53-1005042-01
22 May 2017

Brocade Virtual Traffic Manager: Java Development Guide

Supporting 17.2

Brocade Communications Systems, Incorporated

Corporate and Latin American Headquarters
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134
Tel: 1-408-333-8000
Fax: 1-408-333-8101
E-mail: info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems China HK, Ltd.
No. 1 Guanghua Road
Chao Yang District
Units 2718 and 2818
Beijing 100020, China
Tel: +8610 6588 8888
Fax: +8610 6588 9999
E-mail: china-info@brocade.com

European Headquarters
Brocade Communications Switzerland Sàrl
Centre Swissair
Tour B - 4ème étage
29, Route de l'Aéroport
Case Postale 105
CH-1215 Genève 15
Switzerland
Tel: +41 22 799 5640
Fax: +41 22 799 5641
E-mail: emea-info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems Co., Ltd. (Shenzhen WFOE)
Citic Plaza
No. 233 Tian He Road North
Unit 1308 – 13th Floor
Guangzhou, China
Tel: +8620 3891 2000
Fax: +8620 3891 2111
E-mail: china-info@brocade.com

Contents

Preface

Read this preface for an overview of the information provided in this guide. This preface includes the following sections:

- "Document Conventions," next
- "Brocade Resources" on page 3
- "Document Feedback" on page 3
- "Contacting Brocade Technical Support" on page 3

Document Conventions

The document conventions describe text formatting conventions, command syntax conventions, and important notice formats used in Brocade technical documentation.

Notes and Warnings

Note, important, and caution statements might be used in this document. They are listed in the order of increasing severity of potential hazards.

Note: A Note provides a tip, guidance, or advice, emphasizes important information, or provides a reference to related information.

Important: An Important statement indicates a stronger note, for example, to alert you when traffic might be interrupted or the device might reboot.

Caution: A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

Text Formatting Conventions

Text formatting conventions such as boldface, italic, or Courier font may be used in the flow of the text to highlight specific words or phrases.

Format	Description
bold text	Identifies command names
	Identifies keywords and operands
	Identifies the names of user-manipulated GUI elements
	Identifies text to enter at the GUI
italic text	Identifies emphasis
	Identifies variables
	Identifies document titles
`Courier font`	Identifies CLI output
	Identifies command syntax examples

Command Syntax Conventions

Bold and italic text identify command syntax components. Delimiters and operators define groupings of parameters and their logical relationships.

Convention	Description
bold text	Identifies command names, keywords, and command options.
italic text	Identifies a variable.
value	In Fibre Channel products, a fixed value provided as input to a command option is printed in plain text. For example, **--show** WWN.
[ ]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ x | y | z }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options. In Fibre Channel products, square brackets may be used instead for this purpose.
x | y	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, for example, passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, member[member...].
\	Indicates a “soft” line break in command examples. If a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

Brocade Resources

Visit the Brocade website to locate related documentation for your product and additional Brocade resources.

White papers, data sheets, and the most recent versions of Brocade software and hardware manuals are available at www.brocade.com. Product documentation for all supported releases is available to registered users at MyBrocade. Click the **Support** tab and select **Document Library** to access documentation on MyBrocade or www.brocade.com. You can locate documentation by product or by operating system.

Release notes are bundled with software downloads on MyBrocade. Links to software downloads are available on the MyBrocade landing page and in the Document Library.

Document Feedback

Quality is our first concern at Brocade and we have made every effort to ensure the accuracy and completeness of this document. However, if you find an error or an omission, or you think that a topic needs further development, we want to hear from you. You can provide feedback in two ways:

- Through the online feedback form in the HTML documents posted on www.brocade.com.
- By sending your feedback to documentation@brocade.com.

Provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.

Contacting Brocade Technical Support

As a Brocade customer, you can contact Brocade Technical Support 24x7 online, by telephone, or by e-mail. Brocade OEM customers contact their OEM/Solutions provider.

Brocade Customers

For product support information and the latest information on contacting the Technical Assistance Center, go to www.brocade.com and select **Support**.

If you have purchased Brocade product support directly from Brocade, use one of the following methods to contact the Brocade Technical Assistance Center 24x7.

Online	Telephone	E-mail
Preferred method of contact for nonurgent issues: • Case management through the MyBrocade portal. • Quick Access links to Knowledge Base, Community, Document Library, Software Downloads and Licensing tools.	Required for Sev 1-Critical and Sev 2-High issues: • Continental US: 1-800-752-8061 • Europe, Middle East, Africa, and Asia Pacific: +800-AT FIBREE (+800 28 34 27 33) • Toll-free numbers are available in many countries. • For areas unable to access a toll free number: +1-408-333-6061	support@brocade.com Please include: • Problem summary • Serial number • Installation details • Environment description

Brocade OEM Customers

If you have purchased Brocade product support from a Brocade OEM/Solution Provider, contact your OEM/Solution Provider for all of your product support needs.

- OEM/Solution Providers are trained and certified by Brocade to support Brocade® products.
- Brocade provides backline support for issues that cannot be resolved by the OEM/Solution Provider.
- Brocade Supplemental Support augments your existing OEM support contract, providing direct access to Brocade expertise. For more information, contact Brocade or your OEM.
- For questions regarding service levels and response times, contact your OEM/Solution Provider.

CHAPTER 1 Java Development

This chapter provides overview information about Java development. This chapter contains the following sections:

About This Guide

The *Brocade Virtual Traffic Manager: Java Development Guide* describes the Java features available in the Brocade Virtual Traffic Manager (Traffic Manager).

The Traffic Manager allows you to embed Java Extensions in TrafficScript code, extending the Traffic Manager's capabilities with an extensive library of available Java code.

This guide introduces you to Java development, explains how to configure Java, how to write a Java extension, and explains how TrafficScript functions in the Java API.

Introduction

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

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

Note: For more information about Java Servlet technology, see http://java.sun.com/products/servlet/2.2/javadoc/.

Using Java Extensions in TrafficScript makes it easy to offer functions like the following:

- **Content processing:** Improved XML/HTML processing using specialized Java libraries.
- **Access to additional libraries:** ISV libraries supplied as value-add solution.
- **Authentication:** Achieved by using protocols such as RADIUS, TACACS, or LDAP.

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 allows Administrators to apply 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 a number of software companies including Oracle and Microsoft.
- **SOAP support**: SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP or HTTPS. SOAP forms the foundation layer of the Web services protocol stack, providing a basic messaging framework upon which abstract layers can be built.
- **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 that can run in the background, not just as a request-response code.
- **UDP communication**: User Datagram Protocol (UDP) is an Internet protocol that allows programs located on networked computers to communicate with other computers. UDP generates messages that are sometimes 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**: Resource persistence is linked with session persistence and is offered as a standard feature. See the *Brocade Virtual Traffic Manager: User's Guide* for more information about maintaining persistence between requests.
- **Sessions using cookies**: Cookies are an easy way to identify the user, provide customization, and allow for session persistence, when needed.

Java API Documentation

Brocade 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 the Java page, click the **Java API Documentation** link.

Java Technical References

For technical information about Java and the extensions technology, see the links listed below.

- Java Servlet documentation: http://java.sun.com/products/servlet/
- Extensions tutorials and essentials: http://www.servlets.com/
- Eclipse software site: http://www.eclipse.org/downloads/

CHAPTER 2

Configuring Java

This chapter contains information about configuring Java. This chapter contains the following sections:

- "Java Requirements," next
- "How Java Extensions Work" on page 9

Java Requirements

Note: 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 Traffic Manager software. Traffic Manager appliance variants come with Java preinstalled.

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

The Traffic Manager is available in a variety of software and appliance configurations. Java support might not be available on all product variants - contact your support provider for details.

How Java Extensions Work

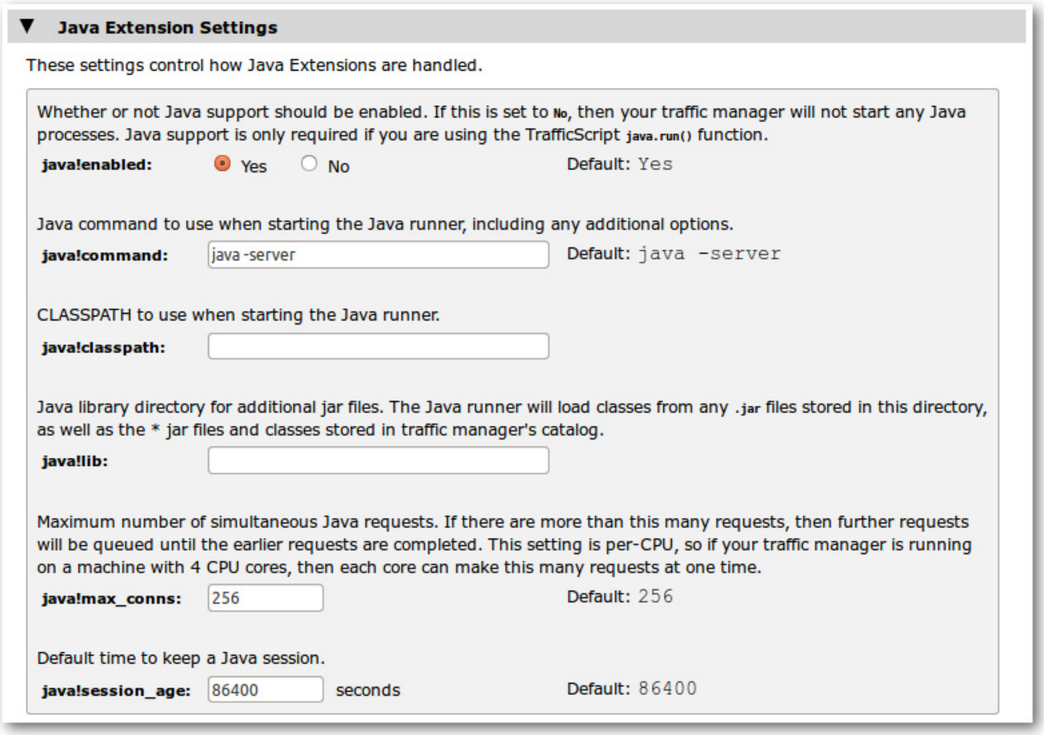
To use a Java Extension, it must be initiated from a TrafficScript rule using the java.run() function. The java.run() function initiates a process called the Java Extension Runner. The Java Extension Runner maintains instances of all the Extensions uploaded in memory, and passes that information from TrafficScript to the instances when the java.run() function is initiated.

Setting Up the Traffic Manager

To use Java code in TrafficScript, first configure how Java operates.

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

Figure 2-1. Main settings for an extension



▼ **Java Extension Settings**

These settings control how Java Extensions are handled.

Whether or not Java support should be enabled. If this is set to No, then your traffic manager 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: java -server Default: `java -server`

CLASSPATH to use when starting the Java runner.

java!classpath:

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 traffic manager's catalog.

java!lib:

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 your traffic manager is running on a machine with 4 CPU cores, then each core can make this many requests at one time.

java!max_conns: 256 Default: 256

Default time to keep a Java session.

java!session_age: 86400 seconds Default: 86400

Under Global Settings, you will find these Java-related fields:

- java!lib (optional): This setting identifies the system location where the 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, and whenever this setting is modified.

- java!classpath (optional): This setting can be used to specify a list of jar files that should be searched when the Java Extension runner starts.

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

To check the setup, click Diagnose, and verify that the Java Extensions section does not report any errors. If error free, the Java Extensions are now ready for use.

CHAPTER 3 Writing a Java Extension

This chapter describes the function of Java classes and Servlet APIs, describes how a Java Extension is used with a Servlet API, how to create Traffic Script functions using Java Extensions, and how to write (compile) and debug Java Extensions. This chapter contains the following sections:

- "Java Classes and Servlet APIs," next
- "Traffic Manager Extensions to the Servlet API" on page 12
- "Creating TrafficScript Functions Using Java Extensions" on page 13
- "Compiling a Java Extension" on page 15
- "Debugging Extensions" on page 17

Java Classes and Servlet APIs

Java Extensions 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 subclasses, such as the HttpServlet class.

Here's an example of a simple HTTP Servlet:

```
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 prints the phrase "Hello World!" whenever the Servlet is used.

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

Note: By default, the doGet/doPost methods display the error message "HTTP method POST/GET is not supported by this URL". To avoid this error, override the doGet/doPost methods.

Traffic Manager Extensions to the Servlet API

The HTTP Servlet specification states that the doGet() and doPost() methods are passed in two arguments, an HttpServletRequest object (commonly named req) and an HttpServletResponse object (commonly named res).

The Traffic Manager's implementation passes in two subclassed objects of type ZXTMHttpServletRequest and ZXTMHttpServletResponse.

These implementations have several additional fields and methods used to access the additional capabilities in the Java Extensions API. To use these fields and methods, cast the req and res objects to the Traffic Manager subtype, as shown below:

```
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 );
      }
   }
}
```

Note the following points about this example:

- This example increments a counter every time a request for an 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 Java Extensions. Specifically, 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 the Java Extension alters data in a response, run the Java Extension from a TrafficScript response.

The online help contains Javadoc-style documentation for the Traffic Manager's extensions to the Servlet API. To access online help, use one of the following methods:

- Access the Traffic Manager Admin Interface, and press the **Help** button.
- Click the Manuals button on the on the Tool bar.
- Select the **Java API Documentation** link.

Modifying Responses and Writing Data From a Java Extension

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 the PrintWriter.println() function), data is streamed back to the client. At this point, the HTTP headers are flushed to the client and you cannot make any more modifications to the HTTP headers.

Only one Java Extension may modify a response. You cannot modify response data using several Java Extensions in succession. 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.

The following Java Extension implements the Soundex algorithm, which converts a name or other string into a phonetic representation.

Note: For more information about Soundex, see http://en.wikipedia.org/wiki/Soundex.

You can use the algorithm to test if two words sound the same, as shown in the following example:

```
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();
   }
}
```

This Java Extension is based on the simple GenericServlet API. The Java Extension should be called (initiated) from TrafficScript, as shown below:

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

Note: Java Extensions cannot return values in the traditional sense.

The Java Extension inspects the first argument and then applies the Soundex algorithm.

The Servlet API does not provide a way 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 use.

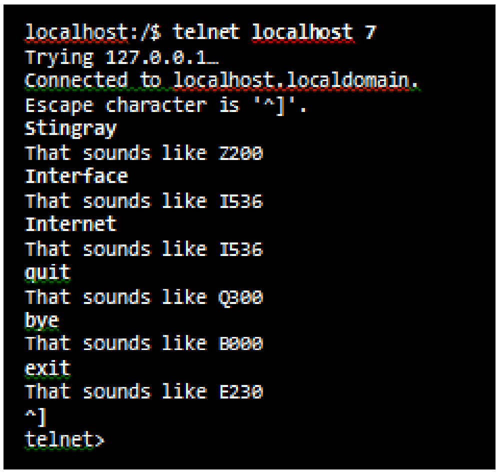
The following TrafficScript request rule, assigned to run every time on a simple client-first virtual server, illustrates the use of this Java Extension:

```
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" );
```

The following illustrates 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>
```

Compiling a Java Extension

To compile Java Extensions for use with the Traffic Manager, you need the following items:

- Java Development Kit (JDK), which contains the Java compiler. This can be downloaded from http://java.sun.com.
- 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.
- Java Extensions API, which can be found in the user interface under Catalogs > Java Extensions Catalog and clicking the Traffic Manager Java Extensions API link.

To compile a Java Extension, complete the following steps:

1. Copy the servlet.jar and zxtm-servlet.jar files to the directory where you are compiling the Java Extension.

2. Run the following command:

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

   This creates a class file called MyServlet.class.

Note: You can bundle a Java Extension with any other needed classes in a single .jar file. The Traffic Manager automatically searches .jar files for the Java Extensions to use.

Running a Java Extension

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

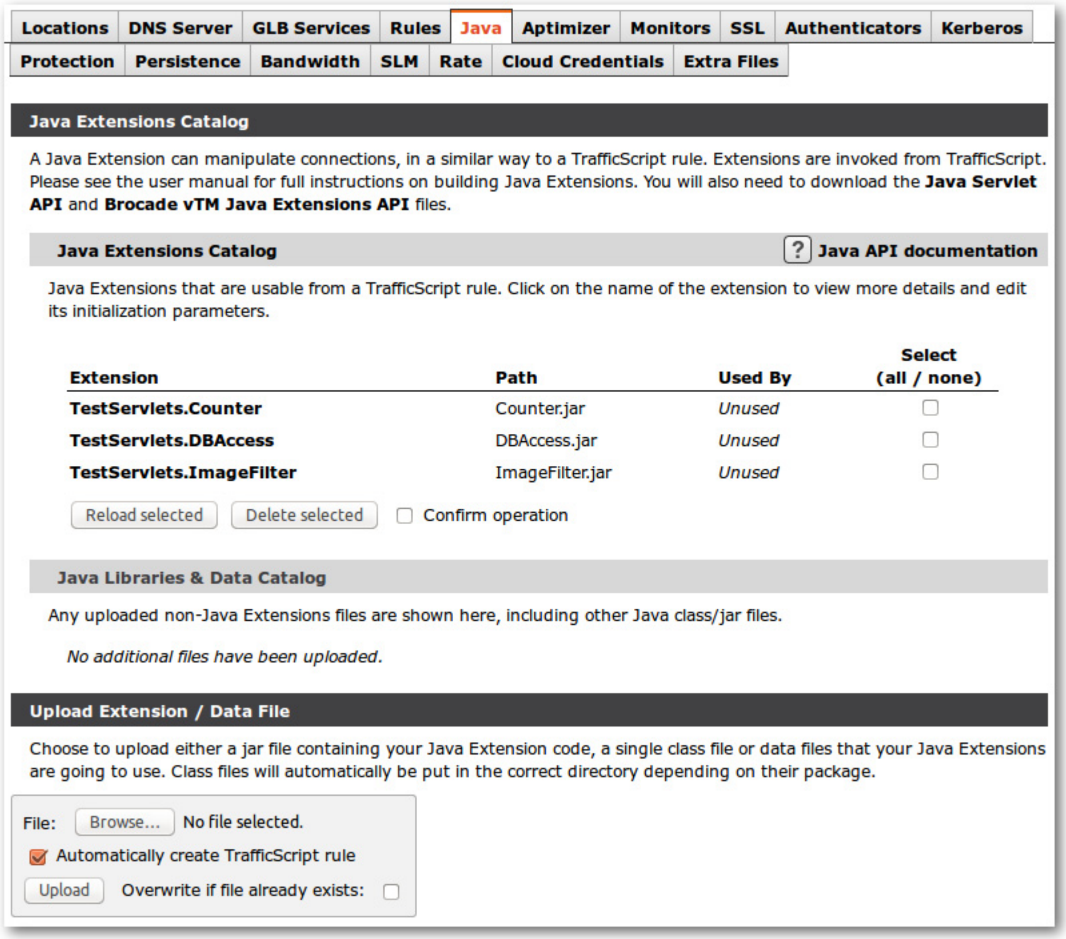
To upload the Java Extension, go to the **Catalogs > Java Extensions Catalog** page of the user interface, and specify the class or .jar file in the Upload section.

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. The new TrafficScript rule contains the code java.run (using the "extension class name" convention).

The Extension user interface page should then show your Java Extension in the Java Extensions Catalog section, as shown below.

Figure 3-1. Java Extensions Catalog Page


Locations | DNS Server | GLB Services | Rules | Java | Aptimizer | Monitors | SSL | Authenticators | Kerberos
Protection | Persistence | Bandwidth | SLM | Rate | Cloud Credentials | Extra Files

Java Extensions Catalog

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

Java Extensions Catalog ? 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	Counter.jar	*Unused*	☐
TestServlets.DBAccess	DBAccess.jar	*Unused*	☐
TestServlets.ImageFilter	ImageFilter.jar	*Unused*	☐

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: Browse... No file selected.

☑ Automatically create TrafficScript rule

Upload Overwrite if file already exists: ☐

Unrecognized Extensions

If an Extension is listed in the Java Libraries & Data Catalog section, it means that the Extension has not been recognized. The Extension will be listed as invalid, along with an error message detailing why it failed to load.

Ensure that the class extends the 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 Extensions

Extensions are cached in memory when they are being used. If you replace an Extension with an updated version by copying the Extension 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**.

Note: Reloading the Extension causes the Extension Runner to remove your Servlet from memory, so any information the Servlet was storing will be lost.

Extensions are not applied directly to virtual servers; Extension must be called (initiated) from within rules. You can create a default Rulebuilder rule when uploading the Extension, which allows you to 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 pass parameters to an Extension by using the following command:

```
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 by using the **Catalogs** > **Extensions** menu and selecting the Extension you want to edit. These parameters are set every time an Extension is used, and, therefore, are useful for defining global settings. The parameters can also be accessed from inside your Extension using the following command:

```
ServletConfig config=getServletConfig();

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

Debugging Extensions

Printing Debug Information

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

To print debugging information, you can use the log function (a member function of GenericServlet) as shown below:

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

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

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

Java Exception Stack Traces

Java exception stack traces are useful for identifying where your code is failing. The main doGet/doPost functions can only “throw” (send) IOExceptions or ServletExceptions (and any type of RuntimeException). Be sure to address code failures before proceeding. Print an error report that you can use for debugging the code or print a stack trace to the event log.

This example shows how to “catch” an exception and write its stack to the event 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.

Setting up the Traffic Manager to Accept Debugging

Note: In this section, Eclipse is used but any Java debugger that supports remote debugging can be used.

To set up the Traffic Manager to accept debugging connections, complete the following steps:

1. Go to the Catalogs > Java Extensions Catalog page of the user interface.

2. Add the following line to the end of java!command, setting the following content:

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

Note: This code must be entered on a single line.

The *address* value sets the port on which the Java runner “listens” for debugging connections. You can set this port to whatever port you choose.

After applying these settings, the Java Extension Runner restarts and prints the following line:

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

This message shows that the Traffic Manager is now ready to receive debugging connections. Next, “point” your Java debugger at the Traffic Manager server on the correct port. To do this using Eclipse, complete the following steps:

1. Add your Extension code to a Java project and ensure that the project compiles correctly. (If you are developing your Extension in Eclipse, you may have already completed this step.)

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

3. Right-click Remote Java Application and click **New**.
4. Under Connection Properties, enter the Traffic Manager's hostname and the port you set as the remote debugging port (for example, 8000).
5. Click **Debug**.

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

Remote debugging also has the ability to "hot swap" altered code into the running system. Therefore, altering and saving code in Eclipse also updates the Java Extension in use.

Note: However, this change only lasts until the debugging session ends. You have to upload the changes manually if you want them to be permanent.

CHAPTER 4

TrafficScript Functions in the Java Extension API

This chapter details how to use the TrafficScript functionality in a Java Extension. This chapter includes the following sections:

- "Equivalent TrafficScript Functions in the Java Extension API" on page 21
- "Attributes List" on page 25

Note: To find more information about Java, see "Java API Documentation" on page 6.

Equivalent TrafficScript Functions in the Java Extension API

This table lists TrafficScipt functions and the equivalent Java Extension API.

TrafficScript Function	Java Extension 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()

TrafficScript Function	Java Extension API
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()

TrafficScript Function	Java Extension API
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”)

TrafficScript Function	Java Extension API
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”)

TrafficScript Function	Java Extension API
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 List

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

Attribute	Description
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