Stingray™ Traffic Manager: Java Development Guide

Version 9.9

January 2015

riverbed®

Riverbed Technology
680 Folsom St.
San Francisco, CA 94107

Phone: 415-247-8800
Fax: 415-247-8801
Web: http://www.riverbed.com

Part Number
712-00157-01

Contents

CHAPTER 1 Java Development

This guide describes the Java features available in Stingray Traffic Manager that allow the user to embed Java Extensions in TrafficScript code, extending its capabilities with a potentially enormous library of available code.

Introduction

Java is a platform-independent, object orientated programming language that has a large community of developers, libraries and applications. Stingray Traffic Manager supports the use of Java Extensions in TrafficScript, offering greater flexibility in traffic manipulation.

Extensions are modules that extend the functionality of virtual servers, working in a similar way to TrafficScript rules. Java Extensions are based on the Java Servlet API[1], which is a widely used API that can generate server responses.

Using Java Extensions in TrafficScript makes easily available to offer functions like:

1. **Content processing:** Improved XML/HTML processing using specialized Java libraries.
2. **Additional libraries:** ISV libraries supplied as value-add solution.
3. **Authentication:** by using RADIUS/TACACS/LDAP etc.

Available Features

The following standard features can be easily added using Java Extensions:

- **Light Weight Directory Access Protocol support**

  LDAP is an Internet protocol that provides access to the information on a server, usually to look up personal contact information and additional data such as encryption certificates, pointers to printers, etc.

- **Active Directory support**

  Active Directory support provides authentication, authorization and lets Administrators apply their policies to networks.

- **RADIUS support**

  RADIUS (Remote Authentication Dial in User Service) is a specialized Internet protocol used to control access to the network. It provides easy authentication, authorization and accounting.

[1] http://java.sun.com/products/servlet/2.2/javadoc/

- **SQL Database interface support**

  SQL is the standard programming language for querying and managing databases. It is supported by Oracle, Microsoft and mySql, amongst others.

- **SOAP support**

  SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built. Stingray Traffic Manager SOAP based Control API that allows other application to manage, query and control it.

- **TACACS support**

  Terminal Access Controller Access-Control System is an authentication protocol, mostly used in UNIX-like systems, that allows encrypted communication with a remote server.

- **Threading**

  Java code can run 'in the background', not just as a request-response code.

- **UDP communication**

  User Datagram Protocol is an Internet protocol that allows programs located on networked computers to communicate with other computers. These short messages are usually referred to as datagrams.

- **Advanced XML and HTML processing**

  XML provides the gateway for advanced formatting and data-exchanging between different types of devices.

- **Persistence of resources between requests**

  This is linked with Session persistence offered as a standard feature. Refer to the Stingray User Manual for more information on how to maintain persistence between requests.

- **Sessions using cookies**

  Cookies are an easy way to identify the user, provide customization and allow for session persistence, when needed.

CHAPTER 2 Configuring Java

Requirements

Note: In order to use Java Extensions, you must install the Java run-time environment (JRE) version 6 (also known as 1.6) or later on the server hosting the Stingray software. Stingray appliance variants come with Java pre-installed.

To download the latest version of the Java Runtime Environment, visit: **http://www.java.com/getjava/**

Stingray Traffic Manager is available in a variety of software and appliance configurations. Java support is an optional feature that is license key activated. It is not available in Stingray Load Balancer.

How Java Extensions Work

To use a Java Extension, it must be called from a TrafficScript rule using the `java.run()` function. This makes a call to a process called the "Java Extension Runner". The runner maintains instances of all the Extensions uploaded in memory, and passes information from TrafficScript to them when a call to `java.run()` is made.

Setting Up Stingray Traffic Manager

To use Java code in TrafficScript, first you may need to configure how Java is run.

To specify a Java runtime executable, go to the **System > Global Settings > Java Extension Runner** section of the interface. Then enable the Java radio button and set the **java!command** field to the name of the executable (and the path if it is not on the systems default search path), along with any command line options Java should be run with. By default **java!command** is set to `java -server`.

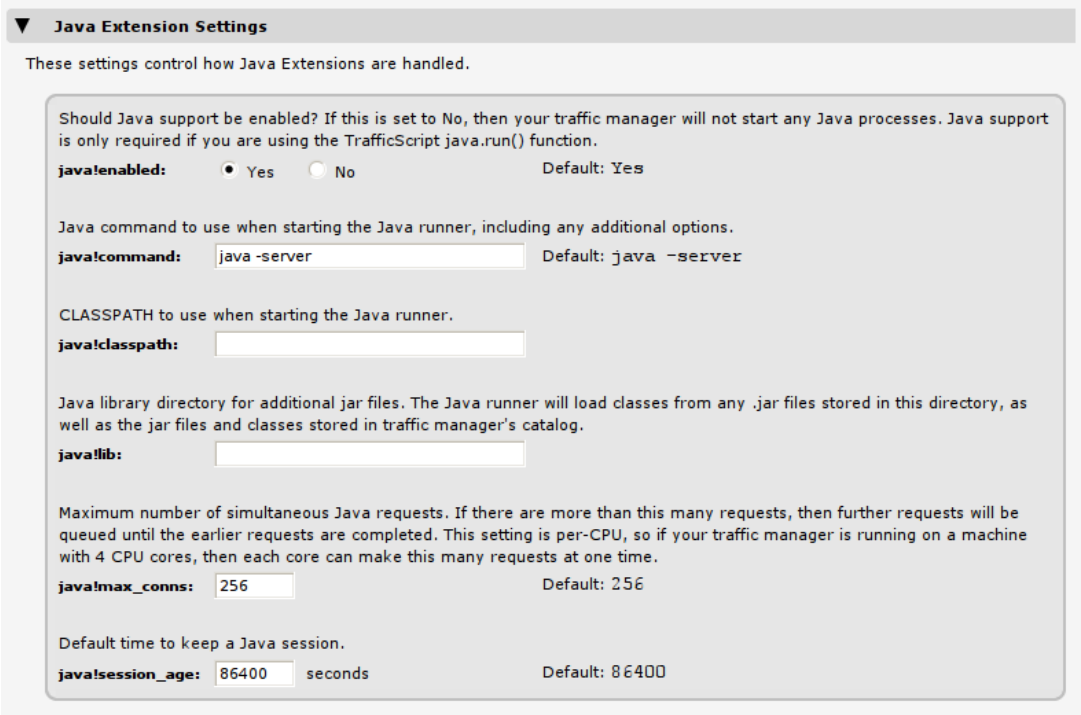

Fig. 1. Main settings for an extension

Also under "Global settings" you will find these Java-related fields:

- **java!lib** (optional): this setting identifies the system location where third-party Java jar files are located, such as `/usr/share/java`.

  All Java classes in this folder will be searched when the Java Extension runner starts up, and whenever this setting is modified.

- **java!classpath** (optional): this can specify a list of jar files that should be searched when the Java Extension runner starts up.

  This setting can be used to identify individual jar files that are not located in **java!lib**.

To check the setup, press the **Diagnose** button, and verify that the "Java Extensions" section reports no errors. Java Extensions are now ready to run.

CHAPTER 3 Writing a Java Extension

Java Extensions can generate server responses, modify requests to backend servers or alter responses from other servers. To use the Servlet API you must create a Java class that extends either the `GenericServlet` or one of its sub-classes, such as the `HttpServlet` class.

A simple HTTP Servlet might look like this:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet
{
   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      res.setContentType( "text/plain" );
      PrintWriter out = res.getWriter();
      out.println( "Hello World!" );
   }
}
```

This is a standard Servlet that just prints "Hello World!" whenever it is used.

The method **`doGet()`** is overridden from `HttpServlet` and is called whenever a HTTP GET request/response (depending if the Java Servlet is called in a TrafficScript response/request rule) is received. There is an identical function called `doPost` which does the same for HTTP POST messages.

Note: By default the `doGet`/`doPost` method will return the error "HTTP method POST/GET is not supported by this URL". If you don't want this error these methods must be overridden.

Stingray Extensions to the Servlet API

The HTTP Servlet specification states that the `doGet()` and `doPost()` methods are passed two arguments, an `HttpServletRequest` object (commonly named '`req`') and an `HttpServletResponse` object (commonly named '`res`'). Stingray Traffic Manager's implementation passes in two sub-classed objects of type `ZXTMHttpServletRequest` and `ZXTMHttpServletResponse`.

These implementations have several additional fields and methods to access the additional capabilities in the Java Extensions API. To use these fields and methods, you will need to cast the `req` and `res` objects to the Stingray subtype:

```
import   java.io.*;
import   javax.servlet.ServletException;
import   javax.servlet.http.*;
import   com.zeus.ZXTMServlet.*;
```

```
public class MyServlet extends HttpServlet
{
   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      ZXTMHttpServletRequest zreq = (ZXTMHttpServletRequest) req;
      ZXTMHttpServletResponse zres = (ZXTMHttpServletResponse) res;
      // Base Servlet API does not need to provide the ability to
      // query the content type of a response
      if( zres.getContentType() == null ||
          !zres.getContentType().equalsIgnoreCase("text/html") )
      {
         return; // We're not interested in non-html data
      }

      // Count how many times a html page has been sent
      // with SNMP counter 1
      zreq.incrementCounter( 1 );

      BufferedReader in = zres.getReader();
      PrintWriter out = zres.getWriter();

      String current = null;
      while( ( current = in.readLine() ) != null ) {
         if( current.indexOf( "<title>" ) != -1 ) {
            current = "<title>My New Title</title>";
         }
         out.println( current );
      }
   }
}
```

This example increments a counter every time a request for a HTML page is made. The counter value can be viewed through the traffic manager's SNMP interface. The Servlet also alters the content of the HTML page, changing its title to "My New Title".

The example highlights one of the main differences between standard Servlets and Stingray Java Extensions; Stingray Java Extensions have the ability to manipulate data received from other sources (in this case a web-server), whereas a normal Servlet is designed to only produce data. Since this Extension alters data in a response, it should be run from a TrafficScript Response.

Riverbed provides Javadoc-style documentation for the Traffic Manager's extensions to the Servlet API. See "Javadoc Documentation" on page 24 for more information.

Modifying Responses and Writing Data From a Java Extension

Generally, it is possible to run several Java Extensions or TrafficScript rules to process a response before the response is written back to the client.

However, once a Java Extension begins modifying the response data (for example, using `PrintWriter.println()` as above), data will start to be streamed back to the client. At this point, the HTTP headers are flushed to the client and it is not possible to modify them further.

Only one Java Extensions may modify a response. You cannot modify response data using several Java Extensions in succession. You should ensure that you do not run any Java Extensions once the response data is written to the client.

Creating TrafficScript Functions Using Java Extensions

Java Extensions can be used to do more than just process traffic. You can use Java Extensions to implement new functions in TrafficScript, thus extending the TrafficScript language further.

The following extension implements the Soundex[2] algorithm, which converts a name or other string into a phonetic representation. You can use it to test if two words sound the same:

```
import java.io.IOException;
import javax.servlet.*;
import com.zeus.ZXTMServlet.ZXTMServletRequest;

public class Soundex extends GenericServlet {
   private static final long serialVersionUID = 1L;

   public void service( ServletRequest req, ServletResponse res )
      throws IOException
   {
      String[] args = (String[])req.getAttribute( "args" );
      String result = doSoundex( args[0] );
      ((ZXTMServletRequest)req).setConnectionData( "soundex", result );
   }

   static String soundex = "01230120022455012623010202";
   String doSoundex( String s ) {
      s = s.toUpperCase();
      StringBuilder r = new StringBuilder();

      char last = '0';
      if( s.length() > 0 ) last = s.charAt( 0 );
      r.append( last );
      for( int i = 1; i < s.length(); i++ ) {
         int j = s.charAt( i )-'A';
         char next = ( j >= 0 && j < soundex.length() ) ?
            soundex.charAt( j ) : '0';
         if( next != last && next != '0' ) {
            r.append( next ); last = next;
         }
         if( r.length() >= 4 ) break;
      }
      while( r.length() < 4 ) r.append( '0' );
      return r.toString();
   }
}
```

[2] http://en.wikipedia.org/wiki/Soundex

This extension is based on the simple GenericServlet API. It should be called from TrafficScript as follows:

```
java.run( "Soundex", $word );
```

Note that Java Extensions cannot return values in the traditional functional sense.

The extension inspects its first argument and applies the Soundex algorithm. The Servlet API does not provide a means for a servlet to return a value to its caller; you need to use the `ZXTMServletRequest.setConnectionData()` method to set a local connection variable that the TrafficScript rule can then look up.

The following TrafficScript request rule, assigned to run every time on a simple client-first virtual server, illustrates the use of this Java Extension:

```
sub soundex( $word ) {
   java.run( "Soundex", $word );
   return connection.data.get( "soundex" );
}

$word = string.trim( request.getLine() );

request.sendResponse( "That sounds like " . soundex( $word ) . "\n" );
```

Here is the rule in use:

```
localhost:/$ telnet localhost 7
Trying 127.0.0.1…
Connected to localhost.localdomain.
Escape character is '^]'.
Stingray
That sounds like Z200
Interface
That sounds like I536
Internet
That sounds like I536
quit
That sounds like Q300
bye
That sounds like B000
exit
That sounds like E230
^]
telnet>
```

… and so on…

Compiling an Extension

To compile Java Extensions for use with Stingray Traffic Manager you will need:

1. Java Development Kit (JDK), which contains the Java compiler. This can be downloaded from http://java.sun.com.
2. Java Servlet API which can be found in the user interface by going to **Catalogs > Java Extensions Catalog** and clicking the **Java Servlet API** link.
3. Stingray Java Extensions API which are can be found in the user interface under **Catalogs > Java Extensions Catalog** and clicking the **Stingray Java Extensions API** link.

To compile an Extension:

1. Copy `servlet.jar` and `zxtm-servlet.jar` to the directory in which you are compiling.
2. Now run the command:

```
$ javac -cp servlet.jar:zxtm-servlet.jar MyServlet.java
```

This will create a class file called *MyServlet.class.*

You can also package up an Extension along with any other needed classes in a single JAR file. Stingray Traffic Manager will automatically search JAR files for Extensions to use.

Running an Extension

To utilise a Stingray Java Extension, you must first compile and upload the Extension.

To upload the Extension, go to the **Catalogs > Java Extensions Catalog** page and specify your class or jar file in the Upload section. Alternatively you can copy the file(s) to the `$ZEUSHOME/zxtm/conf/jars` directory.

Whenever a Java Extension is uploaded, a new TrafficScript rule is created. This rule contains the code `java.run("extension class name")`.

The Extension user interface page should then show your Extension under the Extensions **Catalog** section.

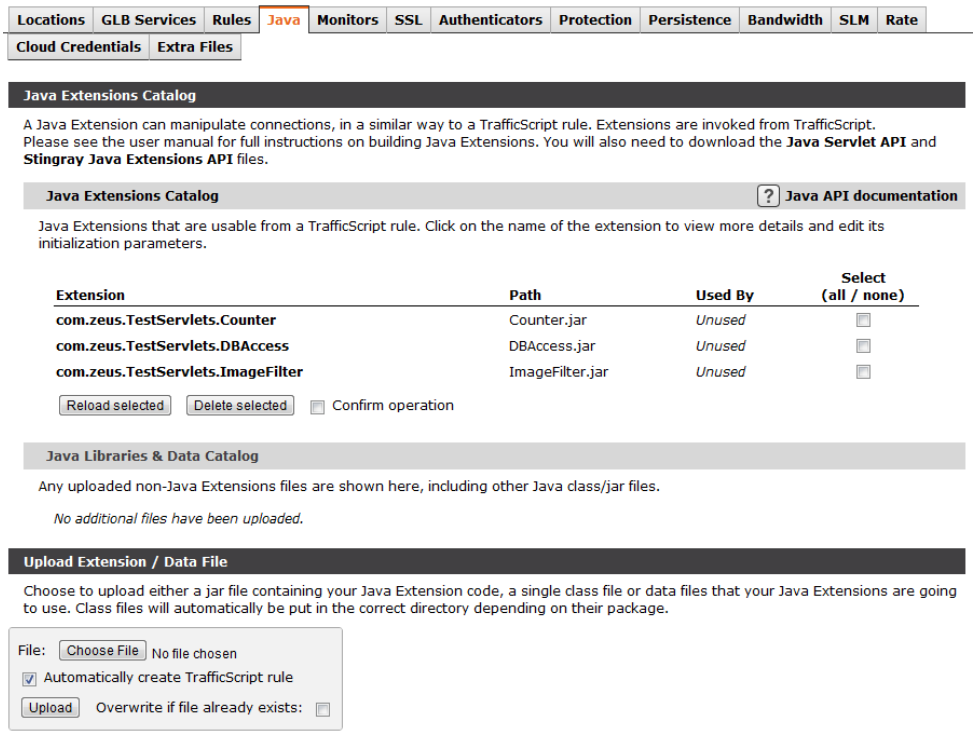

Fig. 2. Java Extensions' catalog and upload page

Unrecognized Extensions

If an Extension is shown in the Libraries and Data catalog, it means that it has not been recognized as an Extension. The Extension will probably be listed as "Invalid" along with an error message detailing why it is failing to load. Ensure that the class extends GenericServlet (or a subclass of GenericServlet, such as HttpServlet) and that any JAR libraries required to run the Extension have been uploaded, or are present in the java!lib directory specified on the Global Settings page.

Replacing Old Extensions

Extensions are cached in memory when they are being used. If you replace an Extension with an updated version by copying directly into the traffic manager configuration instead of using the management interface, you may need to tell the traffic manager to reload the new Extension. To do this go to the Extension catalog, select your Extension, check confirm and click "Reload selected". Reloading causes the Extension Runner to unload your Servlet from memory, so any information it was storing will be lost.

Extensions are not applied directly to virtual servers, they must be called from within rules. You are given the option to create a default Rulebuilder rule when uploading the Extension which allows you to easily use the Extension in a virtual server. Alternatively you can run your Extension from TrafficScript using the function:

```
java.run( "MyPackage.MyServlet" );
```

Extension Parameters

You can also pass parameters to an Extension when it is run:

```
java.run( "MyPackage.MyServlet", "Hey there!" );
```

The Extension can access these parameters using the following code:

```
String[] args = (String[]) req.getAttribute( "args" );
```

You can also specify parameters through the Extensions catalog (use **Catalogs > Extensions** and click on the Extension you want to edit). These parameters are set every time an Extension is run, and therefore are useful for defining global settings. They can be accessed from inside your Extension using:

```
ServletConfig config=getServletConfig();

String param = config.getInitParameter( "param_name" );
```

Debugging Java Extensions

Printing Debug Information

A simple way to view information about a running Extension is to print statements detailing the Extensions status.

To print out debugging information you can use the log function (a member function of GenericServlet):

```
log( "Hello log!" );
```

This will output the string `"Hello log!"` to the main event log, where it will appear as follows:

```
INFO: MyPackage.MyServlet: Hello log!
```

Exceptions

Java exception stack traces are useful ways of telling where your code is failing. The main `doGet`/`doPost` functions can only throw `IOException` or `ServletException` (and any type of `RuntimeException`), so it's a good idea to catch exceptions and either print a sensible error or for debugging print a stack trace to the log.

This example shows how to catch an exception and write its stack to the log:

```
public void doGet( HttpServletRequest req, HttpServletResponse res )
   throws ServletException, IOException
{
   try {

      throw new IOException( "Hello" );

   } catch( Exception e ) {
      res.sendError( 500, e.toString() );
```

```
      // Save stack trace as a string and print to the log
      StringWriter sw = new StringWriter();
      e.printStackTrace( new PrintWriter( sw ) );
      log( sw.toString() );
   }
}
```

Remote Debugging

Java has a remote debugging facility that allows you to use a Java debugger on an Extension running on the traffic manager. In this example we will use Eclipse but any Java debugger that supports remote debugging can be used.

Setting up Stingray Traffic Manager to Accept Debugging

To set up Stingray Traffic Manager to accept debugging connections go to **System->Global Settings->Java Extension Runner** and append the following line to the end of the **java!command** setting the following content:

```
-Xdebug -Xnoagent
   -Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
```

(this code must be entered in a single line)

The 'address=' part option sets the port on which the Java runner listens for debugging connections, and can be set to whatever port you choose.

After applying this setting the Java Extension Runner will restart and print the following log line:

```
WARN: Java: Listening for transport dt_socket at address:
8000
```

This message shows that Stingray Traffic Manager is now ready to receive debugging connections. Now all you need to do is point your Java debugger at the Stingray server on the correct port. For Eclipse:

1. Add your Extension code to a Java project and ensure it compiles correctly (if you are doing your Extension development in Eclipse you may already have done this).

2. Go to the debugger and select "Open Debug Dialog...".

3. Right click "Remote Java Application" and click "New".

4. Under "Connection Properties" enter traffic manager's host name and the port you set as the remote debugging port (e.g. 8000).

5. Click "Debug".

Eclipse will now connect to the Java Extension Runner process and the debugger can now be used as if the code was being run locally.

Remote debugging also has the ability to "Hot Swap" altered code into the running system, so altering and saving code in Eclipse will update the running Java Extension. Note however this change only lasts until the debugging session ends, you have to upload the new changes manually if you want them to be permanent.

CHAPTER 4 TrafficScript Functions in the Java API

Equivalent TrafficScript Functions in the Java API

This section details how to use the TrafficScript functionality in a Java Extension. You can find out more information on each Java function from the API documentation referenced from the Java Extensions Catalog page.

TrafficScript Functions	*Stingray Java Extensions API*
connection.close()/discard()	ZXTMServletResponse.dropConnection()
connection.getMemoryUsage()	No equivalent.
connection.getNode()	ServletRequest.getAttribute(“node”)
connection.get/setPersistence()	ServletRequest.get/setAttribute(“persistence”)
connection.getPool()	ServletRequest.getAttribute(“pool”)
connection.get/ setServiceLevelClass()	ServletRequest.get/setAttribute(“servicelevel”)
connection.getVirtualServer()	ServletRequest.getAttribute(“virtualserver”)
connection.setPersistenceKey()	ServletRequest.setAttribute(“persistencekey”)
connection.setPersistenceNode()	ServletRequest.setAttribute(“persistencenode”)
connection.sleep()	Thread.sleep()
connection.data.get/set()	ZXTMServletRequest.get/setConnectionData()
counter.increment()	ZXTMServletRequest.incrementCounter()
data.get/set()	ZXTMServletRequest.get/setData()
data.getMemoryUsage()	No equivalent.
data.reset()	No equivalent.
event.emit(event, message)	ZXTMServletRequest.emitEvent(event, message)
http.add/remove/setHeader()	ZXTMHttpServletRequest.add/remove/setHeader()
http.addResponseHeader()	HttpServletResponse.addHeader()

http.changeSite()	HttpServletResponse.sendRedirect()
http.doesFormParamExist()/ getFormParam()/ getQueryString()	HttpServletRequest.getQueryString()
http.getBody()	ServletRequest.getInputStream()/getReader()
http.getCookie()	HttpServletRequest.getCookies()
http.getHeader()/headerExists()	HttpServletRequest.getHeader()
http.getHeaderNames()	HttpServletRequest.getHeaderNames()
http.getHostHeader()	HttpServletRequest.getHeader()
http.getMethod()	HttpServletRequest.getMethod()
http.getMultipartAttachment()	Servlet can read the body data itself
http.getPath()	HttpServletRequest.getRequestURL()
http.getRawURL()	No equivalent.
http.getResponseBody()	ZXTMHttpServletResponse.getInputStream()/ getReader()
http.getResponseCode()	ZXTMHttpServletResponse.getStatus()
http.setResponseCode()	HttpServletResponse.setStatus()
http.getResponseCookie()	ZXTMHttpServletResponse.getCookies()
http.getResponseHeader()/ responseHeaderExists()	ZXTMHttpServletResponse.getHeader()
http.getResponseHeaderNames()	ZXTMHttpServletResponse.getHeaderNames()
http.getVersion()	ServletRequest.getProtocol()
http.normalisePath()	No equivalent.
http.redirect()	HttpServlet.sendRedirect()
http.removeCookie()/ setCookie()	ZXTMHttpServletRequest.removeCookie()
http.removeResponseCookie()	ZXTMHttpServletResponse.removeCookie()
http.removeResponseHeader()	ZXTMHttpServletResponse.removeHeader()

http.scrubRequest/ ResponseHeaders()	No equivalent.
http.sendResponse()	HttpServletResponse.sendError()
http.setBody()	ZXTMHttpServletResponse.getWriter().print()
http.setCookie()	HttpServletResponse.addCookie()
http.setIdempotent()	ServletRequest.setAttribute(“idempotent”)
http.setMethod()	ZXTMHttpServletRequest.setMethod()
http.setPath()	ZXTMHttpServletRequest.setRequestURI()
http.set/setRawQueryString()	ZXTMHttpServletRequest.setQueryString()
http.setResponseBody()	ServletResponse.getOutputStream()/getWriter()
http.setResponseCode()	HttpServletResponse.setStatus()
http.setResponseCookie()	HttpServletResponse.addCookie()
http.stream.startResponse()	HttpServletResponse.addHeader(), followed by HttpServletResponse.setStatus()
http.stream.readResponse()/ readBulkResponse()	ZXTMHttpServletResponse.getInputStream()/ getReader()
http.stream.writeResponse()	ServletResponse.getOutputStream().write()/ ServletResponse.getWriter().print()
http.stream.finishResponse()	Servlet completes
http.stream.continueFromBackend	No equivalent
http.cache.disable()/enable()	ServletRequest.getAttribute(“cache”)
http.cache.setKey()	ServletRequest.getAttribute(“cachekey”)
http.compress.disable()/enable()	ServletRequest.getAttribute(“compress”)
http.request.get()/head()/post()	Use built in Java functions.
geo.getCity()	ZXTMServletRequest.geoGetCity()
geo.getCountry()	ZXTMServletRequest.geoGetCountry()
geo.getCountryCode()	ZXTMServletRequest.geoGetCountryCode()
geo.getDistanceKM()	ZXTMServletRequest.geoGetDistanceKM()

geo.getDistanceMiles()	ZXTMServletRequest.geoGetDistanceMiles()
geo.getIPDistanceKM()	ZXTMServletRequest.geoGetIPDistanceKM()
geo.getIPDistanceMiles()	ZXTMServletRequest.geoGetIPDistanceMiles()
geo.getLatitude()/getLongitude()	ZXTMServletRequest.geoGetLatLon()
geo.getRegion()	ZXTMServletRequest.geoGetRegion()
geo.getRegionCode()	ZXTMServletRequest.geoGetRegionCode()
lang.*	Use built in Java functions.
math.*	Use built in Java functions.
net.dns.resolveHost()/IP()	**java**.net.InetAddress.getByName()/ getHostName()
pool.activeNodes()	ZXTMServletRequest.getActiveNodes()
pool.select()/use()	ServletRequest.setAttribute(“pool”) and ServletRequest.setAttribute(“proxy”)
rate.getBacklog()	ZXTMServletRequest.getRateBacklog()
rate.use()	No equivalent.
request.avoidNode()	ServletRequest.setAttribute(“avoidnodes”)
request.endsAt()/endsWith()	No equivalent
request.get()/getLine()	ZXTMServletRequest.getInputStream()/ getReader()
request.get/setBandwidthClass()	ServletRequest.get/setAttribute(“bandwidth”)
request.getDestIP()/Port()	No equivalent.
request.getLength()	No equivalent.
request.getLocalIP()/Port()	ServletRequest.getAttribute(“dstip”/”dstport”)
request.get/setRemoteIP()/Port()	ServletRequest.get/setAttribute(“srcip”/”srcport”)
request.getRetries()	ServletRequest.getAttribute(“retries”)
request.get/setToS()	ServletRequest.get/setAttribute(“tos”)
request.getFD()	No equivalent.
request.isResendable()	ServletRequest.getAttribute(“resendable”)

request.retry()	ZXTMServletRequest.retry()
request.sendResponse()	ServletResponse.getOutputStream()/getWriter()
request.set()	ZXTMServletRequest.getOutputStream()/ getWriter()
resource.exists()/get()/getMD5()/ getMTime()	Use Java built in functions.
response.append()	Not provided, Extension can read/write its own response
response.close()	ZXTMServletResponse.dropConnection()
response.flush()	ServletResponse.getOutputStream().flush()
response.get()/getLine()	ZXTMServletResponse.getInputStream()/ getReader()
response.get/setBandwidthClass()	ServletRequest.get/ setAttribute(“serverbandwidth”)
response.getLength()	No equivalent.
response.getLocalIP()/Port()	ServletRequest.getAttribute(“serverdstip” / ”serverdstport”)
response.getRemoteIP()/Port()	ServletRequest.getAttribute(“serversrcip” / ”serversrcport”)
response.get/setToS()	ServletRequest.get/setAttribute(“servertos”)
response.getFD()	No equivalent.
response.set()	ServletResponse.getOutputStream()/getWriter()
rule.getName()	ServletRequest.get/setAttribute(“rule”)
rule.getState()	ZXTMServletResponse.isResponseRule() - 'true' if this is in a response rule
slm.conforming()	ZXTMServletRequest.getSLMConforming()
slm.isOK()	ZXTMServletRequest.isSLMOK()
slm.threshold()	ZXTMServletRequest.getSLMThreshold()
ssl.clientCert*()	ServletRequest.getAttribute(“javax.servlet.request.X509Certificate”)

ssl.clientCipher()	ServletRequest.getAttribute(“javax.net.ssl.cipher_suite”)
ssl.isSSL()	ServletRequest.getAttribute(“SSL_PROTOCOL”) is set.
ssl.sslSessionID()	ServletRequest.getAttribute(“SSL_SESSIONID”)
string.*	Use built in Java functions.
sys.*	Use built in Java functions.
xml.*	Use built in Java functions.

Attributes Listing

Attributes are parameters that can be used with the ServletRequest.get/setAttribute() methods to view and alter the connection information.

Attribute	*Description*
args	Any arguments passed to the Servlet from TrafficScript
avoidnodes	Space separated list of nodes to avoid with the load balancing
bandwidth	Get/set the bandwidth class to use
cache	Set to 0 for http.cache.disable(), 1 for http.cache.enable()
cachekey	Set the cache key.
compress	Set to 0 for http.compress.disable(), 1 for 'default' and 2 for http.compress.enable() .
dstip	Destination IP address (i.e. the address on the traffic manager).
dstport	Destination port (i.e. the port on the traffic manager) .
idempotent	Get/set whether this request is idempotent - 0/1 for no/yes
node	Node used by this connection - readable by a response rule only.
persistence	Get/set the persistence class to use .

persistencekey	Set the persistence data to use for universal session persistence
persistencenode	Set the node to persist on
pool	Get/set the pool to use
proxy	Set the machine to forward proxy on to. This should be set to IP:Port (i.e. the servlet does any DNS lookup)
resendable	Read-only: is the request is resendable to another node. Valid in response rules only. Returns 0/1
retries	Read the number of retries - request.getRetries()
rule	The name of the TrafficScript rule that called this extension.
serverbandwidth	Get/set the bandwidth class to use for server-side data
serverdstip	The IP address the server (node) sent to.
serverdstport	The port the server (node) sent to.
serversrcip	The IP address of the server (node).
serversrcport	The server's (node's) source port.
servertos	IP Type-Of-Service flag to use for the server connection - takes same args as request.getToS()
servicelevel	Get/set the SLM class to use
srcip	Client's IP address
srcport	Client's source port
tos	IP Type-Of-Service flag to use for the client connection
virtualserver	Read the virtual server name

CHAPTER 5 Further Resources

Javadoc Documentation

Riverbed provides Javadoc-style documentation for the Traffic Manager's extensions to the Servlet API. To view this documentation, see the **Catalogs > Java** page of the Admin UI. From here, click the **Java API Documentation** link.

Stingray Manuals

The Traffic Manager includes an *Installation and Getting Started Guide*, intended to get you up and running quickly, and a more detailed *User Manual*. Riverbed also provides full reference manuals for functionality such as TrafficScript and the product APIs.

You can access these manuals from the Riverbed Support website at:

https://support.riverbed.com

Online Help

By clicking the **Help** button on any page of the Admin Server interface, you can see detailed help information for that page. You can also view contents and index pages to navigate around the online help.

Information Online

Product specifications can be found at:

http://www.riverbed.com/products-solutions/products/application-delivery-stingray/

Visit the Riverbed Splash community website for further documentation, examples, white papers and other resources:

http://splash.riverbed.com

Technical References on Java

You can find useful information about Java and the Extensions technology in the following selected links:

- Java Servlet Documentation:

  http://java.sun.com/products/servlet/

- Extensions tutorials and essentials:

  http://www.servlets.com/

- Eclipse's site:

  http://www.eclipse.org/downloads/

CHAPTER 6 Index