.xml

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="app-tier">
  ...
  <host id="...">
    <secure-host-name>https://hogwarts.com</secure-host-name>
    ...
</resin>
```

31.209 <resin:IfUserInRole>

The resin:IfUserInRole condition enforces role-based security. It requires that authenticated users have a specified role.

Table 31.140: <resin:IfUserInRole> Attributes

ATTRIBUTE	DESCRIPTION
role	Roles which are allowed to access the resource.

The following is an example of how <resin:IfUserInRole> might be used:

WEB-INF/resin-web.xml Protecting WebDav for WebDav Users

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:Allow url-pattern="/webdav/*">
    <resin:IfUserInRole role='webdav'/>
  </resin:Allow>
  ...
</web-app>
```

31.210 <resin:import>

<resin:import> reads configuration from another file or set of files. For example, the WEB-INF/web.xml and WEB-INF/resin-web.xml files are implemented as <resin:import> in the app-default.xml.

The target file is validated by the schema of the including context. So a resin:import in <web-app-default> will have a target with a top-level of <web-app>, and a resin:import in <cluster> will have a top-level tag of <cluster>.

Table 31.141: <resin:import> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	a path to a file	either path or fileset is required
fileset	a fileset describing all the files to import.	either path or fileset is required
optional	if true, no error when file does not exist	false

```
element import {
  (path | fileset)
  & optional?
}

element fileset {
  dir
  & exclude*
  & include*
}
```

The following example shows how Resin implements the WEB-INF/web.xml and WEB-INF/resin-web.xml files. Both are simply resin:import in a web-app-default. When Resin configures the web-app, it will process the web-app-default program, and call resin:import for the web.xml file.

Example: import implementation of WEB-INF/web.xml

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
  <cluster id="app-tier">

    <web-app-default>
      <resin:import path="WEB-INF/web.xml" optional="true"/>
      <resin:import path="WEB-INF/resin-web.xml" optional="true"/>
    </web-app-default>
```

```
  </cluster>
</resin>
```

Virtual hosts can use resin:import to add a custom host.xml file. The host.xml can use any <host> attribute, including <host-name> and <host-alias> to customize the virtual host configuration.

Example: adding host.xml in host-deploy

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">
  <cluster id="app-tier">

    <host-deploy path="/var/resin/hosts">
      <host-default>
        <resin:import path="host.xml" optional="true"/>

        <web-app-deploy path="webapps"/>
      </host-default>
    </web-app-default>

  </cluster>
</resin>
```

Some applications may want to split their configuration into multiple files using the fileset. For example, a admin/config-candi.xtp application might want to define beans in WEB-INF/beans/*.xml and give the web-app flexibility in which bean files to create.

Example: Bean IoC fileset in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
          xmlns:core="http://caucho.com/ns/resin/core">

  <resin:import>
    <fileset dir="WEB-INF/beans">
      <include>*.xml</include>
    </fileset>
  </resin:import>

</web-app>
```

31.211 <resin:JaasAuthenticator>

The JaasAuthenticator uses a JAAS LoginModule for authentication. A common use of the JaasAuthenticator is to serve as an adapter for the large number of JAAS LoginModule's included in the Sun JDK for authentication purposes. However, the JAAS authenticator can be used with any valid JAAS login module.

Table 31.142: <resin:JaasAuthenticator> Attributes

ATTRIBUTE	MEANING	DEFAULT
init-param	Adds a property to the LoginModule.	None
login-module	The fully qualified class name of the LoginModule implementation.	Required
logout-on-session-timeout	If true, the user will be logged out when the session times out.	true
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64

WEB-INF/resin-web.xml for JaasAuthenticator

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:JaasAuthenticator>
    <resin:login-module>com.sun.security.auth.module.Krb5LoginModule</resin:login-module>
    <resin:init-param>
      <debug>true</debug>
    </resin:init-param>
  </resin:JaasAuthenticator>
  ...
</web-app>
```

31.212 <resin:JmsConnectionFactory>

Combined Queue and Topic ConnectionFactory.

31.213 <resin:JmxService>

Enables cluster-wide JMX administration.

31.214 <resin:LdapAuthenticator>

```
 The LdapAuthenticator uses JNDI to connect to an LDAP (or Active
Directory) server for authentication.
```

Table 31.143: <resin:LdapAuthenticator> Attributes

ATTRIBUTE	MEANING	DEFAULT
dn-prefix	String to prepend to query before portion selecting user by name.	None
dn-suffix	String to append to query after portion selecting user by name.	None
jndi-env	Add a property to the JNDI provider used for connecting to the LDAP server.	See below
logout-on-session-timeout	If true, the user will be logged out when the session times out.	true
security-authentication	Sets the Context.SECURITY_AUTHENTICATION for the LDAP environment.	
security-principal	Sets the Context.SECURITY_PRINCIPAL for the LDAP environment.	
security-credentials	Sets the Context.SECURITY_CREDENTIALS for the LDAP environment.	

Table 31.143: (continued)

ATTRIBUTE	MEANING	DEFAULT
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64
user-attribute	The attribute name to use in the query for matching the user.	uid
password-attribute	The attribute name to use for obtaining the password.	userPassword
url	The URL for the server.	ldap://localhost:389

WEB-INF/resin-web.xml for LdapAuthenticator

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:LdapAuthenticator password-digest="none">
    <resin:url>ldap://localhost:389</resin:url>
    <resin:dn-suffix>dc=hogwarts,dc=com</resin:dn-suffix>
  </resin:LdapAuthenticator>
  ...
</web-app>
```

31.215 jndi-env

`jndi-env` configures properties of the LDAP provider implementation. Prior to 3.1.1, the URL of the server is specified with `jndi-env` and the

`java.naming.provider.url` property.

The following example shows the usage of the `jndi-env` configuration property:

WEB-INF/resin-web.xml LdapAuthenticator jndi-env

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:LdapAuthenticator password-digest="none">
    <resin:jndi-env java.naming.factory.initial="com.sun.jndi.ldap.LdapCtxFactory"/>
    <resin:jndi-env java.naming.provider.url="ldap://localhost:389"/>
    <resin:dn-suffix>dc=hogwarts,dc=com</dn-suffix>
  </resin:LdapAuthenticator>
  ...
<web-app>
```

31.216 <resin:LoadBalance>

Load balance to a cluster of backend Resin servers.

Table 31.144: <resin:LoadBalance> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
regexp	The regular expression of the URL to match.	None (match all)
cluster	The id of the cluster to which to load balance.	required
strategy	round-robin or adaptive	adaptive

Load Balancing to a backend cluster

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  <cluster id='app-tier'>
    <server id='a' port='6810'/>
    <server id='b' port='6820'/>

    ...
  </cluster>

  <cluster id='web-tier'>
    <server id='http' port='80'/>

    <host id=''>

      <resin:LoadBalance cluster="app-tier"/>

    </host>
  </cluster>
</resin>
```

31.217 <resin:LogService>

Stores high-priority log messages in a database.

31.218 <resin:MemoryQueue>

Memory based queue.

31.219 <resin:MemorySingleSignon>

Memory-based single signon.

31.220 <resin:MemoryTopic>

Memory based topic.

31.221 <resin:message>

Logs a message to the given log file. The content of the element is the message.

```
element resin:message {
  string
}
```

logging in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="http://caucho.com/ns/resin/core">

  <resin:message>Starting server ${server.name}</resin:message>

</web-app>
```

31.222 <resin:MovedPermanently>

Send an HTTP redirect to a new URL specified by target .

Table 31.145: <resin:MovedPermanently> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
target	The target to which to redirect the request.

Flipping elements of a path with a MovedPermanently

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:MovedPermanently regexp='^/([^/]+)/([^/]+)' target='/$2/$1'/>

</web-app>
```

31.223 <resin:Not>

Matches if the child does not match.

Negative matching of a query parameter

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="" target="/match">
    <resin:Not>
      <resin:IfQueryParam name="foo" regexp="bar"/>
    </resin:Not>
  </resin:Redirect>

</web-app>
```

31.224 <resin:NotAnd>

Matches if any child does not match.

Negative matching of at least one of two query parameters

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="" target="/match">
    <resin:NotAnd>
      <resin:IfQueryParam name="foo" regexp="foo"/>
      <resin:IfQueryParam name="bar" regexp="bar"/>
    </resin:NotAnd>
  </resin:Redirect>

</web-app>
```

31.225 <resin:NotOr>

Matches if all the children do not match.

Negative matching of two query parameters

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="" target="/match">
    <resin:NotOr>
      <resin:IfQueryParam name="foo" regexp="foo"/>
      <resin:IfQueryParam name="bar" regexp="bar"/>
    </resin:NotOr>
  </resin:Redirect>

</web-app>
```

31.226 <resin:Or>

Matches if any children match.

Matching at least one of two query parameters

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp="" target="/match">
    <resin:Or>
      <resin:IfQueryParam name="foo" regexp="foo"/>
      <resin:IfQueryParam name="foo" regexp="bar"/>
    </resin:Or>
  </resin:Redirect>

</web-app>
```

31.227 <resin:otherwise>

<resin:otherwise> is the catch-all configuration for a <resin:choose> block when none of the <resin:when> items match.

```
element resin:otherwise {
  context-dependent content
}
```

31.228 <resin:PingMailer>

Mails a notification when a ping check fails.

31.229 <resin:PingThread>

Checks status of Resin sutomatically.

31.230 <resin:ProjectJarRepository>

Maven-style library jar management for webapps.

31.231 <resin:PropertiesAuthenticator>

The PropertiesAuthenticator allows you to use Java properties to store authentication information. This is very useful for a variety of applications such as very small sites, developement, unit testing or integration testing. You can either specify properties in-line in XML or via an external properties file.

Table 31.146: <resin:PropertiesAuthenticator> Attributes

ATTRIBUTE	MEANING	DEFAULT
path	Path to the properties file.	None
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64

The following is an example of in-lining properties with the authenticator. This is useful for extremely simple web-sites maintained by developers as well as testing.

WEB-INF/resin-web.xml - in-line Properties

```
<web-app xmlns="http://caucho.com/ns/resin"
             xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:PropertiesAuthenticator password-digest="none">
     harry=quidditch,user,admin
     draco=mudblood,disabled,user
  </resin:PropertiesAuthenticator>
  ...
```

```
</web-app>
```

Alternatively, external properties files can be used as in the example below. This is useful for a simple site where authentication may be managed by administrators or non-technical users.

WEB-INF/resin-web.xml - File Property

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin"
  ...
  <resin:PropertiesAuthenticator path="WEB-INF/users.properties"/>
  ...
</web-app>
```

WEB-INF/users.properties

```
harry=/Tj/54ylCloUeMi2YQIVCQ===,user,admin
```

As the example indicates, the properties file includes the user as property name while the value is the password (that may be hashed as in the example or may be in plain-text) and any roles that are assigned to the user separated by commas. The password and role values are also separated by a comma.

31.232 <resin:Redirect>

Send an HTTP redirect to a new URL specified by target .

Table 31.147: <resin:Redirect> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
target	The target to which to redirect the request.

Flipping elements of a path with a Redirect

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

  <resin:Redirect regexp='^/([^/]+)/([^/]+)' target='/$2/$1'/>

</web-app>
```

31.233 <resin:RemoteAdminService>

Enables administration by remote agents, like an eclipse console.

31.234 <resin:ScheduledTask>

<resin:ScheduledTask> schedules a job to be executed at specific times or after specific delays. The times can be specified by a cron syntax or by a simple delay parameter. The job can be either a `Runnable` bean, a method specified by an EL expression, or a URL.

When specified as an Java Injection bean, the bean task has full IoC capabilities, including injection, @TransactionAttribute aspects, interception and @Observes.

Table 31.148: <resin:ScheduledTask> Attributes

ATTRIBUTE	DESCRIPTION
cron	a cron-style scheduling description
delay	a simple delay-based execution
init	IoC initialization for the bean
mbean-name	optional MBean name for JMX registration
method	EL expression for a method to be invoked as the task
name	optional IoC name for registering the task
period	how often the task should be invoked in simple mode
task	alternate task assignment for predefined beans

A runnable task

```
<resin:ScheduledTask>
  <cron>* * * * *</cron>
  <task>
    <example:MyClass xmlns:qa="urn:java:com.example"/>
  </task>
</resin:ScheduledTask>
```

Weekly maintenance script

```
<resin:ScheduledTask url="/saturday-maintenance.jsp">
  <cron>* * * * 6</cron>
</resin:ScheduledTask>
```

31.235 <resin:SendError>

Send a HTTP error response.

Table 31.149: <resin:SendError> Attributes

ATTRIBUTE	DESCRIPTION
regexp	The regular expression of the URL to match.
code	The HTTP status to send.
message	A descriptive message to send to the client.

Sending an error code based on request path

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin">

        <resin:SendError regexp="/upcoming" code="501" message="Not Implemented"/>

</web-app>
```

31.236 <resin:set>

resin:set adds an EL variable to the current context.

Table 31.150: <resin:set> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
var	name of the variable to set	required
value	value	required

```
element set {
  var
  & value
  & default
  & attribute * { string }
}
```

Example: resin:set in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin"
          xmlns:resin="http://caucho.com/ns/resin/core">
  <resin:set var="root" value="/var/resin"/>

  <cluster id="app-tier">
    <root-directory>${root}</root-directory>

    ...
  </cluster>
</resin>
```

31.237 <resin:SetHeader>

Sets a response header.

Table 31.151: <resin:SetHeader> Attributes

ATTRIBUTE	DESCRIPTION
name	The name of header to set.
value	The value to which the header will be set.
regexp	regular expression matching URLs.

Example: Setting a custom header based on network

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:SetHeader regexp="\.jsp$" name="X-External" value="true">
    <resin:Not>
      <resin:IfNetwork>
        <resin:value>192.168.1.0/24</resin:value>
```

```
      </resin:IfNetwork>
    </resin:Not>
  </resin:SetHeader>

</web-app>
```

31.238 <resin:SetRequestSecure>

Marks the request as secure.

Setting a request as secure based on network

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:SetRequestSecure>
    <resin:IfNetwork>
      <resin:value>192.168.1.0/24</resin:value>
    </resin:IfNetwork>
  </resin:SetHeader>

</web-app>
```

31.239 <resin:SetVary>

Sets the Vary header.

Table 31.152: <resin:SetVary> Attributes

ATTRIBUTE	DESCRIPTION
value	The value to which the Vary header will be set.
regexp	regular expression matching URLs.

Example: Setting the Vary header for JSPs

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">

  <resin:SetVary regexp="\.jsp" value="Accept"/>

</web-app>
```

31.240 <resin:StatService>

Gathers timed runtime status of Resin for graphing.

Table 31.153: <resin:StatService> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
unix-load-avg-exit-threshold	CPU load triggering a Resin exit (and restart)	100.0
cpu-load-log-info-threshold	CPU load triggering a log message at the info level	1.0
cpu-load-log-warning-threshold	CPU load triggering a log message at the warning level	5.0
cpu-load-log-thread-dump-threshold	CPU load triggering thread dump to the log	5.0
sample-period	how often to sample the statistics	60s
thread-dump-interval	minimum time between thread dumps	15m
unix-load-avg-exit-threshold	Unix CPU load triggering a Resin exit (and restart)	5.0

31.241 <resin:when>

<resin:when> conditionally configures a block within a <resin:choose> block. If the test matches, Resin will use the enclosed configuration.

Table 31.154: <resin:when> Attributes

ATTRIBUTE	DESCRIPTION
test	the test to perform

```
element resin:when {
  attribute test { string },

  context-dependent content
}
```

31.242 <resin:XaLogService>

Transaction log service.

31.243 <resin:XmlAuthenticator>

In a similar vein to the properties authenticator, the XML authenticator allows you to store authentication information in XML - either in-line or in an external file. This authenticator has some of the same use-cases as the properties file authenticator, in a slight more human readable form, especially for a non-technical user.

ATTRIBUTE	MEANING	DEFAULT
user	Specifies a user authentication record. There maybe zero, one or more records.	None

ATTRIBUTE	MEANING	DEFAULT
password-digest	Specifies the digest algorithm and format used to secure the password (see following section in this document for details).	md5-base64
path	Specifies the path to an XML file containing users and passwords.	None
logout-on-session-timeout	If true, the user will be logged out when the session times out.	true

The following example uses in-line XML for authentication. When configuring the XmlAuthenticator in resin.xml (or resin-web.xml), each user adds a new configured user. The user value contains the username, password, and the roles for the user.

XmlAuthenticator in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:resin="urn:java:com.caucho.resin">
  ...
  <resin:XmlAuthenticator password-digest="none">
    <resin:user name="Harry Potter" password="quidditch" group="user,gryffindor"/>
    <resin:user name="Draco Malfoy" password="pureblood" group="user,slytherin"/>
  </resin:XmlAuthenticator>
  ...
</web-app>
```

This example shows how to use an external XML file for authentication:

WEB-INF/resin-web.xml - File XML

```
<web-app xmlns="http://caucho.com/ns/resin">
  ...
  <resin:XmlAuthenticator path="WEB-INF/users.xml"/>
  ...
</web-app>
```

WEB-INF/users.xml

```
<users>
  <user name="harry password="/Tj/54ylCloUeMi2YQIVCQ===" roles="user,admin"/>
<users>
```

31.244 <resin:XmlRoleMap>

Role to group permission mapping.

31.245 RequestPredicate

Interface for custom request predicates.

31.246 <resource>

<resource> is an obsolete synonym for <bean> to define custom singletons. Applications should use the <bean> syntax instead.

31.247 <resource-ref>

<resource-ref> declares that the application needs a resouce configuration.

resource-ref is not directly used by Resin. It's a servlet configuration item intended to tell GUI tools which resources need configuration. Resource configuration in Resin uses the resource, reference, database, and ejb-server tags.

For backwards compatibility, Resin 2.1-style configuration files may still use resource-ref to configure resources, but it's recommended to convert the configuration.

```
element resource-ref {
  attribute id?,
  description*,
  res-ref-name,
  ref-type,
  res-auth,
  res-sharing-scope?
}
```

31.248 <save-mode>

<save-mode> configures when Resin should save a persistence session during a request. The values are:

Table 31.155: <save-mode> Attributes

ATTRIBUTE	DESCRIPTION
after-request	Save the session after the request has been served and completed
before-headers	Save the session before sending headers to the browser
on-shutdown	Only save the session when Resin is shutting down

In some situations, like redirects, a fast browser can send a request back to Resin before the session is persisted with the after-request save-mode. If the server is configured without , the load balancer might send the request to a different server, which may not get the updated session. In the situation, either the save-mode should be changed to before-headers or sticky sessions should be enabled.

If the save-mode is before-headers , the application should take care to make any session changes before sending data to the browser.

31.249 <save-allocation-stack-trace>

<save-allocation-stack-trace> helps debugging application with a missing `Connection.close()` by saving the stack trace where the `Connection.getConnection()` was called. When Resin detects that the connection has failed to close, it can then print the allocation stack trace, which is more informative for tracking down errors.

31.250 <scheduled-task>

The <scheduled-task> is replaced by bean-style #resin:ScheduledTask .

ScheduledTask in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:ee="urn:java:ee"
         xmlns:mypkg="urn:java:com.mycom.mypkg">

  <resin:ScheduleTask>
    <cron>* * 23 * *</cron>

    <task>
      <mypkg:MyBean/>
    </task>
  </resin:ScheduleTask>

</web-app>
```

31.251 <secure>

The <secure> flag requires that the web-app only be accessed in a secure/SSL mode. Equivalent to a <security-constraint>.

```
element secure {
  r_boolean-Type
}
```

31.252 <secure-host-name>

<secure-host-name> sets a host-name or URL to be used for secure redirection. For some security configurations, Resin needs to redirect from an insecure site to a secure one. The <secure-host-name> configures the host to redirect to.

See admin/security.xtp .

```
element secure-host-name {
  string
}
```

31.253 <security-constraint>

Specifies protected areas of the web site. Sites using authentication as an optional personalization feature will typically not use any security constraints. See admin/security-overview.xtp for an overview.

Security constraints can also be custom classes.

See admin/security-overview.xtp for an overview of security issues and configuration.

Table 31.156: <security-constraint> Attributes

ATTRIBUTE	DESCRIPTION
auth-constraint	Defines a security condition based on a logged-in user's role
constraint	Defines a custom security condition
ip-constraint	Defines a security condition based the remote IP address
role-name	Defines role-name requirement
url-pattern	URL pattern to match the security constraint
user-data-constraint	Defines SSL or non-SSL requirements
web-resource-collection	URL patterns and HTTP methods defining the constraint

```
element security-constraint {
  auth-constraint*
  & constraint*
  & ip-constraint*
  & role-name*
  & user-data-constraint*
  & url-pattern?
  & web-resource-collection*
}
```

Example: user role required in WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

<security-constraint>
  <web-resource-collection>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint role-name='user'>
</security-constraint>

</web-app>
```

31.254 <security-manager>

<security-manager> enables the use of the security manager for the JVM. Because the JVM security manager is very slow, we generally do not recommend enabling it for server applications. Instead, see the admin/starting-resin-watchdog.xtp configuration for alternative methods for securiting the JVM in ISP configurations.

```
element security-manager {
  r_boolean-Type
}
```

Example: enabling security-manager

```
<resin xmlns="http://caucho.com/ns/resin">

  ...
  <security-manager/>
  ...
```

31.255 <security-provider>

<security-provider> adds one or more security providers. Each entry specifies the name of a security provider class. The name is used to instantiate an instance of the object, which is then passed to Security.addProvider .

```
element security-provider {
  string
}
```

Example: adding custom security providers

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">

  ...

  <security-provider>
    com.sun.net.ssl.internal.ssl.Provider
  </security-provider>
  <security-provider>
    example.MyProvider
  </security-provider>

  ...
```

31.256 <server>

<server> configures a JVM instance in the cluster. Each <server> is uniquely identified by its id attribute. The id will match the -server command line argument.

The server listens to an internal network address, e.g. 192.168.0.10:6800 for clustering, load balancing, and administration.

The current server is managed with a javadoc|com.caucho.management.server.ServerMXBean . The is resin:type=Server .

Peer servers are managed with javadoc|com.caucho.management.server.ServerConnectorMXBean . The ObjectName is resin:type=Serve
id .

Table 31.157: <server> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
id	unique server identifier	required
address	IP address of the cluster port	127.0.0.1
port	The cluster port	6800

server

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <server id="a" address="192.168.0.10" port="6800">
          <http port="8080"/>
        </server>

        <server id="b" address="192.168.0.11" server-port="6800">
          <http port="8080"/>
        </server>

        <server id="c" address="192.168.0.12" server-port="6800">
          <http port="8080"/>
        </server>

        <host id="">
          ...
    </cluster>
</resin>
```

Main configuration for the server, configuring ports, threads and virtual hosts. Common resources for all virtual hosts and web-apps. Thread pooling HTTP and Server/Cluster ports Caching virtual host configuration and common web-app-default

The <server> will generally contain a <class-loader> configuration which loads the resin/lib jars dynamically, allowing for system-wide jars to be dropped into resin/lib. <server> configures the main dynamic environment. Database pools common to all virtual hosts, for example, should be configured in the <server> block.

The <server> configures the <thread-pool> and a set of <http> and <server> ports which share the thread pool. Requests received on those ports will use worker threads from the thread pool.

Table 31.158: <server> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
alternate-session-url-prefix	a prefix to add the session to the beginning of the URL as a path prefix instead of the standard ;jsessionid= suffix. For clients like mobile devices with limited memory, this will allow careful web designers to minimize the page size.	null
keepalive-max	the maximum number of keepalive connections	512
keepalive-timeout	the maximum time a connection is maintained in the keepalive state	15s

alternate-session-url-prefix

```
<server>
  ...
  <alternate-session-url-prefix>/~J=</alternate-session-url-prefix>
  ...
```

31.257 EL variables and functions

Table 31.159: EL variables defined by <server>

VARIABLE	CORRESPONDING API
serverId	server .getServerId()
root-dir	server .getRootDirectory()
server-root	server .getRootDirectory()

Table 31.160: EL functions defined by <server>

FUNCTION	CORRESPONDING API
jndi	javadoc
com.caucho.naming.Jndi	lookup(String)

31.258 <server-default>

<server-default> defines default values for all <server> instances. Since most <server> configuration is identical for all server instances, the shared configuration belongs in a <server-default>. For example, <http> ports, timeouts, JVM arguments, and keepalives are typically identical for all server instances and therefore belong in a server-default.

server

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <server-default>
           <thread-max>512</thread-max>

           <jvm-arg>-Xmx512m -Xss1m</jvm-arg>

           <http port="8080"/>
        </server-default>

        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.11" port="6800"/>
        <server id="c" address="192.168.0.12" port="6800"/>

        <host id="">
          ...
    </cluster>
</resin>
```

31.259 <server-header>

Configures the HTTP Server: header which Resin sends back to any HTTP client.

```
element server-header {
  string
}
```

server-header

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <server-header>MyServer/1.0</server-header>
    </cluster>
</resin>
```

31.260 <servlet>

Defines a servlet to process HTTP URL requests. The servlet class can either implement javax.servlet.Servlet to handle the HTTP request directly or it can use a remoting protocol like Hessian to handle remoting requests.

Since servlets are full admin/config-candi.xtp beans, they can use dependency injection, EJB aspects like @TransactionAttribute, custom @InterceptorType interceptors, and listen for @Observes events.

Table 31.161: <servlet> Attributes

ATTRIBUTE	DESCRIPTION
init	admin/config-candi.xtp initialization

Table 31.161: (continued)

ATTRIBUTE	DESCRIPTION
init-param	Initialization parameters
load-on-startup	Initializes the servlet when the server starts.
protocol	Protocol configuration for Resin remoting.
run-at	Times to execute the servlet automatically.
servlet-name	The servlet's name (alias)
servlet-class	The servlet's class (In Resin, defaults to servlet-name)

```
element servlet {
  servlet-name
  < (servlet-class | jsp-file)
  < init*
  < init-param*
  < load-on-startup?
  < protocol?
  < run-as?
  < run-at?
}
```

Example: WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns">

  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>test.HelloWorld</servlet-class>
    <init-param>
      <param-name>title</param-name>
      <param-value>Hello, World</param-value>
    </init-param>
  </servlet>

  <!-- using Resin shortcut syntax -->
  <servlet servlet-name='cron'
           servlet-class='test.DailyChores'>
    <init-param title='Daily Chores'/>
    <load-on-startup/>
    <run-at>3:00</run-at>
  </servlet>

  <!-- mapping a url to use the servlet -->
  <servlet-mapping url-pattern='/hello.html'
                   servlet-name='hello'/>

</web-app>
```

Several servlet configurations might configure the same servlet class with different init-param values. Each will have a separate servlet-name .

Example: multiple servlets using the same class

```
<web-app xmlns="http://caucho.com/ns/resin">
  <servlet servlet-name='foo-a'>
    <servlet-class>test.FooServlet</servlet-class>
    <init-param name='foo-a sample'/>
  </servlet>
```

```
  <servlet servlet-name='foo-b'>
    <servlet-class>test.FooServlet</servlet-class>
    <init-param name='foo-b sample'/>
  </servlet>
</web-app>
```

load-on-startup can specify an (optional) integer value. If the value is 0 or greater, it indicates an order for servlets to be loaded, servlets with higher numbers get loaded after servlets with lower numbers.

There are a number of named servlets that are usually available to a Resin application, as defined in $RESIN_HOME/conf/app-default.xml .

Example: servlet-mappings in $RESIN_HOME/conf/app-default.xml

```
  <servlet servlet-name="directory"
           servlet-class="com.caucho.servlets.DirectoryServlet"/>

  <servlet servlet-name="file"
           servlet-class="com.caucho.servlets.FileServlet"/>

  <servlet servlet-name="jsp"
           servlet-class="com.caucho.jsp.JspServlet"/>

  <servlet servlet-name="xtp"
           servlet-class="com.caucho.jsp.XtpServlet"/>

<servlet servlet-name="j_security_check"
         servlet-class="com.caucho.server.security.FormLoginServlet"/>
```

31.261 <servlet-hack>

Use of servlet-hack is discouraged. Using servlet-hack violates the JDK's classloader delegation model and can produce surprising ClassCastExceptions.

servlet-hack reverses the normal class loader order. Instead of parent classloaders having priority, child classloaders have priority.

```
element servlet-hack {
  boolean
}
```

31.262 <servlet-mapping>

Maps url patterns to servlets. servlet-mapping has two children, url-pattern and servlet-name . url-pattern selects the urls which should execute the servlet.

Table 31.162: <servlet-mapping> Attributes

ATTRIBUTE	DESCRIPTION
init	admin/config-candi.xtp configuration of the servlet.
protocol	Defines the optional remoting protocol.
servlet-class	The servlet-mapping can define the servlet directly as a shortcut.
servlet-name	The servlet name
url-pattern	A pattern matching the url: /foo/* , /foo , or *.foo
url-regexp	A regular expression matching the portion of the url that follows the

```
element servlet-mapping {
  init?
  & protocol?
  & servlet-class?
  & servlet-name?
  < url-pattern*
  < url-regexp*
}
```

Example: WEB-INF/resin-web.xml <servlet-mapping>

```
<web-app xmlns="http://caucho.com/ns/resin">

  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>test.HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <url-pattern>/hello.html</servlet-class>
    <servlet-name>hello</servlet-class>
  </servlet-mapping>

  <!-- resin shortcut syntax -->
  <servlet-mapping url-pattern='*.xtp'
                   servlet-name='com.caucho.jsp.XtpServlet'/>

</web-app>
```

In Resin, the special `servlet-name` *invoker* is used to dispatch servlets by class name.

Example: WEB-INF/resin-web.xml servlet invoker

```
<web-app xmlns="http://caucho.com/ns/resin">

  <!--
    used with urls like
    http://localhost:8080/servlets/test.HelloServlet
  -->
  <servlet-mapping url-pattern="/servlet/*" servlet-name="invoker"/>

</web-app>
```

There are a number of mappings to servlets that are usually available to a Resin application, as defined in $RESIN_HOME/conf/app-default.xml .

Example: servlet-mappings in $RESIN_HOME/conf/app-default.xml

```
<cluster>

<web-app-default>
  <servlet-mapping url-pattern="*.jsp" servlet-name="jsp"/>
  <servlet-mapping url-pattern="*.xtp" servlet-name="xtp"/>

  <servlet-mapping url-pattern="/servlet/*" servlet-name="invoker"/>
  <servlet-mapping url-pattern="/" servlet-name="file"/>
</web-app-default>
</cluster>
```

The plugins use servlet-mapping to decide which URLs to send to Resin. The following servlet-name values are used by the plugins:

Table 31.163: servlet-name values used by plugins

ATTRIBUTE	DESCRIPTION
plugin_match	The plugin will send the request to Resin, but Resin will ignore the entry. Use to get around regexp limitations. (Resin 1.2.2)
plugin_ignore	The plugin will ignore the request. Use this to define a sub-url the web server should handle, not Resin. (Resin 1.2.2)

31.263 <servlet-regexp>

Maps URL by regular expressions to custom servlets.

```
element servlet-regexp {
  init?
  & servlet-class?
  & servlet-name?
  & url-regexp
}
```

```
<servlet-regexp url-regexp="/([^.]*).do"
                servlet-class="qa.\${regexp[1]}Servlet">
  <init a="b"/>
</servlet-regexp>
```

31.264 ServletRequestPredicate

Although extremely rare, it is sometimes useful to create a custom predicate (for example for encapsulating complex custom authorization logic). You can easily do this by extending com.caucho.rewrite.RequestPredicate . This essentially allows you to create your own <IfXXX> rule.

The following example demonstrates how to create a custom Resin predicate:

WEB-INF/resin-web.xml - Custom rule

```
<web-app xmlns="http://caucho.com/ns/resin"
         xmlns:resin="urn:java:com.caucho.resin"
         xmlns:foo="urn:java:com.foo"
 ...
 <resin:Allow url-pattern="/safe/*"
    <foo:IfMyTest value="abcxyz"/>
 </resin:Allow url-pattern="/safe/*"
 ...
</web-app>
```

```
package com.foo;

import javax.servlet.http.HttpServletRequest;
import com.caucho.security.ServletRequestPredicate;

public class IfMyTest extends ServletRequestPredicate {
  private String value;
```

```
  // Your custom attribute for the tag.
  public void setValue(String value)
  {
    this.value = value;
  }

  // Here you must actually determine the match.
  public boolean isMatch(HttpServletRequest request)
  {
    return value.equals(request.getHeader("Foo"));
  }
}
```

31.265 <session-cookie>

Configures the cookie used for servlet sessions.

```
element session-cookie {
  string
}
```

31.266 <session-config>

<session-config> configures Resin's session handling, including the cookies Resin sends, the maximum sessions, and session persistence and clustering.

See also: admin/clustering-overview.xtp for more information about distributed sessions and persistence.

Table 31.164: <session-config> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
always-load-session	Reload data from the store on every request	false
always-save-session	Save session data to the store on every request	false
enable-cookies	Enable cookies for sessions	true
enable-url-rewriting	Enable URL rewriting for sessions	true
cookie-version	Version of the cookie spec for sessions	1.0
cookie-domain	Domain for session cookies	none
cookie-domain-regexp	Regex for cookies-domain for multi-domain webapps	none
cookie-max-age	Max age for persistent session cookies	none
cookie-length	Maximum length of the cookie	
ignore-serialization-errors	When persisting a session, ignore any values which don't implement java.io.Serializable	true
invalidate-after-listener	If true, invalidate the sessions after the session listeners have been called	
reuse-session-id	Reuse the session id even if the session has timed out. A value of false defeats single signon capabilities. (resin 2.0.4)	true
session-max	Maximum active sessions	4096

Table 31.164: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
session-timeout	The session timeout in minutes, 0 means never timeout.	30 minutes
serialization-type	Use one of "java" or "hessian" for serialization, hessian is significantly faster and smaller (resin 3.1.2)	java
serialize-collection-type	For hessian serialization, ignore the collection type when serializing lists (disable for some Hibernate lists)	true
save-mode	When to save sessions, one of "before-headers", "after-request", or "on-shutdown"	after-request
use-persistent-store	Uses the current deploy-ref.xtp#session-config to save sessions. (resin 3.0.8)	none

```
element session-config {
  always-load-session?
  & always-save-session?
  & cookie-append-server-index?
  & cookie-domain?
  & cookie-domain-regexp?
  & cookie-length?
  & cookie-max-age?
  & cookie-port?
  & cookie-secure?
  & cookie-version?
  & enable-cookies?
  & enable-url-rewriting?
  & ignore-serialization-errors?
  & invalidate-after-listener?
  & reuse-session-id?
  & save-mode?
  & save-on-shutdown?
  & serialization-type?
  & session-max?
  & session-timeout?
  & use-persistent-store?
}
```

The session-timeout and session-max are usually used together to control the number of sessions. Sessions are stored in an LRU cache. When the number of sessions in the cache fills up past session-max , the oldest sessions are recovered. In addition, sessions idle for longer than `session-timeout` are purged.

using session-config and session-timeout to control the number of sessions

```
<web-app id='/dir'>

  <session-config>
     <!-- 2 hour timeout -->
     <session-timeout>120</session-timeout>
     <session-max>4096</session-max>
  </session-config>

</web-app>
```

cookie-length is used to limit the maximum length for the session's generated cookie for special situations like WAP devices. Reducing this value reduces the randomness in the cookie and increases the chance of session collisions.

reuse-session-id defaults to true so that Resin can share the session id amongst different web-apps.

The class that corresponds to <session-config> is com.caucho.server.session.SessionManager

31.267 <session-sticky-disable>

Disables sticky sessions from the load balancer.

```
element session-sticky-disable {
  r_boolean-Type
}
```

31.268 <session-url-prefix>

Configures the URL prefix used for session rewriting. Session rewriting is discouraged as a potential security issue.

```
element session-cookie {
  string
}
```

31.269 <shutdown-wait-max>

<shutdown-wait-max> configures the maximum time the server will wait for the graceful shutdown before forcing an exit.

```
element shutdown-wait-max {
  r_period-Type
}
```

31.270 <simple-loader>

<simple-loader> Configures a simple WEB-INF/classes -style class loader.

class files in the specified directory will be loaded without any special compilation steps (in contrast with compiling-loader.)

Table 31.165: <simple-loader> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required
prefix	Class package prefix to only load to a subset of classes. Resin 3.0	none

```
element simple-loader {
  path
  & prefix?
```

```
}
```

31.271 <socket-timeout>

<socket-timeout> is the maximum time a socket admin/clustering-overview.xtp and deploy-ref.xtp#session-config will wait for a read or write to a cluster socket.

Crashed servers may never respond to a read request or accept a write. The socket-timeout lets Resin recover from these kinds of crashes.

Lower values can detect crashes more quickly, but too-low values may report bogus failures when the server machine is just a little slow.

socket-timeout

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">
        <server-default>
          <socket-timeout>60s</socket-timeout>
        </server-default>

        <server id="a" address="192.168.0.10" port="6800"/>
        <server id="b" address="192.168.0.11" port="6800"/>

        <host id="">
          ...
    </cluster>
</resin>
```

31.272 <spy>

The <spy> tag is a very useful logging tag for debugging database problems. If <spy> is enabled, all database queries will be logged at the "fine" level. Applications can use <spy> to debug unexpected database queries, or to improve query performance.

Example: spy output

```
0.6:setString(1,1)
0.6:executeQuery(select o.DATA from my_bean o where o.ID=?)
```

31.273 <ssl-session-cookie>

Defines an alternative session cookie to be used for a SSL connection. Having two separate cookies increases security.

```
element ssl-session-cookie {
  string
}
```

Example: ssl-session-cookie

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="web-tier">
        <ssl-session-cookie>SSLJSESSIONID</ssl-session-cookie>
        ...
    </cluster>
</resin>
```

31.274 <startup-mode>

<startup-mode> configures the virtual-host's behavior on Resin startup, either "automatic", "lazy" or "manual".

Table 31.166: <startup-mode> Attributes

MODE	DESCRIPTION
automatic	automatically start when Resin starts
lazy	start only when the first request is received
manual	start only when JMX administration requests a start

```
element startup-mode {
  string
}
```

31.275 <stat-service>

<stat-service> periodically checks the status of the server, and reports errors as necessary.

Table 31.167: <stat-service> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
unix-load-avg-exit-threshold	CPU load triggering a Resin exit (and restart)	100.0
cpu-load-log-info-threshold	CPU load triggering a log message at the info level	1.0
cpu-load-log-warning-threshold	CPU load triggering a log message at the warning level	5.0
cpu-load-log-thread-dump-threshold	CPU load triggering thread dump to the log	5.0
sample-period	how often to sample the statistics	60s
thread-dump-interval	minimum time between thread dumps	15m

```
element stat-service {
  unix-load-avg-exit-threshold?
  & cpu-load-log-info-threshold?
  & cpu-load-log-warning-threshold?
  & cpu-load-thread-dump-threshold?
  & sample-period?
  & thread-dump-interval?
}
```

31.276 <statistics-enable>

<statistics-enable> enables more detailed statistics for the

`WebAppMXBean` administration. The statistics gathering is disabled by default for performance reasons.

```
element statistics-enable {
  r_boolean-Type
}
```

31.277 <stderr-log>

Configures the destination for `System.err` .

The admin/logging.xtp describes stderr-log in detail.

Table 31.168: <stderr-log> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-format	defines a format string for log rollover	
path	sets the VFS path for the log file	
path-format	sets a pattern for creating the VFS path for the messages	
rollover-count	sets the maximum number of rollover files	
rollover-period	sets the number of days before a log file rollover	1m
rollover-size	sets the maximum log size before a rollover	1g
timestamp	sets the formatting string for the timestamp label	

```
element stderr-log {
  (path | path-format)
  & archive-format?
  & rollover-period?
  & rollover-size?
  & rollover-count?
  & timestamp?
}
```

31.278 <stdout-log>

Configures the destination for `System.out` .

The admin/logging.xtp describes stderr-log in detail.

Table 31.169: <stdout-log> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-format	defines a format string for log rollover	
path	sets the VFS path for the log file	
path-format	sets a pattern for creating the VFS path for the messages	
rollover-count	sets the maximum number of rollover files	

Table 31.169: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
rollover-period	sets the number of days before a log file rollover	1m
rollover-size	sets the maximum log size before a rollover	1g
timestamp	sets the formatting string for the timestamp label	

```
element stdout-log {
  (path | path-format)
  & archive-format?
  & rollover-period?
  & rollover-size?
  & rollover-count?
  & timestamp?
}
```

31.279 <strict-mapping>

Forces servlet-mapping to follow strict Servlet 2.2, disallowing PATH_INFO. Value is true or false .

```
element strict-mapping {
  r_boolean-Type
}
```

Example: enabling strict-mapping in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <strict-mapping>true</strict-mapping>
</web-app>
```

31.280 <system-property>

Sets a Java system property. The effect is the same as if you had called java.lang.System#setProperty(String,String) before starting Resin.

```
element system-property {
  attribute * { string }+
}
```

Example: setting system property

```
<resin xmlns="http://caucho.com/ns/resin">
  <system-property foo="bar"/>
</resin>
```

31.281 Time intervals

Time intervals are used throughout Resin configuration. To specify the units for a given interval, use the appropriate suffix as shown below.

SUFFIX	MEANING
s	seconds
m	minutes
h	hours
D	days

31.282 <temp-dir>

<temp-dir> configures the application temp directory. This is the path used in javax.servlet.context.tempdir .

```
element temp-dir {
  string
}
```

31.283 <thread-idle-max>

<thread-idle-max> configures the maximum number of idle threads in the thread pool. <thread-idle-max> works with <thread-idle-min> to maintain a steady number of idle threads, avoiding the creation or destruction threads when possible.

<thread-idle-max> should be set high enough beyond <thread-idle-min> so a spiky load will avoid creating a thread and then immediately destroying it.

thread-idle-max in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <thread-max>512</thread-max>

      <thread-idle-min>10</thread-idle-min>
      <thread-idle-max>20</thread-idle-max>

      <jvm-arg>-Xss1m -Xmx1024m</jvm-arg>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.284 <thread-idle-min>

<thread-idle-min> configures the minimum number of idle threads in the thread pool. <thread-idle-min> helps spiky loads, avoiding delays for thread requests by keeping threads ready for future requests. When the number of idle threads drops below <thread-idle-min>, Resin creates a new thread.

<thread-idle-min> should be set high enough to deal with load spikes. Since idle threads are relatively inexpensive in modern operating systems, having a number of idle threads is not a major resource hog, especially since these threads are idle, waiting for a new job.

<thread-idle-min> works together with <thread-idle-max> to avoid thread allocation thrashing, i.e. avoiding creating a new thread because of <thread-idle-min> and then quickly destroying it because of <thread-idle-max>.

thread-idle-min in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <thread-max>512</thread-max>

      <thread-idle-min>10</thread-idle-min>
      <thread-idle-max>20</thread-idle-max>

      <jvm-arg>-Xss1m -Xmx1024m</jvm-arg>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.285 <thread-max>

<thread-max> configures the maximum number of threads managed by Resin's thread pool. Resin's thread pool is used for connection threads, timers, and Resin worker threads for JMS, JCA and EJB. Since Resin's thread pool only manages Resin threads, the actual number of threads in the JVM will be higher.

Modern operating systems can handle a fairly large number of threads, so values of 512 or 1024 are often reasonable values for thread-max. The main limitation for thread-max is virtual memory. Since each thread takes up stack space (configured with -Xss), a 32-bit system might have a thread limit based on virtual memory.

For example, on Linux the user space is only 2G. If the heap memory is 1024m (-Xmx1024m) and the stack size is 1m (-Xss1m), the maximum number of threads is somewhat less than 1024.

In general, JVMs do not handle running out of threads very well, either freezing or throwing out of memory errors. Although it may be necessary to limit the number of threads to avoid running out of memory, <thread-max> should generally be set to a high value.

thread-max in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="web-tier">
    <server-default>
      <thread-max>512</thread-max>

      <jvm-arg>-Xss1m -Xmx1024m</jvm-arg>
    </server-default>

    <server id="web-a" address="192.168.0.10"/>
        ...
  </cluster>
</resin>
```

31.286 <transaction-timeout>

<transaction-timeout> configures the maximum time a transaction can be alive before a mandatory rollback.

31.287 <tree-loader>

<tree-loader> configures a jar library, WEB-INF/lib -style class loader similar to library-loader , but will also find `.jar` and `.zip` files in subdirectories.

Table 31.170: <tree-loader> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
path	Filesystem path for the class loader. Since Resin 3.0	required

```
element tree-loader {
  path
}
```

31.288 <url-character-encoding>

Defines the character encoding for decoding URLs.

The HTTP specification does not define the character-encoding for URLs, so the server must make assumptions about the encoding.

```
element url-character-encoding {
  string
}
```

31.289 <url-regexp>

<url-regexp> matches the portion of the url that follows the . A webapp in `webapps/ROOT` , and a url `http://localhost/foo/h` will have a value of `"/foo/hello.html"` for the purposes of the regular expression match. A webapp in `webapps/baz` and a url `http://localhost/baz/hello.html` will have a url of

`"/hello.html"` for the purposes of the regular expression match, because "/baz" is the context path.

31.290 <user-data-constraint>

Restricts access to secure transports, such as SSL

Table 31.171: <user-data-constraint> Attributes

ATTRIBUTE	DESCRIPTION
transport-guarantee	Required transport properties. NONE, INTEGRAL, and CONFIDENTIAL are allowed values.

```
element user-data-constraint {
  transport-guarantee
}
```

31.291 <user-name>

<user-name> configures the operating system user Resin should run as. Since the HTTP port 80 is protected in Unix, the web server needs to start as root to bind to port 80. For security, Resin should switch to a non-root user after binding to port 80.

resin.xml with user-name

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">

    <server-default>
      <http port="80"/>

      <user-name>resin</user-name>
    </server-default>

    <server id="web-a"/>
    ...
  </cluster>
</resin>
```

31.292 Variables: java

Table 31.172: java properties

PROPERTY	DESCRIPTION
version	Returns the JDK version from the java.version property

31.293 Variables: resin

Table 31.173: resin properties

PROPERTY	DESCRIPTION
address	The machine's address as returned by InetAddress
conf	The path to the resin.xml
home	The --resin-home value, i.e. the location of the Resin installation
homeName	The local hostname as returned by InetAddress
id	The --serverId command line value (see serverId)
isProfessional()	True if Resin professional is installed and licensed
root	The --resin-root value, i.e. the site's deployment directory
serverId	The --serverId command line value
version	The Resin version
versionDate	The compilation date of Resin

31.294 Variables: system

`System.getProperties()`

value.

Example: returning -Dfoo=bar

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">

  <resin:message>${system['foo']}</resin:message>

  ...
</resin>
```

31.295 <watchdog>

The <watchdog> tag is used in ISP-style configurations where the <watchdog-manager> is configured separately from the resin.xml instances, and where the configuration file is generally not readable by the instance users.

The <watchdog> tag corresponds to the <server> tag in standard resin.xml configurations, and specifies the resin.xml, the root directory and resin-user.

Table 31.174: <watchdog> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
chroot	calls chroot before starting the Resin process	
group-name	setgid value for unix systems	
java-exe	java executable to use for the Resin instance	java
java-home	JAVA_HOME value for the Resin instance	
open-port	list of ports the watchdog should open for the Resin instance, e.g. for ports that require root access	
resin.xml	Path to the resin.xml file for the Resin instance	same as for watchdog
resin-root	Root directory for the Resin instance	same as for watchdog

```
element watchdog {
  attribute id { string }
  & chroot?
  & group-name?
  & java-exe?
  & java-home?
  & jvm-arg*
  & resin.xml?
  & resin-root?
  & open-port { address & port }*
  & user-name?
}
```

31.296 <watchdog-arg>

The <watchdog-arg> configures arguments for the watchdog process. The watchdog improves reliability by monitoring the Resin instance, restarting it if necessary.

The <watchdog-arg> typically is used to enable for the watchdog JVM.

resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster id="app-tier">
    <server-default>

      <watchdog-arg>-Dcom.sun.management.jmxremote</watchdog-arg>

    </server-default>

    ...

  </cluster>
</resin>
```

31.297 <watchdog-manager>

For ISP configurations, <watchdog-manager> is used for a separate resin.xml just to configure the watchdog-manager itself. The <watchdog-manager> selects the <user-name>, <resin.xml>, ports, and Resin home directories before giving access to the user's Resin instance.

Table 31.175: <watchdog-manager> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
watchdog	Watchdog configuration for a Resin instance, corresponding to a <server> in the resin.xml	
watchdog-address	The TCP address the watchdog-manager listens for start, stop, status	127.0.0.1
watchdog-default	Defaults applied to all watchdog instances	
watchdog-jvm-arg	JVM arguments for the watchdog-manager when launched	
watchdog-port	The TCP port the watchdog-manager listens for start, stop, status	6700

```
element watchdog-manager {
  watchdog*
  & watchdog-address?
  & watchdog-default*
  & watchdog-jvm-arg*
  & watchdog-port?
}
```

31.298 <watchdog-port>

<watchdog-port> configures the administration port for the watchdog JVM. The watchdog launches the server JVM and monitors its health, restarting the JVM when necessary to improve site reliability. The command line arguments use the watchdog-port for the "start" and "stop" commands to tell the watchdog to start and stop the server JVM. The administration also uses the watchdog port for watchdog administration.

The watchdog port will use the same <address> as the server, so it will always be an internal network address, never an external internet address.

31.299 <web-app>

web-app configures a web application.

When specified by id , the application will be initialized on server start. When specified by url-regexp , the application will be initialized at the first request. This means that load-on-startup servlets may start later than expected for url-regexp applications.

Table 31.176: <web-app> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
active-wait-time	how long a thread should wait for the web-app to initialize before returning a 503-busy.	15s
archive-path	Specifies the location of the web-app's .war file.	n/a
context-path	Specifies the URL prefix for the web-app.	the id value
id	The url prefix selecting this application.	n/a
redeploy-mode	*automatic* or *manual*, see redeploy-mode	automatic
redeploy-check-interval	how often to check the .war archive for redeployment	60s
root-directory	The root directory for the application, corresponding to a url of / id / . A relative path is relative to the root-directory of the containing host . Can use regexp replacement variables.	A relative path constricted with the id or the regexp match
startup-mode	*automatic*, *lazy*, or *manual*, see startup-mode	automatic
startup-priority	specifies a priority for web-app startup to force an ordering between webapps	-1
url-regexp	A regexp to select this application.	n/a

The following example creates a web-app for /apache using the Apache htdocs directory to serve pages.

```
<host id=''>
  <web-app id='/apache' root-directory='/usr/local/apache/htdocs'>

  ...

</host>
```

The following example sets the root web-app to the IIS root directory.

```
  <web-app id='/' root-directory='C:/inetpub/wwwroot'>
```

When the web-app is specified with a url-regexp , root-directory can use replacement variables ($2).

In the following, each user gets his or her own independent application using ~user .

```
<host id=''>

  <web-app url-regexp='/~([^/]*)'
           root-directory='/home/$1/public_html'>

    ...

  </web-app>

</host>
```

31.300 <web-app-default>

<web-app-default> defines default values for any in the cluster.

Example: web-app-default

```
<resin xmlns="http://caucho.com/ns/resin">
    <cluster id="app-tier">

        <web-app-default>
            <servlet servlet-name="resin-php"
                     servlet-class="com.caucho.quercus.servlet.QuercusServlet"/>

            <servlet-mapping url-pattern="*.php"
                             servlet-name="resin-php"/>
        </web-app-default>

        <host id="">
          ...
    </cluster>
</resin>
```

31.301 <web-app-deploy>

Specifies war expansion.

web-app-deploy can be used in web-apps to define a subdirectory for war expansion. The tutorials in the documentation use web-app-deploy to allow servlet/tutorial/helloworld to be an independent war file.

Table 31.177: <web-app-deploy> Attributes

ATTRIBUTE	DESCRIPTION	DEFAULT
archive-directory	directory containing the .war files	value of path
dependency-check-interval	How often to check the .war files for a redeploy	60s
expand-cleanup-fileset	defines the files which should be automatically deleted when an updated .war expands	all files

Table 31.177: (continued)

ATTRIBUTE	DESCRIPTION	DEFAULT
expand-directory	directory where wars should be expanded	value of path
expand-prefix	prefix string to use when creating the expansion directory, e.g. _war_	
expand-suffix	prefix string to use when creating the expansion directory, e.g. .war	
path	The path to the webapps directory	required
redeploy-mode	"automatic" or "manual"	automatic
require-file	additional files to use for dependency checking for auto restart	
startup-mode	"automatic", "lazy" or "manual"	automatic
url-prefix	url-prefix added to all expanded webapps	""
versioning	if true, use the web-app's numeric suffix as a version	false
web-app-default	defaults to be applied to expaned web-apps	
web-app	overriding configuration for specific web-apps	

```
element web-app-deploy {
  archive-directory?
  expand-cleanup-fileset?
  expand-directory?
  expand-prefix?
  expand-suffix?
  path?
  redeploy-check-interval?
  redeploy-mode?
  require-file*
  startup-mode?
  url-prefix?
  versioning?
  web-app-default*
  web-app*
}
```

31.302 Overriding web-app-deploy configuration

The web-app-deploy can override configuration for an expanded war with a matching <web-app> inside the <web-app-deploy>. The <root-directory> is used to match web-apps.

Example: resin.xml overriding web.xml

```
<resin xmlns="http://caucho.com/ns/resin">
<cluster id="">
<host id="">

<web-app-deploy path="webapps">
  <web-app context-path="/wiki"
             root-directory="wiki">
    <context-param database="jdbc/wiki">
```

```
  </web-app>
</web-app-deploy>

</host>
</cluster>
</resin>
```

31.303 versioning

The versioning attribute of the <web-app-deploy> tag improves web-app version updates by enabling a graceful update of sessions. The web-apps are named with numeric suffixes, e.g. foo-10, foo-11, etc, and can be browsed as /foo. When a new version of the web-app is deployed, Resin continues to send current session requests to the previous web-app. New sessions go to the new web-app version. So users will not be aware of the application upgrade.

31.304 <web-resource-collection>

Specifies a collection of areas of the web site for security purposes. See admin/security-overview.xtp for an overview.

Table 31.178: <web-resource-collection> Attributes

ATTRIBUTE	DESCRIPTION
web-resource-name	a name for a web resource collection
description	
url-pattern	url patterns describing the resource
http-method	HTTP methods to be restricted.
method	

```
element web-resource-collection {
  url-method*
  & http-method*
  & web-resource-name?
}
```

31.305 <welcome-file-list>

Sets the files to use as when no filename is present in url. According to the spec, each file is in a <welcome-file> element.

```
element welcome-file-list {
  string
  | welcome-file*
}
```

Example: WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.xtp</welcome-file>
    <welcome-file>home.xtp</welcome-file>
  </welcome-file-list>
</web-app>
```

Resin also provides a shortcut where you can just list the files:

Example: WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <welcome-file-list>
    index.jsp, index.xtp, home.xtp
  </welcome-file-list>
</web-app>
```

31.306 <work-dir>

<work-dir> configures a work directory for automatically generated code, e.g. for JSP, PHP, and JPA classes.

```
element work-dir {
  string
}
```