



Zeus Extensible Traffic Manager Java Development Guide

Version 5.0

Zeus Technology Limited
The Jeffreys Building
Cowley Road
Cambridge CB4 0WS
United Kingdom

Zeus Technology
5201 Great America Parkway Suite 320
Santa Clara
CA 95054
United States

UK: +44 (0)1223 525000
US: +1-408-850-7204
Email: info@zeus.com
Web: <http://www.zeus.com/>

Copyright Notice

© Zeus Technology Limited 2008. Copyright in this documentation belongs to Zeus Technology Limited. All rights are reserved. This documentation may not be reproduced in whole or in part in any manner or form (including photocopying or storing it in any medium by electronic means and whether or not transiently or incidentally to some other use of this documentation) other than in accordance with any applicable license agreement or with the prior written consent of Zeus Technology Limited. Any copies of this documentation must incorporate this notice.

Zeus Technology, the Zeus logo, Zeus Extensible Traffic Manager, ZXTM, TrafficScript, TrafficCluster and RuleBuilder are trademarks of Zeus Technology Limited. Other trademarks used may be owned by third parties.

© Java™ is a registered trademark of Sun Microsystems.

Table of Contents

Copyright Notice	2
Table of Contents	3
1. Java development	4
1.1 Introduction.....	4
1.2 Available features	4
1.3 Configuring Java.....	6
1.3.1 Requirements.....	6
1.3.2 How Java Extensions work in ZXTM	6
1.4 Writing a Java Extension.....	8
1.4.1 Compiling an Extension.....	9
1.5 Running an Extension.....	10
1.6 Debugging Java Extensions	12
1.6.1 Printing debug information	12
1.6.2 Exceptions	12
1.6.3 Remote debugging	13
2 TrafficScript functions in the Java API	15
2.1 Equivalent TrafficScript functions in the Java API	15
2.2 Attributes Listing	19
3 Further Resources	21
3.1 ZXTM Manuals.....	21
3.2 Information online	21
3.3 Technical references on Java	21
Index	23



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

I. Java development

I.1 Introduction

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

Extensions are modules that extend the functionality of ZXTM's virtual servers, working in a similar way to TrafficScript rules (for more information on rules, see chapter 7. 'TrafficScript Rules' in the ZXTM User Manual). Java Extensions are based on the Java Servlet API, which is a widely used API that can generate server responses.

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

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

I.2 Available features

The following standard features can be easily added to ZXTM 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.

- **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. ZXTM supports a SOAP 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 by ZXTM as a standard feature. Refer to the ZXTM 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.



1.3 Configuring Java

1.3.1 Requirements



In order to use Java with ZXTM, you must install the Sun Java run-time environment (JRE) version 1.5 or later. The ZXTM Appliances come with a compatible version of Java already installed.

To download the latest version of the Java Runtime Environment, visit:

<http://www.java.com/getjava/>

Zeus Extensible 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 ZXTM Load Balancer.

1.3.2 How Java Extensions work in ZXTM

To use a Java Extension in ZXTM, 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 to ZXTM in memory, and passes information from TrafficScript to them when a call to **java.run()** is made.

Setting up ZXTM

To use Java code in TrafficScript, first you may need to configure how ZXTM runs Java.

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".

Java Extension Settings

These settings alter how the Java Extensions are handled.

Should Java support be enabled? If this is set to No, then ZXTM will not start any Java processes. Java support is only required if you are using the TrafficScript `java.run()` function.

java!enabled: Yes No Default: Yes

Java command to use when starting the Java runner, including any additional options.

java!command: Default: `java -server`

CLASSPATH to use when starting the Java runner.

java!classpath: Default:

Java library directory for additional jar files. The Java runner will load classes from any .jar files stored in this directory, as well as the jar files and classes stored in ZXTM's catalog.

java!lib: Default:

Maximum number of simultaneous Java requests. If there are more than this many requests, then further requests will be queued until the earlier requests are completed. This setting is per-CPU, so if ZXTM is running on a machine with 4 CPU cores, then each core can make this many requests at one time.

java!max_conns: Default: 256

Default time to keep a Java session.

java!session_age: seconds Default: 86400

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`.

ZXTM will search all Java classes in this folder 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. ZXTM is now ready to run Java Extensions.

I.4 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 by ZXTM. 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.

When the Servlet is run by ZXTM the arguments **req** and **res**, which represent the HTTP request & response, are extended to include additional functions. To use these extra functions, cast **req** and **res** to `ZXTMHttpServletRequest` and `ZXTMHttpServletResponse` respectively:

```
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 allow the querying of an existing content
type
    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 ZXTM'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 ZXTM specific Servlets; ZXTM Servlets 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.

Refer to the ZXTM Java help pages for more information on all the available functions related to ZXTMHttpServletRequest and ZXTMHttpServletResponse.

1.4.1 Compiling an Extension

To compile Java Extensions for use with ZXTM 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. ZXTM Java Extensions API which are can be found in the user interface under **[Catalogs > Java Extensions Catalog]** and clicking the **ZXTM 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. ZXTM will automatically search JAR files for Extensions to use.

1.5 Running an Extension

To utilise an Extension within ZXTM, you must first compile and upload the Extension first.

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 to ZXTM, 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.

✓ File 'H:\SERVLETS\SLM.jar' was uploaded successfully.

Java Extensions Catalog

A Java Extension can manipulate connections to ZXTM, in a similar way to a TrafficScript rule. Extensions are invoked from TrafficScript. Please see the user manual for full instructions on building Java Extension. You will also need to download the **Java Servlet API** and **ZXTM Java Extensions API** files.

[Java API documentation](#)

Java Extensions that are usable from a TrafficScript rule. Click on the name of the extension to view more details and edit its initialization parameters.

Extension	Path	Used By	Select (all / none)
TestServlets.Counter	H:\SERVLETS\Counter.jar	Unused	<input type="checkbox"/>
TestServlets.PubChooser	H:\SERVLETS\PubChooser.jar	Unused	<input type="checkbox"/>
TestServlets.SLM	H:\SERVLETS\SLM.jar	Unused	<input type="checkbox"/>

Confirm operation

Java Libraries & Data Catalog

Any uploaded non-Java Extensions files are shown here, including other Java class/jar files.

No additional files have been uploaded.

Upload Extension / Data File

Choose to upload either a jar file containing your Java Extension code, a single class file or data files that your Java Extensions are going to use. Class files will automatically be put in the correct directory depending on their package.

File:

Automatically create TrafficScript rule

Overwrite if file already exists:

Fig. 2. Java Extensions' catalog and upload page

Unrecognized Extensions

If an Extension is shown in the Libraries and Data catalog, it means that ZXTM does not recognize it 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 are uploaded to ZXTM or are present in the java!lib directory specified on the Global Settings page.

Replacing old Extensions

ZXTM caches Extensions in memory when they are used. If you replace an Extension with an updated version by copying it to ZXTM instead of using the management interface, you may need to tell ZXTM 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" );
```

1.6 Debugging Java Extensions

1.6.1 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 ZXTM's main log, where it will appear as follows:

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

1.6.2 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() );
    }
}
```

1.6.3 Remote debugging

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

Setting up ZXTM to accept debugging

To set up ZXTM 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 -Xrunjdp: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 ZXTM is now ready to receive debugging connections. Now all you need to do is point your Java debugger at the ZXTM 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 ZXTM'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 ZXTM's 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 Extension running on ZXTM. 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.

2 TrafficScript functions in the Java API

2.1 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 found, please refer to the Java Extensions Catalog page.

<i>TrafficScript Functions</i>	<i>ZXTM Java Extensions API</i>
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/setServiceLevelClasses()	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.
http.add/remove/setHeader()	ZXTMHttpRequest.add/remove/setHeader()
http.addResponseHeader()	HttpServletResponse.addHeader()
http.changeSite()	HttpServletResponse.sendRedirect()
http.doesFormParamExist()/getFormParam()/getQueryString()	HttpRequest.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()	ZXTMHttpResponse.getInputStream()/getReader()
http.getResponseCode()	ZXTMHttpResponse.getStatus()
http.setResponseCode()	HttpResponse.setStatus()
http.getResponseCookie()	ZXTMHttpResponse.getCookies()
http.getResponseHeader()/responseHeaderExists()	ZXTMHttpResponse.getHeader()
http.getResponseHeaderNames()	ZXTMHttpResponse.getHeaderNames()
http.getVersion()	ServletRequest.getProtocol()
http.normalisePath()	No equivalent.
http.redirect()	HttpServlet.sendRedirect()
http.removeCookie()/setCookie()	ZXTMHttpServletRequest.removeCookie()
http.removeResponseCookie()	ZXTMHttpResponse.removeCookie()
http.removeResponseHeader()	ZXTMHttpResponse.removeHeader()
http.scrubRequest/ResponseHeaders()	No equivalent.
http.sendResponse()	HttpResponse.sendError()
http.setBody()	ZXTMServletRequest.getOutputStream()/getWriter()
http.setCookie()	ZXTMHttpServletRequest.setCookie()

http.setIdempotent()	ServletRequest.setAttribute("idempotent")
http.setMethod()	ZXTMHttpRequest.setMethod()
http.setPath()	ZXTMHttpRequest.setRequestURI()
http.set/setRawQueryString()	ZXTMHttpRequest.setQueryString()
http.setResponseBody()	ServletResponse.getOutputStream()/getWriter()
http.setResponseCode()	HttpServletResponse.setStatus()
http.setResponseCookie()	HttpServletResponse.addCookie()
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.
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.sessionID()	ServletRequest.getAttribute("SSL_SESSIONID")
string.*	Use built in Java functions.
sys.*	Use built in Java functions.
xml.*	Use built in Java functions.

2.2 Attributes Listing

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

<i>Attribute</i>	<i>Description</i>
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 <code>http.cache.disable()</code> , 1 for <code>http.cache.enable()</code>
cachekey	Set the cache key.
compress	Set to 0 for <code>http.compress.disable()</code> , 1 for 'default' and 2 for <code>http.compress.enable()</code> .
dstip	Destination IP address (i.e. the address on ZXTM) .
dstport	Destination port (i.e. the port on ZXTM) .
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
persistencecode	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

3 Further Resources

3.1 ZXTM Manuals

Bundled with the software is a Getting Started guide, intended to get you up and running quickly with the software. If you have purchased a ZXTM Appliance, you will also find a specific Appliance Quick-Start guide which you should read before installing and configuring the appliance for the first time.

There are also full manuals for the TrafficScript rules language and the ZXTM Control API.

You can access these manuals via the Help pages, or download the most recent versions from the ZXTM KnowledgeHub at <http://knowledgehub.zeus.com/>.

3.2 Information online

Product specifications can be found at:

<http://www.zeus.com/products/>

Visit ZXTM's KnowledgeHub for further documentation, examples, white papers and other resources:

<http://knowledgehub.zeus.com/>

3.3 Technical references on Java

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

- Java page on extensions:

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

- JSP, extensions, ad other programming resources:

<http://resources.corejsp.com/>

- Extensions tutorials and essentials:

<http://www.servlets.com/>

- Personal page on Java full of interesting articles, tutorials and resources:

http://www.frontiernet.net/~imaging/servlets_intro.html



- Eclipse's site:

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

Index

Debugging			
setting up ZXTM for	13		
Equivalent TrafficScript functions in the Java			
API	15		
Java			
Calling extensions from TrafficScript	6		
Compiling an extension	10		
configuration.....	6		
Extensions debugging	12		
Extensions debugging exceptions	12		
Overview.....	4		
Printing debug information.....	12		
Remote debugging exceptions	13		
Running an extension.....	10		
Technical references	21		
		Technical requirements	6
		Java API additional information	9
		Java extensions	
		Parameters.....	12
		Writing	8
		Java Extensions	
		Available features.....	4
		List of Java extensions' attributes	19
		Replacing old extensions	11
		Unrecognized extensions	11
		Writing Java extensions.....	8
		ZXTM	
		User manuals.....	21



